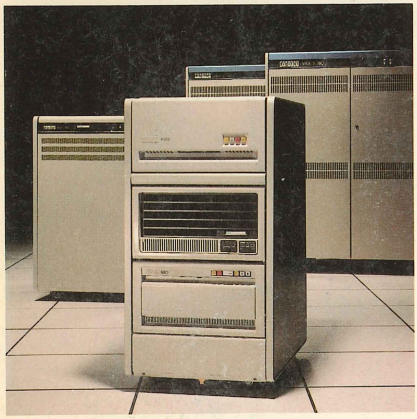
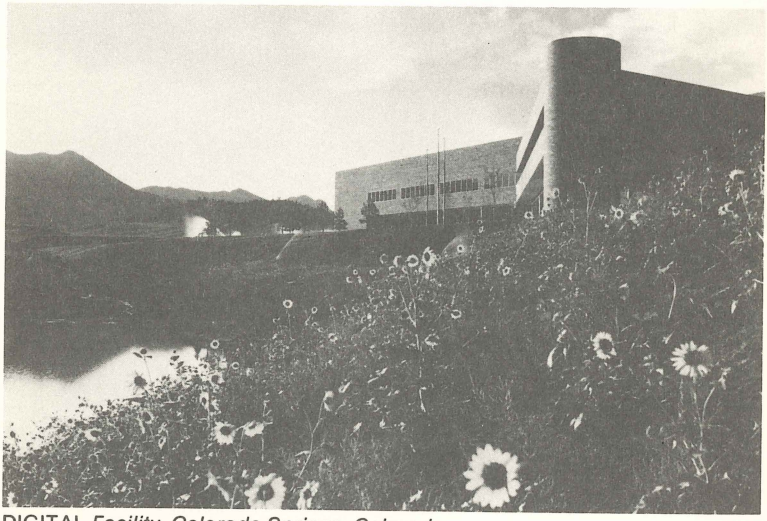
VAX



HARDWARE HANDBOOK

digital



DIGITAL *Facility, Colorado Springs, Colorado*

CORPORATE PROFILE

Digital Equipment Corporation designs, manufactures, sells and services computers and associated peripheral equipment, and related software and supplies. The Company's products are used worldwide in a wide variety of applications and programs, including scientific research, computation, communications, education, data analysis, industrial control, timesharing, commercial data processing, word processing, health care, instrumentation, engineering and simulation.

G. Rees

HARDWARE HANDBOOK

digital

Digital Equipment Corporation makes no representation that the interconnection of its products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting of license to make, use, or sell equipment constructed in accordance with this description.

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

This handbook was designed, produced, and typeset
by DIGITAL's New Products Marketing Group
using an in-house text-processing system.

TABLE OF CONTENTS

PREFACE

VAX is DIGITAL's family of 32-bit minicomputers. This handbook provides a brief introduction to VAX and detailed descriptions of the VAX family members: the new VAX-11/730, the VAX-11/750, the VAX-11/780, and the new VAX-11/782.

PART I introduces the reader to VAX with an overview of the VAX architecture and the capabilities of the newly enhanced VAX/VMS operating system. To complete the VAX family picture, a hardware overview of the VAX processors is provided.

For detailed information on the new entry-level VAX, PART II of this book describes the VAX-11/730 console subsystem, the central processing unit, the main memory subsystem, the UNIBUS subsystem, and the privileged registers.

PART III covers the VAX-11/750, including the topics listed above and a chapter on the MASSBUS subsystem.

Both the VAX-11/780 and the VAX-11/782 are discussed in PART IV. Additional chapters for the VAX-11/780 include Chapter 15 on the Synchronous Backplane Interconnect and Chapter 19 on Interconnects and the VAX-11/782.

PART V details the wide range of dependability features built into VAX computer systems. These features were designed to ensure VAX reliability, availability, and maintainability.

PART VI of this book contains 13 appendices, a glossary, and an index for your convenience.

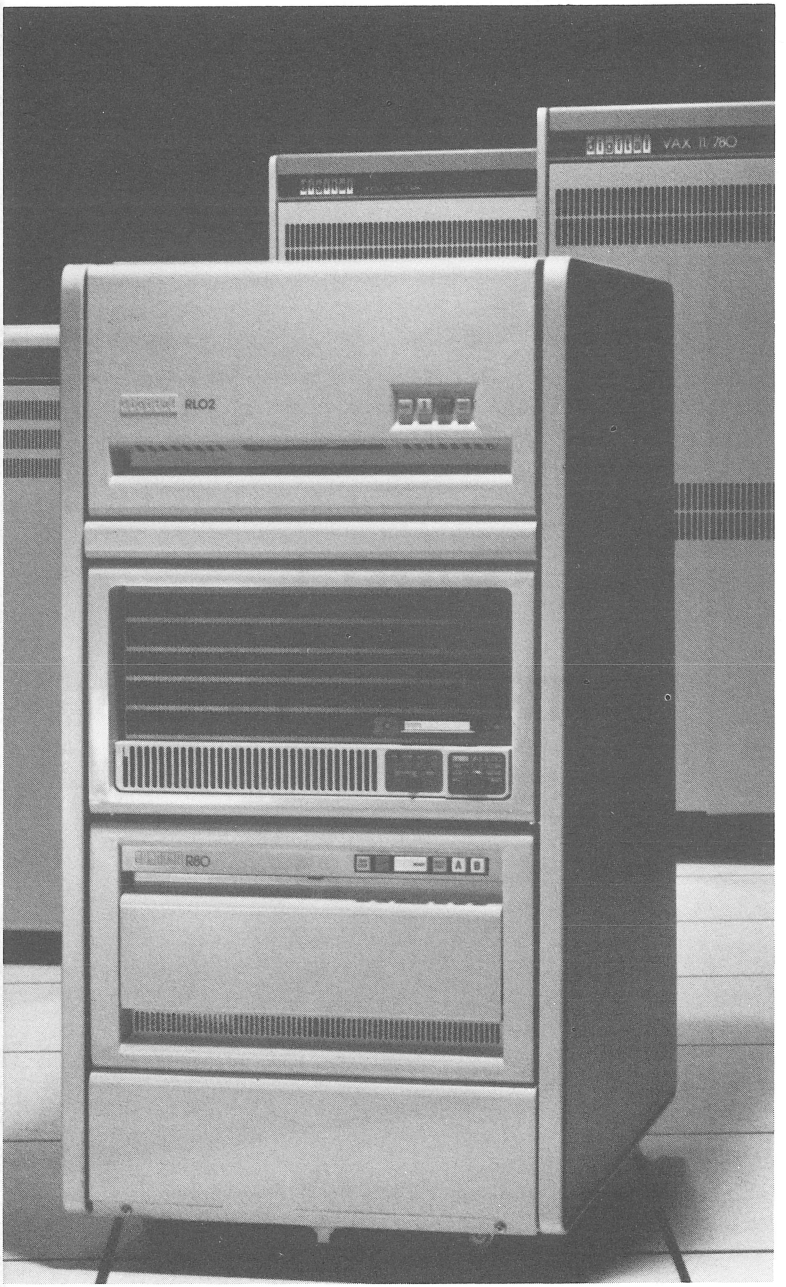
In addition to this handbook, two other VAX handbooks are available:

- The *VAX Architecture Handbook*, describing the VAX system architecture, addressing modes, and the native mode instruction set
- The *VAX Software Handbook*, describing the VAX/VMS operating system, its operation, hardware interaction, data structures, features, and capabilities

As with all VAX handbooks, a comment card has been placed in the back of the book. All comments are greatly appreciated, as they help us make the VAX handbook set meet your needs.

PART I

INTRODUCTION


RLO2
R80
A
B
VAX 11/780

CHAPTER 1
AN INTRODUCTION TO VAX

THE VAX FAMILY

VAX is DIGITAL's family of 32-bit minicomputer systems. The new family members—the VAX-11/730 and the VAX-11/782 attached processor system—together with the VAX-11/750 and VAX-11/780, make the power of VAX systems available to a wide range of users, applications and budgets.

All VAX processors implement a 32-bit architecture, an extensive instruction set with numerous data types, and a 32-bit bus structure for high throughput. All VAX system hardware is complemented by the newly enhanced VAX/VMS operating system, a powerful multiprogramming operating system that handles multiuser, realtime and multistream batch applications, plus online program development.

The newest member of the VAX family, the VAX-11/730, incorporates bit-slice and Programmed Array Logic (PAL) technology. Like the other family members, the VAX-11/730 implements the VAX architecture and runs the VAX/VMS operating system and layered software. With the VAX-11/730, however, VAX functionality is available at a much lower price, providing the ability to move VAX power down to the project or section level. The VAX-11/730 can also be used as a powerful, remote DECnet node, allowing its users access to higher performance members of the VAX family when necessary. Connection to mass storage devices and other peripherals is provided through a UNIBUS adapter.

The VAX-11/750, the mid-range member of the VAX family, incorporates many innovations designed to increase performance and to reduce the overall cost of ownership. The VAX-11/750 is the first 32-bit minicomputer to be implemented primarily in custom bipolar LSI Schottky logic (designed entirely by DIGITAL engineers). One UNIBUS adapter (integral to the processor) and up to three MASSBUS adapters or one additional UNIBUS and two MASSBUS adapters may be used for connection to mass storage devices and other peripherals.

The VAX-11/780 was designed to meet the needs of many users with large databases and extensive processing needs. Central to its I/O system is a 32-bit wide data and control path that can move up to 13.3 MB of data per second among the system's major hardware components. Up to four UNIBUS and four MASSBUS adapters may be used for connection to mass storage devices and other peripherals. The support of high-performance disks and tapes by the VAX-11/750,

VAX-11/780 and VAX-11/782, combined with their ability to network with the other VAX family members provides significant and varied configuration possibilities.

The VAX-11/782 attached processor computer system is a tightly-coupled asymmetrical multiprocessor system that can provide up to 1.8 times the performance of a single VAX-11/780 system for compute-intensive applications. Consisting of two VAX-11/780 CPUs, the VAX-11/782 attached processor computer system can support up to 8 MB of MA780 shared memory.

NOTE

Specific information on the VAX-11/782 attached processor system is contained in Chapter 19, Interconnects and the VAX-11/782. Unless otherwise specified, most of the VAX-11/780 features in this chapter apply to the VAX-11/782.

Application Performance

VAX hardware and software were designed to complement each other. Hardware implementation combined with the VAX/VMS operating system, 32-bit addressing, a 4 billion byte virtual memory, an address translation buffer, a prefetch instruction buffer, an optional floating point accelerator, and the powerful VAX instruction set, give VAX systems their impressive performance.

The impressive CPU power and throughput, plus the high performance-to-cost ratio, make VAX systems ideal for interactive applications. The high computational ability and large program size mean VAX systems can handle tough realtime applications as well. Furthermore, the VAX/VMS operating system provides extensive facilities for good batch performance—including job control, multistream, spooled input and output, operator control, conditional command branching, and accounting functions. A choice of options such as additional physical memory, user control store, and additional peripheral equipment interfaces, allow even greater flexibility in configuring systems to optimize performance for specific applications.

Ease of Use

VAX systems are user-oriented systems designed for easy operation. The DIGITAL Command Language (DCL) interface used by VAX/VMS is easy to learn and is suitable for both interactive and batch environments. The software compatibility of VAX systems allows software developed for one VAX system to run on another VAX system without modification. Because VAX systems use the same instruction set, it

also means that users need not learn a new series of instructions to take full advantage of another VAX system's capabilities.

VAX/VMS provides extensive system management facilities, giving system managers and operators the tools necessary to control the system configuration and the operations of system users for maximum efficiency. Users will appreciate the extensive HELP commands and complete multiuser security. The VAX family processors also implement a PDP-11 Compatibility Mode which recognizes almost all PDP-11 instructions. This allows users to execute code written for the PDP-11 with few modifications.

The VAX console subsystem also contributes to ease of use. A separate console terminal replaces the traditional toggle switches and lights, and a carefully designed console command language lets the user perform operations such as EXAMINEs and DEPOSITs, or boot the system, using simple commands. This console terminal also provides a hardcopy record for complete documentation of console transactions. Furthermore, switches on the front panel of the CPU can be set up to reboot the system automatically, with no operator intervention, in the event of a power failure or system crash.

Additionally, VAX systems are designed to facilitate rapid, low-cost applications development. With the complete set of VAX/VMS development tools, file system features, optional information management products, and other software packages, applications are easier to develop and require far less debugging time. DIGITAL's extensive educational services are also available to train and assist users in exploiting the wide and varied capabilities of VAX systems.

Easy Installation and Maintenance

A variety of system configurations is available so customers can purchase exactly what is required. VAX systems are easily tuned and adapted allowing additional peripherals and options to be interfaced at any time. Customers may choose from a wide variety of peripherals and packaging options to configure a VAX system to suit their requirements—whether the site is an office, a laboratory, or an industrial setting.

Once the system is installed, extensive reliability, availability, and maintainability features (discussed in Part V of this book) in both the hardware and the software ensure data integrity and increase system uptime. Features such as ECC (Error Correcting Code) memory, online error logging, and a complete range of online and stand-alone diagnostics verify system integrity and help ensure proper system

operation. The Remote Diagnosis option for the VAX-11/750 and VAX-11/780 allows a customer to be directly linked to a DIGITAL Diagnostic Center for diagnosis of hardware and software failures. For VAX-11/730 users, customer runnable diagnostics allow a system user the capability of verifying proper hardware operation and the quick isolation of system failures to the subsystem or device level. The Remote Support option, utilizing Remote Diagnosis technology, provides the DIGITAL service engineer with a further level of technical resources.

Sound Long-Term Investment

The features of the VAX series described above, together with the many other features described in the chapters that follow, make VAX systems a sound long-term investment. The new VAX-11/730 and VAX-11/782 systems are a reflection of DIGITAL's ongoing commitment to the VAX family of 32-bit minicomputers and further proof that the VAX family exemplifies the architecture of the 1980s. The wide range of systems possible with the VAX-11/730, VAX-11/750, VAX-11/780,and VAX-11/782 ensures that these systems can be tailored to individual application requirements and can be easily reconfigured if those needs should change in the future—an important consideration for customers involved in long-term projects and implementations.

The following sections in this chapter will introduce the reader to a variety of VAX family architectural and software features, as well as to many of the important hardware features of the various VAX implementations. Appendix A in the back of this book contains a table of commonly used VAX family mnemonics.

ARCHITECTURE OVERVIEW

The VAX family architecture is characterized by a powerful and complete instruction set of 304 instructions (see Appendix B), a wide range of data types, an efficient set of addressing modes, full demand paging memory management, and a very large virtual address space of over 4 billion bytes.

The VAX Native Instruction Set is an extension of the PDP-11 instruction set. Instructions can be grouped into classes based on their functions and uses:

1. Instructions to manipulate arithmetic and logical data types. These include integer, floating point, packed decimal, character string, and bit field instructions.

   The data type identifies how many stored bits are to be treated as a unit and how the unit is to be interpreted. Data types that may be used are:

Data Type	Represented As
Integer	Byte (8 bits), word (16 bits), longword (32 bits), quadword (64 bits)
Floating point	4-byte F_floating, 8-byte D_floating, 8-byte G_floating, 16-byte H_floating
Packed decimal	String of bytes (up to 31 decimal digits, 2 digits per byte)
Character string	String of bytes interpreted as character codes (up to 64 KB); a numeric string is a character string of codes for decimal numbers (up to 31 digits)
Bits and bit-fields	Field length is arbitrary and is defined by the programmer (0 to 32 bits in length)

Integer, floating point, packed decimal, and character data are stored starting on an arbitrary byte boundary. Bit and bit field data start on an arbitrary bit boundary. A collection of data structures can be packed together to use less storage space.

2. Instructions to manipulate special kinds of data. These include queue manipulation instructions (i.e., those that insert and remove queue entries), address manipulation instructions, and user-programmed general register load and save instructions. These instructions are used extensively by the VAX/VMS operating system.
3. Instructions to control basic program flow. These include BRANCH, JUMP, and CASE instructions, subroutine CALL instructions, and procedure CALL instructions.
4. Instructions to perform special operating system functions quickly. These include process control instructions (such as two special context switching instructions which allow process context variables to be loaded and saved using only one instruction for each operation), and the FIND FIRST instruction which (among other uses) allows the operating system to locate the highest priority executable process. These instructions contribute to rapid and efficient rescheduling.
5. Instructions provided specifically for high-level language constructs. During the design of the VAX family architecture, special attention was given to implementing frequently-used, high-level language constructs as single VAX instructions. These instructions contribute to decreased program size and increased

execution speed. Some of the constructs which have become single VAX instructions include:

— The FORTRAN-computed GOTO statement (translates into the CASE instruction).

— The loop construct (e.g., add, compare, and branch translates into the ACB instruction).

— An extensive CALL facility (which aligns the stack on a longword boundary, saves user-specified registers, and cleans up the stack on return); the CALL facility is used compatibly among all native mode languages and operating system services.

VAX instructions and data are variable in length. They need not be aligned on longword boundaries in physical memory, but may begin at any odd or even byte address. Therefore, instructions not requiring arguments use only one byte, while other instructions may take two, three, or up to 54 bytes depending on the number of arguments and their addressing modes. The advantage of byte alignment is that instruction streams and data structures can be stored in much less physical memory.

The VAX processors offer several addressing modes. Eleven of these use the general registers to identify the operand location and operate similarly to the PDP-11 addressing modes. The names of the modes are:

- Register
- Register deferred
- Autoincrement
- Autoincrement deferred
- Autodecrement
- Byte, word and longword displacement (similar to the PDP-11 index mode)
- Byte, word, and longword displacement deferred (similar to the PDP-11 index deferred mode)

The two additional addressing modes implemented by VAX family processors are:

- Indexed
- Literal

Because the VAX instruction set is so flexible, most functions require fewer instructions and less storage than on non-VAX processors. The result is more compact and efficient programs, faster program execution, faster context switching, more precise and faster math functions,

and improved compiler-generated code.

General Registers and Stacks — The VAX family CPUs provide sixteen 32-bit general registers which can be used for temporary storage, or as accumulators, index registers, and base registers. Registers R0 through R11 can be used as general purpose registers and the remaining four have special significance depending on the instruction being executed: Register 12 (the Argument Pointer); Register 13 (the Frame Pointer); Register 14 (the Stack Pointer); and Register 15 (the Program Counter).

Stacks are associated with the processor's execution state. The processor may be in a process context (in one of four modes: kernel, executive, supervisor, or user) or in the systemwide interrupt service context. A stack pointer is associated with each of these states. Whenever the processor changes from one state to another, Register 14 (the Stack Pointer) is updated accordingly.

For further information concerning the VAX architecture, refer to the VAX Architecture Handbook.

SOFTWARE OVERVIEW

VAX/VMS is the multiuser, multifunction operating system for the VAX family processors. The software compatibility between VAX family members means that users with any VAX system can run existing software on any other VAX system without having to make modifications. Furthermore, VAX/VMS provides a reliable, high performance environment for the concurrent execution of multiuser timesharing, batch, and time-critical applications. VAX/VMS also provides:

- Virtual memory management for executing large programs
- Event-driven priority scheduling
- Shared memory, file, and interprocess communication data protection based on ownership and application groups
- Programmed system services for process and subprocess control and interprocess communication

VAX/VMS uses the VAX memory management features to provide swapping, paging, protection, and sharing of both code and data. Memory is allocated dynamically. Applications can control the amount of physical memory allocated to executing processes, to the protection of pages, and to swapping; furthermore, these controls can be added after the application is implemented.

CPU time and memory residency are scheduled on a pre-emptive priority basis. Thus, time-critical processes do not have to compete with lower priority processes for scheduling services. Scheduling rotates among processes of the same priority.

VAX/VMS includes system services to control processes and process execution, control time-critical response, control scheduling, and obtain information. Process control services allow the creation of subprocesses as well as independent detached processes. Processes can communicate and synchronize using mailboxes, shared areas of memory, or shared files. A group of processes can also communicate and synchronize, using multiple common-event flag clusters.

Memory access protection is provided both between and within processes. Each process has its own independent virtual address space which can be mapped to private pages or shared pages. VAX/VMS uses the four processor access modes to read- and/or write-protect individual pages within a process. Protection of files, shared pages of memory, and interprocess communication facilities such as mailboxes and event flags, is based on user identification codes individually assigned to accessors and data.

A complete program development environment is offered and supported by VAX/VMS. It includes the native assembly language, VAX-11 MACRO, and the following high-level programming languages: VAX-11 FORTRAN, VAX-11 COBOL, VAX-11 BASIC, VAX-11 BLISS, VAX-11 PASCAL, VAX-11 PL/I, VAX-11 CORAL 66, VAX-11 C, VAX-11 DSM, and VAX-11 DIBOL. It provides the tools necessary to write, assemble or compile, and link programs, as well as to build libraries of source, object, and image modules.

VAX Information Management Facilities — VAX/VMS supports a set of software tools that provide a full range of information management capabilities. These information management products can be used to organize, maintain, retrieve and manipulate data quickly and easily. They include:

- VAX-11 FMS, a forms management system with interactive and language-callable video forms
- VAX-11 DATATRIEVE, a multifaceted data management facility that can store, update, and retrieve information and generate reports
- VAX-11 Common Data Dictionary, a central repository for data about data, allowing the information management tools to use a single set of data descriptions as a common resource
- VAX-11 RMS, a record management facility for sequential, relative, and multikey ISAM (Indexed Sequential Access Method) file organizations
- VAX-11 DBMS, a CODASYL-compliant database management system for creating, maintaining, and updating databases

The VAX/VMS on-disk structure provides a multiple-level hierarchy of named directories and subdirectories. Files can extend across multi-

ple volumes and can be as large as the volume set on which they reside. Volumes are mounted to identify them to the system. VAX/VMS also supports multivolume ANSI format magnetic tape files with transparent volume switching.

Communications — The variety of communications interfaces supported by the VAX/VMS operating system allows VAX systems to be connected to other VAX systems, other DIGITAL systems, and to other manufacturers' computer systems.

Synchronous, point-to-point, and multipoint connections are supported for interprocessor communication. For terminal to host communications, asynchronous connections are supported. (For further information, see the Terminals and Communications Handbook.)

Using DECnet communications software, various types of computer system networks can be constructed to facilitate remote communications, resource sharing, and distributed computation. DECnet-VAX Phase III software offers adaptive routing, task-to-task communications, network file transfer, network command terminals, and network resource sharing and network management capabilities. Other communications software supported by VAX/VMS includes:

- VAX-11 2780/3780 protocol emulator for transferring data between a VAX system and an IBM or other manufacturer's system by means of remote job entry protocols
- VAX-11 3271 protocol emulator for an interactive program-to-program link to an IBM host running CICS or IMS
- MUX200/VAX emulator for communication with a CDC6000, CYBER series or other host computer system capable of using 200UT mode 4A communications protocol
- VAX-11 PSI (Packetnet System Interface) for connection to a public data network using the X.25 communications protocol

For further information concerning VAX software and the VAX/VMS operating system, refer to the VAX Software Handbook or the VAX Technical Summary.

HARDWARE OVERVIEW

VAX computer systems consist of the central processing unit, the console subsystem, the main memory subsystem, and the input/output (I/O) subsystems—the UNIBUS, the MASSBUS (VAX-11/750, VAX-11/780, and VAX-11/782 only), and the DR780 (VAX-11/780 only) subsystems. The VAX hardware configurations are illustrated in Figures 1-1 through 1-4.

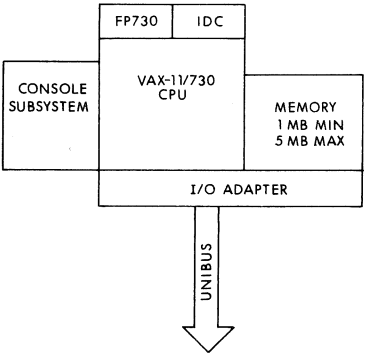


Figure 1-1 VAX-11/730 Hardware Configuration

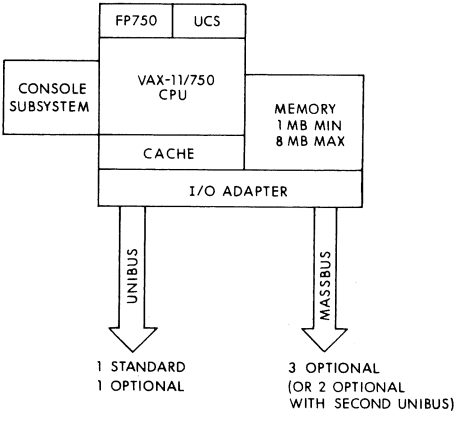


Figure 1-2 VAX-11/750 Hardware Configuration

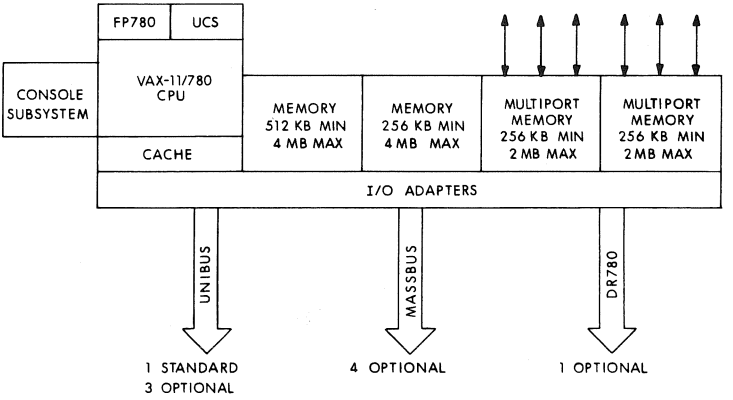


Figure 1-3 VAX-11/780 Hardware Configuration

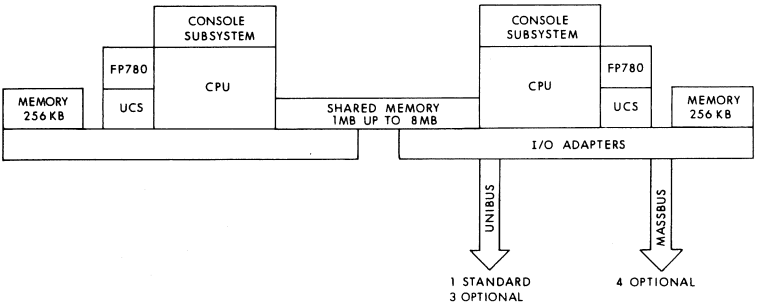


Figure 1-4 VAX-11/782 Hardware Configuration

VAX Central Processing Units (CPUs)

VAX processors are 32-bit high-speed microprogrammed computers that execute instructions in native mode, and nonprivileged PDP-11 instructions in compatibility mode.

The processors can directly address four billion bytes of virtual address space, and provide a complete and powerful instruction set that includes integral decimal, character string, and floating point instructions. Each VAX processor includes memory management hardware, sixteen 32-bit general registers, 32 interrupt priority levels, and a console subsystem which is linked directly to the processor. In addition, the VAX-11/750 and VAX-11/780 each have a memory cache, 4 KB and 8 KB respectively.

An optional high-performance Floating Point Accelerator (FPA) is available for each of the VAX processors. The FPA works in parallel with the basic CPU to execute the standard floating point instruction set. The FPA decreases instruction execution time of floating point arithmetic and some integer arithmetic operations.

Caches — The VAX-11/750 and the VAX-11/780 provide three cache systems: the memory cache, an address translation buffer, and an instruction buffer. The VAX-11/730 does not have a memory cache, but has the two buffers.

The memory cache (typically 90% hit rate on the VAX-11/750 and 95% on the VAX-11/780) provides the central processor with high-speed data access by storing frequently referenced addresses, data and instruction items. The memory cache significantly reduces the effective memory access time for the CPU.

The instruction buffer is an 8-byte buffer that enables the CPU to fetch and decode the next instruction while the current instruction completes execution. The instruction buffer in combination with the parallel data paths (which can perform integer arithmetic, shifting, and memory access at the same time) significantly enhances VAX performance because the CPU seldom has to wait for the next instruction to be fetched.

VAX systems provide an address translation buffer that eliminates extra memory accesses during virtual-to-physical address translations. The address translation buffer contains frequently used virtual-to-physical address translations: 128 page table entry translations in the VAX-11/730 and VAX-11/780, and 512 in the VAX-11/750.

Clocks — The VAX CPUs each have two clocks. The VAX-11/750 and VAX-11/780 have a programmable realtime clock used by system diagnostics and by the VAX/VMS operating system for accounting and

scheduling, and a time-of-year clock, which insures the correct time of day and date. The VAX-11/730 has an interval timer and a time-of-year clock.

Memory Management — The VAX memory management hardware enables the VAX/VMS operating system to provide a flexible and efficient virtual memory programming environment. Hardware memory management, with the operating system, provides both paging (with user control) and swapping.

In addition, memory management provides four hierarchical modes: kernel, executive, supervisor, and user, with read/write access control for each mode.

The memory management hardware facilitates program and data sharing, and allows larger program size and increased performance.

The Console Subsystem

The console subsystem configuration makes the VAX series extremely approachable systems. Four elements combine to give the user access to the system's capabilities: a set of machine status switches and lights on the CPU front panel, a console terminal, a console command language, and an integral mass storage device. Simple console commands, entered through the console terminal, replace the traditional toggle switches and provide operational control (i.e., bootstrapping, initialization, self-testing, examining/depositing data in memory, etc.). When it is not being used to communicate console command language instructions to the CPU, the same terminal functions as a VAX system terminal and is available to authorized users for normal system operations. This dual-purpose console terminal results in significant hardware savings.

The VAX console subsystem and the console command language also facilitate the loading of diagnostics and software updates from the mass storage device: a TU58 tape cartridge on the VAX-11/730 and VAX-11/750, and an RX01 floppy disk system on the VAX-11/780.

With the customer's cooperation, the VAX-11/750 and VAX-11/780 console subsystems may also be equipped with a Remote Diagnosis option which allows connection to a host computer at the DIGITAL Diagnostic Center for fault detection or preventive maintenance procedures. This option can significantly lower maintenance costs as well as contribute to system availability and maintainability. The diagnostic strategy for the VAX-11/730 makes use of a new feature called Customer Runnable Diagnostics (CRD). For more information on CRDs, see Chapter 21 of this handbook. The VAX-11/730 Remote Support option, utilizing Remote Diagnosis technology, provides the DIGITAL

service engineer with a further level of technical resources.

The Main Memory Subsystem

VAX-11/730 Systems — The main memory subsystem consists of a controller and from one to five memory array modules that use 64 Kbit MOS (metal oxide semiconductor) RAM chips for data storage. The array modules are 1 MB each, to give a maximum memory capacity of 5 MB. ECC (Error Correcting Code) allows the correction of all single-bit errors, and the detection of all double-bit errors, to insure data integrity.

VAX-11/750 Systems — The main memory subsystem consists of a controller and from one to eight memory array modules that use 64 K MOS (metal oxide semiconductor) RAM chips for data storage. The array modules are 1 MB each, to give a maximum memory capacity of 8 MB. ECC (Error Correcting Code) allows the correction of all single-bit errors, and the detection of all double-bit errors, to insure data integrity.

VAX-11/780 Systems — The main memory subsystem consists of a controller and from one to sixteen array boards utilizing 16 Kbit MOS RAM chips. Each array board contains 256 KB of physical memory, which can be increased to a maximum array size of 8 MB by means of a second controller. Two memory controllers with equal amounts of memory can be interleaved to improve I/O throughput.

For the VAX-11/782, the local memory is used only for stand-alone diagnostics.

A multiport memory option, the MA780, is also available on VAX-11/780 systems. It can be applied either as a processor interconnect or it can be used to upgrade to a VAX-11/782.

If the MA780 is used as an interconnect, up to two multiport options can be added to a system, each supporting up to 2 MB of shared memory. This increases the VAX-11/780's maximum physical memory from the 8 MB limit for local main memory to 12 MB. Up to four VAX-11/780 systems may share an MA780 memory.

When the MA780 is part of a VAX-11/782 configuration, it is used differently. All main memory resides in the MA780 and is shared by two VAX-11/780 processors. In this instance, up to two MA780s can be used, providing a total of 8 MB of shared main memory.

The Input/Output Subsystems

VAX-11/730 Systems — The UNIBUS subsystem connects most medium and low speed peripheral devices to the VAX-11/730 system. An asynchronous, bidirectional bus, the UNIBUS lets the user select

from a range of existing peripherals (those supported by VAX/VMS and diagnostics) and also provides easy connection for customer-designed special devices.

VAX-11/750 Systems — The VAX-11/750 system configuration includes one UNIBUS adapter as standard equipment. Up to three MASSBUS adapters or one additional UNIBUS and two MASSBUS adapters can be added for connection to mass storage devices and other peripherals.

VAX-11/780 Systems — The VAX-11/780's I/O subsystems also include UNIBUS and MASSBUS adapters. Each system's configuration includes one UNIBUS adapter as standard equipment with the capability to add up to three more as options. Up to four MASSBUS adapters can also be included.

The VAX-11/780 offers as an option a very high performance, 32-bit, general purpose parallel interface which allows user devices to be connected directly to the system's Synchronous Backplane Interconnect. Called the DR780, it is capable of transferring data to and from memory at speeds of up to 6.67 MB per second. (The VAX-11/782 does not support this option.)

READING THIS HANDBOOK

For more information on the specific VAX processors, refer to the following parts of the book:

- Part II—The VAX-11/730
- Part III—The VAX-11/750
- Part IV—The VAX-11/780

The VAX-11/782 is covered in Chapter 19. Part V of the book discusses the reliability, availability, and maintainability features of VAX systems. Part VI includes the Appendices and the Glossary.

PART II
THE VAX-11/730

CHAPTER 2

VAX-11/730 CONSOLE SUBSYSTEM

FEATURES	BENEFITS
Console terminal	Dual function as console and user terminal results in hardware savings
Console command language	Gives the user a powerful, yet easy-to-use, debugging tool
Standard ASCII terminal	Provides a high degree of flexibility
EIA communications interfacing	Allows standard industry-compatible communications
Front panel switches and indicator lights	Offer control and status of certain aspects of the machine operation
Remote access port	Allows use of a second or remote console terminal for automated product test and the Remote Support option
Dual TU58 cartridge tape drives	Provide inexpensive, reliable devices and mediums for: loading CPU microcode; booting; diagnostics; field updates to microcode, diagnostics, and software; and convenient personal data storage on cartridge
Unattended reboot	The system reboots itself upon recovery of electricity after a power failure or other system crash if the Auto Restart switch is in the ON position

INTRODUCTION

The VAX-11/730 console subsystem is designed to allow the user to interactively communicate instructions to the central processing unit using the console terminal and the console command language. Five major elements make up the VAX-11/730 console subsystem:

1. A front panel on the CPU cabinet with switches and indicator lights
2. A separate ASCII terminal, called the console terminal
3. A remote access port
4. A dual TU58 tape cartridge drive in the CPU box
5. A special console command language with simple commands the user types from the console terminal

The integral TU58 tape cartridge drives are used to load the CPU microcode, boot the system, load diagnostics, and update the operating system software. DIGITAL also supports the TU58 under VAX/VMS for data storage and retrieval.

Figure 2-1 illustrates the hardware elements of the VAX-11/730 console subsystem.

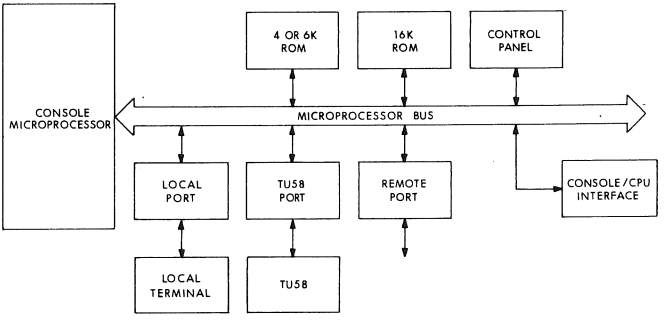


Figure 2-1 VAX-11/730 Console Subsystem

CONSOLE MODES

The VAX-11/730 console runs in two modes: program mode and console mode. These modes are mutually exclusive. One of the modes will always be enabled while there is power to the machine. In program mode (also known as the system terminal mode), the console functions like the other terminals on the VAX-11/730 system. In this mode the console passes characters between the terminal and the program.

When the CPU is not executing instructions, it is halted, and the console terminal is in console mode. In this mode, the CPU is receptive to console commands.

Direct Memory Access (DMA) activity to memory can also occur with the terminal in console mode, and all DMA transactions in progress will continue even when the CPU is halted. This allows the machine to

be halted without destroying these transactions; however, interrupts will not be serviced while the machine is halted. They will be serviced following a CONTINUE from the console.

Table 2-1 lists the actions that cause the CPU to halt and enter console mode. When the keylock switch is in either Disable position (discussed in the Six-Position Keylock Switch section) and a CTRL P is typed, the machine will not halt.

Table 2-1 Console Halt Codes

Code	Meaning
02	CTRL P typed on the console
04	Interrupt stack not valid
05	CPU double error
06	Halt instruction executed
07	Invalid SCB vector
0A	CHMX from the interrupt stack
0B	CHMX to the interrupt stack
0C	SCB physical read error

When the processor enters the console mode it types on the console terminal the address contained in the program counter (PC), a two-digit code which identifies the reason for the halt, and the console prompt symbol, >>>. The prompt symbol shows that the console program is looping, waiting for a command.

VAX-11/730 FRONT PANEL

The front panel of the VAX-11/730 has 2 switches and 4 red indicator lights. Figure 2-2 illustrates the front panel controls and indicators.

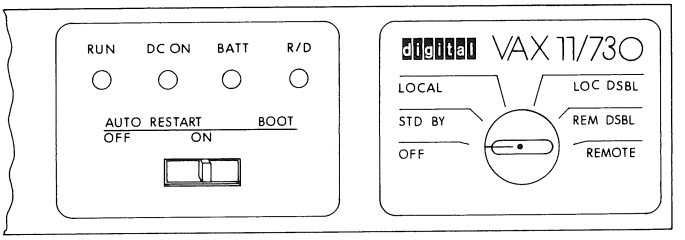


Figure 2-2 VAX-11/730 Front Panel Controls and Indicators

Six-Position Keylock Switch

When the keylock switch is in the **Off** position, no power is applied to the CPU.

Power is applied to the WCS module and Main Memory if the switch is in the **Standby** position.

Table 2-2 describes the remaining four keylock positions.

NOTE

In Table 2-2, "characters" refer to console command language commands and control characters.

Table 2-2

Position	Console Mode	Program Mode
Local	The console accepts and interprets all characters. The Remote Support (RS) line is disabled.	All characters except for CTRL P are passed to the program. The RS line is disabled.
Remote	The local terminal will be disabled upon establishing the RS connection and may be re-enabled by commands from the RS host. The console accepts and interprets all characters from the remote terminal.	All characters except CTRL P passed by the remote terminal (or the local terminal if it has been re-enabled by the remote host) to the program
Local/Disable	All characters are ignored. The remote line is disabled.	All characters are passed to the program.
Remote/Disable	All characters are ignored. The local terminal is disabled and the remote line is enabled.	The remote line may be used as a normal terminal, passing all characters to the program.

Auto Restart Switch

The Auto Restart switch is a three position toggle switch that controls the machine on a power-up sequence. It has the following three positions:

- OFF—The console will halt after loading the microcode. The processor remains halted after a HALT instruction is executed or power is restored.
- ON—The console will load microcode and execute a command file called DEFBOO.CMD. If any halt except a CTRL P halt is executed, or if power is restored following a powerfail, the console will attempt to restart the operating system.
- Boot (momentary)—The console will execute DEFBOO.CMD if the console is in the console mode idle loop (>>>).

State Indicator Lights

The lights on the front panel indicate the following:

- Run—Indicates that the processor is executing instructions
- DC On—Indicates the +5 V is between 4.75 and 5.25 volts, and ±15 V is at least ±13.5 volts
- Remote—Indicates that a remote connection has been made. The light blinks during modem transitions. This light only works if the Remote Support option is used

Diagnostic and Maintenance Aids

The independent console processor of the VAX-11/730 allows testing of other system components by console-based microdiagnostics. Other diagnostic aids related to the console subsystem include:

- Remote Support (RS) option that provides automated checkout capability and an RS or ASCII port
- ROM-resident self-test that can be optionally invoked at power-up and on command
- Voltage monitoring circuits that are incorporated to check +5 V and +15 V
- Monitoring of UNIBUS AC LO
- Customer Runnable Diagnostics

CONSOLE TERMINAL

The console terminal is a powerful tool within the VAX-11/730 console subsystem. By using the console command language, the user can instruct the CPU to perform a wide variety of functions. When the console terminal is in console mode, it is dedicated exclusively to controlling CPU functions such as examining and depositing data in

memory. Console mode also allows the processor to be started, self-tested, initialized to a known state, and single-stepped through instructions. In contrast, program mode dedicates the console terminal to user application processes and the VAX/VMS operating system. In program mode, the CPU will not recognize console commands.

Console Terminal Communications

The electrical interface for the console terminal is an industry-standard full-duplex EIA RS232-C line. The speed of the line is switch-selectable to 300, 1200, 2400, and 9600 baud.

CONSOLE COMMAND LANGUAGE

The VAX-11/73O console command language gives the user an efficient way of communicating with the console subsystem. Instead of using the traditional toggle switches and lights, a user can communicate with the console subsystem by typing simple commands on the console terminal. These instructions to the CPU can only be communicated when the console terminal is in console mode.

Console commands, with the exception of the Directory command, are specified by a single letter with optional modifiers. (The Directory command is specified by DIR.)

Console Command Syntax and Semantics

The syntax and semantics for the console command language are as follows:

Symbol	Function
< >	Angle brackets are used to denote category names. For example, the category name <ADDRESS> may be used to represent any valid address, instead of actually listing all the strings of characters that can represent an address.
[ ]	Brackets surrounding part of an expression indicate that part of the expression is optional.
<SPACE>	Represents one typed space.
<COUNT>	Represents a numeric count in hexadecimal, 32 bits. Leading zeros may be omitted.
<ADDRESS>	Represents an address argument. Valid <ADDRESS> types are explained later in relation to specific commands. Virtual ad-

	dresses that reference nonexistent or non-resident pages will cause the console to abort execution of the console command which referenced that address. An appropriate error message will be displayed. Leading zeros may be omitted.
<DATA>	Represents a numeric argument. Leading zeros may be omitted.
<QUALIFIER>	Represents a command modifier (also called a *switch*). Valid <QUALIFIER> types are explained later in relation to specific commands.
<INPUT-PROMPT>	Represents the console's input prompt string >>>.
<CR>	Carriage return.
<LF>	Line feed.

Prompts

There are three different prompts given by the system:

ROM>	The console subsystem is not able to boot itself and is running in a limited mode. A CTRL-C can be typed to try to reboot the system.
>>>	The console terminal is in console mode
MIC>	The micromonitor is controlling the console subsystem.

Typing Errors and Illegal Characters

Typing errors may be corrected using the DELETE key as long as a <CR> for the line has not been typed. When the DELETE key is typed, the console prints a backslash (\) and the character being deleted. The console also adds a backslash between the last deletion and the next input character.

Example:

Operator types:	1278<del><del>34
Console prints:	1278\87\34
Console sees:	1234

Control and Special Characters

Control characters are typed by holding down the CTRL key and typing the named letter. Unless otherwise specified, the control characters listed are for use when in console mode.

CTRL-C	Aborts current operation. This command instructs the console program to return to its idle loop and reissue the prompt.
CTRL-P (console mode)	Same as CTRL C.
CTRL-P (program mode)	Halts the machine if key switch is in any of the Enable positions.
CTRL-O	Throws away output until pressed again.
CTRL-R	Retypes current contents of input buffer.
CTRL-S	Stops printing. Only control characters are recognized while in this state.
CTRL-Q	Resumes printing.
CTRL-U	Aborts accepting current input line. Returns to idle loop and reissues prompt.
Break	Same action as CTRL-C and CTRL-P in console mode.
ESC	In console mode, causes console to re-execute the last command typed at the console terminal.

Special Notes

1. Qualifier switches (/B, /P, /N:, etc.) may be used anywhere in the command string, except at the beginning. They act as field terminators, just as spaces and carriage returns do.
2. Until the POWER.CMD and CODE0n.CMD files have finished execution, the CPU will not respond properly to user commands that access the CPU. **These command files should not be aborted.**

User commands that require the CPU microcode are:

B	
C	
D and E	/M
	/P
	/V
	/G
	/I
	PS[L], PC, Rn, SP
I	
L/P	
M	
N	
S/P	
W	(Used to detect completion of POWER.CPU)
X/P	

Console Command Language Instructions

BOOT Command

SYNTAX:
B[<SPACE><TU58-SELECT>][<SPACE><DEVICE-NAME>]<CR>

This command causes the Stack Pointer (SP) to be loaded with <the address of the start of a 64 KB block of good memory> +↑X200. (The block of good memory was found by POWER.CPU at power-up time.) It then causes a boot command file to be read from one of the console TU58 drives and executed. Which boot command file is read in and executed is determined by the remaining parameters.

If <TU58-SELECT> is not specified, the console searches for the boot command file on the default TU58. The default TU58 is the TU58 which was used to load microcode when the CPU was powered up.

If <TU58-SELECT> is specified as DD0:, then the console searches on the external TU58.

If <TU58-SELECT> is specified as DD1:, then the console searches on the internal TU58.

If <DEVICE-NAME> is not specified, the file DEFBOO.CMD is read from the indicated TU58 drive and executed.

If <DEVICE-NAME> is specified, it is of the format DQn where n is the unit number of the R80 or RL02 drive which holds the system disk. The file DQnBOO.CMD is read in from the indicated TU58 drive and executed.

CONTINUE Command

SYNTAX: C<CR>

If the CPU clock is running, the CONTINUE command restarts execution of a halted program at the address currently contained in the Program Counter.

If the CPU clock is not running due to a microcode break point while in program mode, the clock is restarted and the console enters program mode. If the clock is off not as the result of a microcode break point, the console turns the clock on and remains in console mode.

RESPONSE (Clock on): <CR><LF>(Console enters Program I/O mode.)

RESPONSE (Clock off): >>>

DEPOSIT and EXAMINE Commands

SYNTAX:

D[<QUALIFIER-
LIST><SPACE><ADDRESS><SPACE><DATA><CR>

E[<QUALIFIER-LIST>][<SPACE><ADDRESS>]<CR>

DEPOSIT and EXAMINE commands will be treated together because their formats are quite similar. Both commands require definition of the address space and the size of the operand in addition to the address.

Table 2-3 DEPOSIT and EXAMINE QUALIFIERS

Data Length Qualifiers	
/B	Byte
/W	Word
/L	Longword
Repetition Qualifiers	
/N:<COUNT>	Executes the EXAMINE or DEPOSIT <COUNT>+1 times. For example, this allows five locations to be examined by specifying /N:4.
Address Space Qualifiers	
/V	Virtual Address
/P	Physical Address

/I	Internal Register
/G	General Register
/M	Machine Dependent Internal Register
/U	Console microcomputer
/C	Writable Control Store (WCS)

Address Specification

nnnnnnnn	Hex Number
PSL or PS, PC,SP	
Rn	n=0 to ↑XF (Hex)
*	Last Location
+	Next Location
-	Previous Location
@	Last data used as address
(DEPOSIT only)	
Data	
nnnnnnnn	Hex Number

DEPOSITs (writes) or EXAMINEs (reads) <DATA> at the <ADDRESS> specified. The address space and size used will depend upon the qualifier or qualifiers specified with the command. If no address space qualifier is used, the default is the last used address space and data length; following another DEPOSIT or EXAMINE, the same space as that of the previous command will be used as the default.

If no size qualifier is typed, the default for a physical or virtual DEPOSIT or EXAMINE is whatever the size was in the previous EXAMINE or DEPOSIT.

<ADDRESS> must be one to eight hexadecimal digits. The initial default is zero; however, the default is unpredictable when the address space is changed. Following another virtual or physical DEPOSIT or EXAMINE, the default is the sum of the address from the last EXAMINE/DEPOSIT plus the size from the last EXAMINE/DEPOSIT. Typing a + or - for <ADDRESS> (for DEPOSIT only) will also get this default. Following another IPR or GPR EXAMINE/DEPOSIT, the default is the sum of the address from the last EXAMINE/DEPOSIT plus one. Using

a PSL for <ADDRESS> performs a longword reference of the Processor Status Longword, independent of the address space and size.

<DATA> must be represented by one to eight hex digits. If more digits than specified by the size are supplied, the extra digits on the left are ignored; if fewer digits are supplied, zeros are appended to the left.

Sample DEPOSIT response: A successful DEPOSIT always receives <INPUT-PROMPT>.

Sample EXAMINE responses (in bold print):

>>>	E/P 1234	Examine physical address 1234
	P 00001234 ABCDEF89	
>>>	E/V 1234	Examine virtual address 1234
	P 00005634 01234567	Note that virtual EXAMINEs display the translated physical address
>>>	E/G 0	Examine general register R0
	G 00000000 98765432	

Index of EXAMINEs and DEPOSITs

E*<CR>	Examine the last location examined or deposited into
E<CR>	Examine the next location
E<SPACE><ADDRESS><CR>	Examine <ADDRESS>; all switches are defaulted to last EXAMINE or DEPOSIT
E/G<SPACE><ADDRESS><CR>	Examine GPR; register number is <ADDRESS>; <ADDRESS> must be a value from 0 to †XF
E/I<SPACE><ADDRESS><CR>	Examine IPR; register number is <ADDRESS>
E/P<SPACE><ADDRESS><CR>	Examine physical <ADDRESS>
E/V<SPACE><ADDRESS><CR>	Examine virtual <ADDRESS>
E<SPACE>PSL<CR>	Examine PSL
E/W/P<SPACE><ADDRESS><CR>	Examine a word at physical <ADDRESS>
E/P/W<SPACE><ADDRESS><CR>	Examine a word at physical <ADDRESS>
E/L/V<SPACE><ADDRESS><CR>	Examine a longword at virtual <ADDRESS>
D*<DATA><CR>	Deposit <DATA> in the last location that was deposited into or examined

D+<DATA><CR>	Deposit <DATA> in the next sequential address
D/G<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in GPR <ADDRESS>
D/I<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in IPR <ADDRESS>
D/P<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in physical <ADDRESS>
D/V<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in virtual <ADDRESS>
D<SPACE>PSL<SPACE><DATA><CR>	Deposit <DATA> in PSL
D<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in <ADDRESS> Switches are defaulted to previous switches
D/V/W<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit a word of <DATA> in virtual <ADDRESS>
D/L/P<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit a longword of <DATA> in physical <ADDRESS>

DIRECTORY Command
SYNTAX: DIR<CR>

Takes a directory of the specified TU58 drive or the default if none is specified.

DIR Takes default

DIR<SPACE>DDn: where n = 0 or 1
0 is the external TU58 and 1 in the internal TU58.

HALT Command
SYNTAX: H<CR>

Causes Halt PC to be printed, but the machine must already be halted to execute this command.

RESPONSE: ?nn<SPACE>PC=xxxxxxxx<CR>
<LF><INPUT-PROMPT>

INITIALIZE Command
SYNTAX: I<CR>

This command performs the following functions:

- Starts the microcode at microaddress 0
- Initializes the TU58 controller
- Initializes the internal machine constants
- Initializes the processor
- Causes a deposit to SP with the address of (the start of 64 Kbytes of good memory) + ↑X200

RESPONSE: <CR><LF><INPUT-PROMPT>

LOAD Command
SYNTAX:
L[<QUALIFIER-LIST>]<SP><FILE-SPECIFICATION><CR>

The Load command is used to read file data from the console's load device to main memory, or to the WCS. If no qualifier is given with the Load Command, physical main memory is loaded.

Table 2-4 LOAD QUALIFIERS

/S:<ADDRESS>

The START qualifier is used to specify a starting address for the load. If no START qualifier is given, the console will start loading at Address 0.

/C

The C qualifier is used to specify that the WCS is to be loaded.

/P

The Physical qualifier is used to force Physical main memory as the destination of the load.

MICROSTEP Command
SYNTAX: M[<SPACE><COUNT>]<CR>

The CPU is allowed to execute the number of microinstructions indicated by <COUNT>. If no <COUNT> is specified, one instruction is performed, and the console enters SPACE-BAR-STEP mode.

The program may be restarted by typing C<CR>, and will continue executing from the current instruction. Typing an N<CR> will cause the program to finish the current instruction and then halt.

NOTE

Stepping through instructions that access the console will not work.

RESPONSE: <CR><LF><TAB> UPC= <CONTENTS OF MICRO PC>

NEXT Command
SYNTAX: N[<SP><COUNT>]<CR>

The CPU is allowed to execute the number of MACRO-instructions indicated by <COUNT>. If no <COUNT> is specified, one instruction is performed, and the console enters SPACE-BAR-STEP mode.

The console enters program mode immediately after issuing the Step, and reenters console mode as soon as the Step completes.

NOTE

Interrupts are blocked while executing the NEXT command from the console.

RESPONSE: <CR><lf>
?02<SPACE><SPACE>PC=<CONTENTS OF PC>

REPEAT Command
SYNTAX: R<SP><CONSOLE COMMAND><CR>

R <CONSOLE COMMAND> causes the console to repeatedly execute the <CONSOLE COMMAND> specified until execution is terminated by a CTRL-C, CTRL-P, or a BREAK. The console commands that may be specified are DEPOSIT, EXAMINE, and INIT.

RESPONSE: <dependent on command specified>

START Command
SYNTAX: S[<SP><ADDRESS>]<CR>
or
S/C<SP><ADDRESS><CR>

The START command is normally used to start execution of programs that run without the operating system and without the diagnostic supervisor. START performs the equivalent of the following console commands:

1. Initialize the CPU.
2. Deposit <address> into the Program Counter (PC). If no address is specified, the current value of the PC is used.
3. Perform the Continue function to begin program CPU execution.

Programs which may be started this way must be loaded into main memory before the START command is given. (See LOAD command.) For example:

>>>S 1000	Start the program that begins at address 1000.

TEST Command
SYNTAX: T<CR>

This command attempts to find the program ENKAA.EXE on Drive 0 or 1. If found, it is loaded and control is transferred to it. If it is not found,

the system reloads itself and returns to console mode. The T command, whether it finds ENKAA.EXE or not, destroys the state of the machine.

Once the MIC> prompt is reached, microdiagnostics can be performed.

BINARY LOAD/UNLOAD Command

SYNTAX:

X<SP><ADDRESS><SP><COUNT><CR><CHECKSUM>

<ADDRESS>	Starting address of load
<COUNT>	Number of bytes to be transferred
<CHECKSUM>	2's complement checksum of command string or binary data

The BINARY LOAD/UNLOAD command instructs the console to load binary data into or unload binary data from physical memory, starting from the location specified by <ADDRESS>. A <COUNT> with bit <31> set indicates BINARY UNLOAD. If bit <31> is clear, it is a BINARY LOAD. The remaining bits in <COUNT> indicate the number of bytes to Load or Unload. After a correct 2's complement checksum calculation, the console issues an <INPUT-PROMPT>, but remains in binary mode and either sends data to the user or prepares to receive data. If the checksum shows an error, a message and an <INPUT-PROMPT> are issued.

In UNLOAD, a binary string of data, of length <COUNT> + 1, will be sent once the <INPUT-PROMPT> indicates that the console has accepted the command. When <COUNT> is exhausted, the final byte is a console-calculated block checksum of all the data.

For LOAD, the console processes the command and <CHECKSUM>. Then, if the checksum is correct, the console responds with <INPUT-PROMPT>, followed by a string of bytes which is the binary data requested. A second checksum is calculated and processed as for the LOAD sequence.

Console Command Errors

When a command is given that the console cannot properly process, it responds by typing ?nn. nn is an error code that describes the nature of the problem.

INTEGRAL TU58 CARTRIDGE TAPE DRIVES

The dual TU58 tape cartridge drives are an important part of the console subsystem. The fundamental function of the TU58 is to load the CPU microcode. If one of the TU58s fails, the second TU58 may be used as backup. Additionally, because the TU58 is connected directly to the console subsystem, it retains the ability to run micro and macro level diagnostics even with some system components inoperative. This feature significantly increases system maintainability. The TU58 can be used to boot the system, to load files into physical memory, and to store files which describe and execute site-specific bootstrap procedures (see BOOTING THE VAX-11/730 SYSTEM later in this chapter).

The tape cartridge is preformatted and contains 256 KB, normally formatted in 512-byte records. The controller provides random access to any record. The TU58 searches at 60 inches per second (i/s) to find the file requested, then reads at 30 i/s. Data read from the tape are verified through checksums at the end of each record or header.

From the system operator's standpoint, the booting process is quite simple. For a typical system boot:

1. Turn on power to the console terminal.
2. Set the Auto Restart switch to the On position.
3. Set the Six-Position Keylock Switch to the Local or Local Disable position. The terminal will then print:

   ```
   CONSOLE
   VERSION XX.XX
   ```

At power-up, the console subsystem bootstraps itself from the TU58, finds 64 KB of good memory, determines what CPU options are installed, and loads the appropriate CPU microcode. It then samples the Auto Restart switch. If it is in the OFF position, the console awaits further commands. If it is in the ON position, a bootstrap program is loaded and begins to execute.

Console Subsystem Action on Boot

There are five distinct ways a boot sequence may be initiated on the VAX-11/730:

1. A power-up sequence (Boot Switch or initial power application) with switch set to ON.
2. By typing the B command while in console mode.
3. Execution of a HALT instruction when the processor is in kernel mode and the Auto Restart switch is in the ON position.
4. Execution of an MTPR to the console register that invokes a boot.
5. Failure of a restart while the Auto Restart switch is in the ON position.

All five of the above mechanisms will initiate the following sequence of actions. Numbers 1 and 2 clear the bootstrap flag. Numbers 3, 4, and 5 will not.

- Clearing of the bootstrap flag.
- Checking of the bootstrap flag. If it is set, in most instances the machine halts. However, there are certain cases where it will attempt to reload all the control stores by starting the console program at its power-on startup point. If it is clear, then it sets the flag. This prevents bootstrap looping.
- Initialization of the CPU (This places the first address +200 of a 64 KB good block of memory into the SP.)
- Execution of the command file DEFBOO.CMD which loads the operating system from the specified mass storage device.

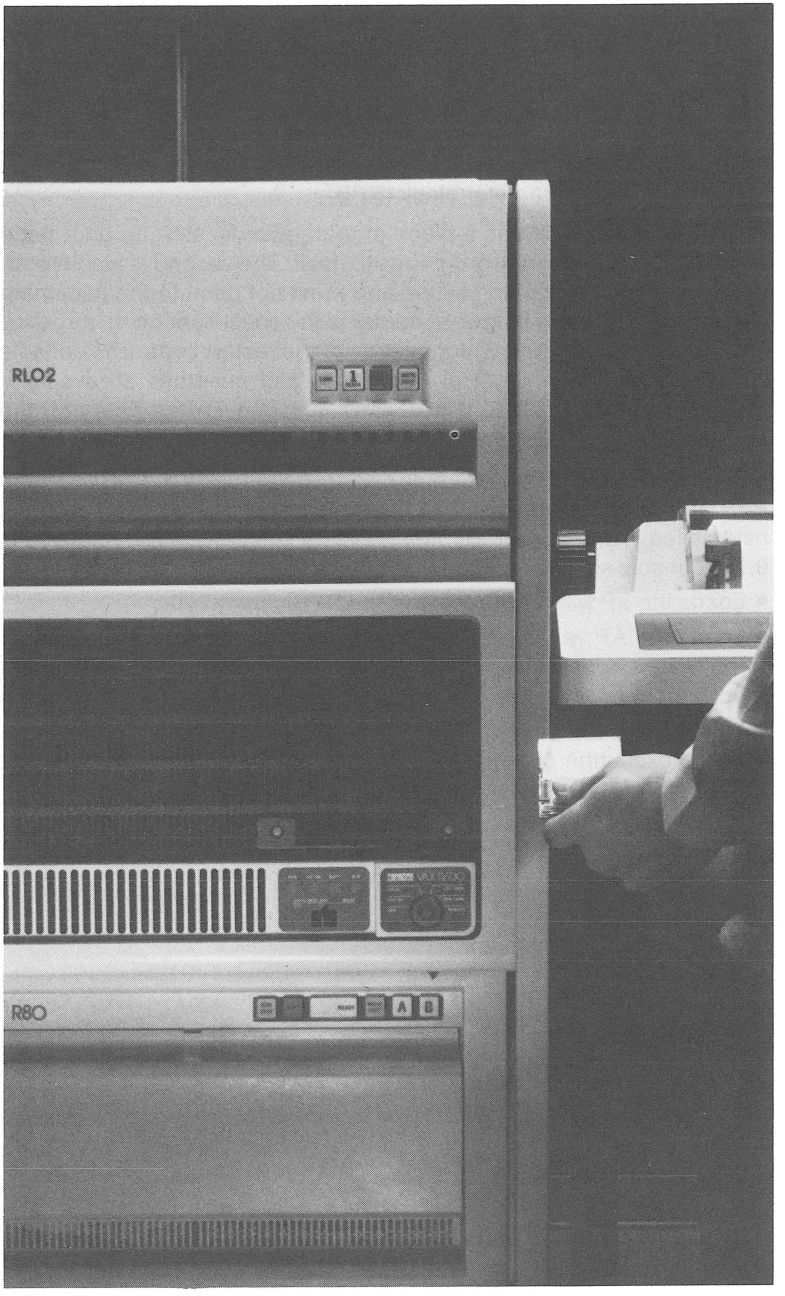
Console Subsystem Action on a Restart

When the console subsystem gains control of the VAX-11/730 following a power restoration or CPU Halt, it examines the Auto Restart switch. If the switch is in the OFF position, the console will remain in console mode with the processor halted. If the Auto Restart switch is set to ON, the console subsystem searches through physical memory for a valid restart parameter block (RPB).

A valid RPB is defined as a block of 4 longwords, starting on a page boundary. The first longword points to itself. The second is a pointer to the address of the restart routine and must not point to the beginning of the RPB. The third longword contains the checksum (sum, throwing away carries) of the first 31 longwords of the restart code. The console subsystem starts at physical address 0 and searches all available memory for a valid RPB. If it doesn't find one, restart fails and the console attempts a system boot.

If it does find an RPB, it examines bit 0 of the fourth longword of the RPB. If this bit is 1, a restart has already been attempted and must have failed. The console will then execute DEFBOO.CMD. If this bit is 0, the console sets it and then:

- Loads the SP with the address of the RPB plus ↑X200.
- Loads the AP with a value that indicates the cause of the restart. If the restart is occuring because of power restoration or reset switch activation, the AP gets a 3. If it is because of a CPU halt, it gets the value specified in Table 2-1.
- Starts execution of the restart routine, whose address is located at the second longword of the RPB.


RLO2
R8O
A
B

CHAPTER 3

VAX-11/730 CENTRAL PROCESSOR

FEATURES	BENEFITS
32-bit microprogrammed processor	Provides the full VAX/VMS architectural implementation
Soft control store	Allows loading of microcode updates and microdiagnostics from TU58
Programmed Array Logic (PAL) technology	Increases logic density and reduces cost
PDP-11 compatibility mode	Gives the PDP-11 user an easy migration path to the VAX/VMS architecture
Translation buffer	Minimizes memory accesses for virtual to physical address conversion
1-Longword Prefetch Instruction Buffer	Allows the next instruction to be fetched while the current instruction is executing
Integrated Disk Controller	Allows up to one R80 and three RL02 disk drives to interface a VAX-11/730 system
Single-system, 40-inch cabinet packaging	Reduces space and cabling requirements
19 inch wide CPU box	Allows for packaging in industry standard cabinets
Optional Floating Point Accelerator	Decreases instruction execution time of floating point arithmetic and some integer arithmetic operations

INTRODUCTION

The VAX-11/730 central processing unit (CPU) is a 32-bit microprogrammed computer that executes the VAX instruction set in native mode and supports the VAX/VMS operating system. Non-privileged PDP-11 instructions can be executed in compatibility mode, allowing existing user mode PDP-11 programs to run without modification.

Three standard HEX modules make up the CPU: the DAP (data path) module, the WCS (writable control store) module, and the MCT (memory controller) module. Additionally, the integrated disk controller and the optional floating point accelerator each consist of a standard HEX module.

The CPU performs the logic and arithmetic operations requested by the computer system users. It uses 32-bit virtual addresses, allowing access to 4.3 gigabytes of virtual address space. These addresses are called virtual because each address is not necessarily the actual address in physical memory. The processor's memory management hardware translates virtual addresses to physical addresses.

The processor provides sixteen 32-bit registers that can be used for temporary storage, as accumulators, index registers, and base registers. Four of these registers have special significance: the Program Counter (PC), the Stack Pointer (SP), and two registers used in the extensive CALL facility.

The native instruction set is highly versatile and bit-efficient. It includes integer, packed decimal, character string, bit field, and floating point instructions, as well as program control and special instructions. Instructions and data are variable in length and can start on any byte boundary or, in the case of bit field data, at any arbitrary bit in memory.

The VAX-11/730 CPU can process the following kinds of data:

- Bits (up to 32)
- Bytes (8 bits)
- Words (16 bits)
- Longwords (32 bits)
- Quadwords (64 bits)
- 32-bit floating point (single precision)
- 64-bit floating point (double precision)
- 64-bit floating point (double precision extended range)
- 128-bit floating point (quadruple precision extended range)
- Packed decimal (up to 31 digits)
- Character strings (up to 64 KB)
- Queues

The remainder of this chapter is divided into two sections. The first section discusses the Programmed Array Logic (PAL) technology used in the VAX-11/730, and the second section describes the hardware elements listed on the next page.

- CPU control store
- Internal data paths
- Address translation buffer/UNIBUS Map
- Prefetch instruction buffer
- Interval timer and time-of-year-clock
- Integrated disk controller (RB730)
- Optional floating point accelerator (FP730)

Figure 3-1 illustrates the central processing unit.

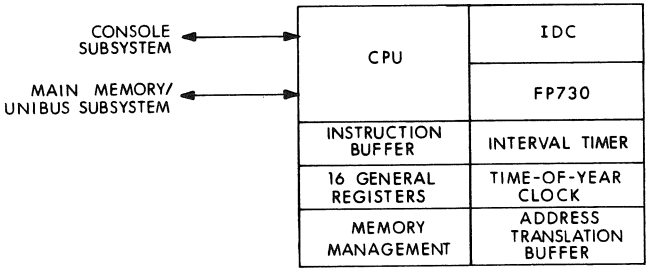


Figure 3-1 The Central Processing Unit

PROGRAMMED ARRAY LOGIC TECHNOLOGY

Programmed array logic (PAL) devices are logic arrays manufactured on a chip using the TTL Schottky bipolar process and fusable-link technology. The high logic density of PALs decreases the required amount of space and reduces cost.

The basic circuitry used in PALs consists of programmable AND arrays connected to fixed OR arrays. In the PAL circuits used in the VAX-11/730, up to 32 programmable AND inputs and up to 8 fixed OR inputs are used per output.

An unprogrammed PAL—one with all fuses intact—is programmed by first determining the AND inputs to be used and then "blowing" the links for the unused AND inputs. This produces the desired AND before OR logic configuration.

HARDWARE ELEMENTS

CPU Control Store

The CPU microcontroller consists of a microsequencer and a control store. The control store is a programmable Read/Write memory with a basic storage capacity of 16 K 24-bit microwords. An additional 1 K microwords of control store is available to support the integrated disk controller.

The control store sequences the CPU to implement the native and PDP-11 compatibility mode instruction sets. The CPU microcode is loaded into the control store from the TU58 tape drive during system bootstrap.

Each microinstruction is 24 bits and contains several control fields for specific CPU functions. The sequence of microinstructions read from the control store and loaded into the control store register (CSR) is determined by the microsequencer.

Internal Data Paths

The CPU data path performs the arithmetic and logical operations necessary to execute the instruction set. The principal data path components are 4-bit processor slices. Eight of these slices are connected in parallel to give an arithmetic and logical processing element 32-bits wide. The data path also contains a 256 location X 32-bit local store RAM that includes, among other things, the general registers and several of the architecturally-defined privileged processor registers. The data path is controlled by microcode executing in the CPU's mirocontroller.

Address Translation Buffer

The address translation buffer contains frequently used virtual address translations. It significantly reduces the amount of time spent by the CPU on the repetitive task of dynamic address translation. The buffer contains 128 virtual-to-physical page address translations: 64 system space translations and 64 process space translations. Each of these sections has parity on each entry for increased integrity.

1-Longword Prefetch Instruction Buffer

The 1-longword prefetch instruction buffer allows the next instruction to be fetched while the current instruction is executing. The control logic continuously fetches data from memory to keep the longword buffer full. In native mode, the variable length instructions are stored in contiguous byte positions in memory and are aligned on byte boundaries. In compatibility mode, the PDP-11 instructions are 16 bits, occupy two contiguous bytes, and are aligned on word boundaries.

Interval Timer and Time-of-Year Clock

The VAX-11/730 processor contains an interval timer and a time-of-year clock. The interval timer permits the measurement of finely resolved intervals. The time-of-year clock is used by software to perform various timekeeping functions.

Integrated Disk Controller (RB730)

The RB730, an integrated disk controller (IDC), interfaces an R80 disk drive and from one to three RL02 disk drives to the system. (Up to four RL02 drives, with no R80 drive, may also be interfaced.) Data transfer between the IDC and the CPU is over the accelerator bus and is controlled in part by dedicated microcode in the CPU. One 32-bit longword of disk Read/Write data is transferred at a time, following the generation of a microlevel (fast) processor interrupt request by the IDC. Data silos (FIFOs) in the IDC provide up to 1 KB of data buffering for both read and write data.

In addition to connecting to the accelerator bus, the IDC connects to the UNIBUS for generating interrupts other than the fast interrupts generated for disk data transfers. The IDC's UNIBUS connection also monitors the system's powerfail state.

Optional Floating Point Accelerator (FP730)

The FP730 floating point accelerator is a hardware option that operates in conjunction with the VAX-11/730 to execute the standard floating point instruction set. Floating point representation permits a greater range of number values than is possible with a 32-bit integer.

Consisting of a single module, the FP730 is easily installed and is functionally transparent to the user. Hardware and software modifications are not required, however, the CPU microcode must be reloaded from the TU58.

The FP730 receives an opcode from the CPU and decodes the information into a starting microaddress. The outputs of the control store ROM control the arithmetic operations and data path logic.

The FP730 executes addition, subtraction, multiplication, and division instructions which operate on single precision (32-bit), double precision (64-bit), extended range double precision (64-bits), and extended range quadrupled precision (128-bits) operands. It executes Extended Multiply (EMOD) and Polynomial Evaluation (POLY) instructions, and converts data between integer and floating point formats and between single and double precision floating point formats. The FP730 also executes 32-bit integer multiplication and division instructions.

The floating point instructions performed by the FP730 are listed in Table 3-1.

Table 3-1 FPA Floating, Double, and Integer Instructions

OPCODE (HEX)	INSTRUCTION MNEMONIC	OPCODE (HEX)	INSTRUCTION MNEMONIC
40	ADDF2	4BFD	CVTRGL
41	ADDF3	6AFD	CVTHL
60	ADDD2	6BFD	CVTRHL
61	ADDD3	56	CVTFD
40FD	ADDG2	99FD	CVTFG
41FD	ADDG3	98FD	CVTFH
60FD	ADDH2	76	CVTDF
61FD	ADDH3	32FD	CVTDH
42	SUBF2	33FD	CVTGF
43	SUBF3	56FD	CVTGH
62	SUBD2	F6FD	CVTHF
63	SUBD3	F7FD	CVTHD
42FD	SUBG2	76FD	CVTHG
43FD	SUBG3	4C	CVTBF
62FD	SUBH2	6C	CVTBD
63FD	SUBH3	4CFD	CVTBG
44	MULF2	6CFD	CVTBH
45	MULF3	4D	CVTWF
64	MULD2	6D	CVTWD
65	MULD3	4DFD	CVTWG
44FD	MULG2	6DFD	CVTWH
45FD	MULG3	4E	CVTLF
64FD	MULH2	6E	CVTLD
65FD	MULH3	4EFD	CVTLG
46	DIVF2	6EFD	CVTLH
47	DIVF3	51	CMPF

66	DIVD2	51FD	CMPG
67	DIVD3	71FD	CMPH
46FD	DIVG2	54	EMODF
47FD	DIVG3	54FD	EMODG
66FD	DIVH2	74FD	EMODH
67FD	DIVH3	55	POLYF
48	CVTFB	55FD	POLYG
68	CVTDB	75FD	POLYH
48FD	CVTGB	71	CMPD
68FD	CVTHB	74	EMODD
49	CVTFW	75	POLYD
69	CVTDW	C4	MULL2
49FD	CVTGW	C5	MULL3
69FD	CVTHW	C6	DIVL2
4A	CVTFL	C7	DIVL3
4B	CVTRFL	4F	ACBF
6A	CVTDL	6F	ACBD
6B	CVTRDL	4FFD	ACBG
4AFD	CVTGL	6FFD	ACBH

CHAPTER 4

VAX-11/730 MAIN MEMORY SUBSYSTEM

FEATURES	BENEFITS
Expandable memory configuration	Allows the addition of 1 MB array modules up to a maximum of 5 MB.
Error correcting memory controller	Enhances data availability and reliability by correcting all single bit errors and detecting all double bit errors within the memory system.
64 Kbit memory chips	1 Mbyte per array module for high density.

INTRODUCTION

The main memory in the VAX-11/730 consists of from one to five memory array modules that use 64 K MOS (metal oxide semiconductor) RAM chips for data storage. The modules are connected to the CPU by the array bus. Up to five 1 MB array modules may be installed to give a maximum memory capacity of 5 MB. The minimum memory configuration is 1 MB.

Memory data transfers over the array bus are 39 bits: one 32-bit longword (four bytes) of data and seven associated ECC (Error Correction Code) bits. The ECC bits provide for the detection and correction of all single-bit errors when a longword is read from the memory array. Double-bit errors are detected but not corrected. All errors are reported back to the CPU where they may be recorded for use in preventive and corrective maintenance operations. Figure 4-1 illustrates the basic memory subsystem.

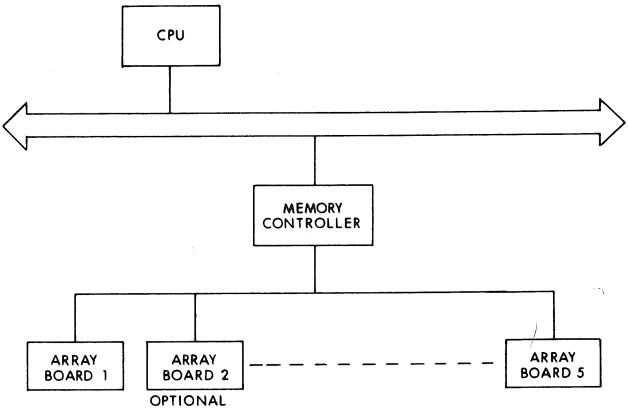


Figure 4-1 Main Memory Configuration

The MCT (memory controller) module contains the memory and UNIBUS control logic. The memory control logic controls data transfers to and from the main memory array modules over the array bus. The UNIBUS control logic controls transfers to and from the peripheral devices over the UNIBUS. Transfers are initiated by the CPU data path under the control of the CPU microcode, or by the UNIBUS devices when direct data transfers are made between the UNIBUS devices and main memory.

The memory and UNIBUS control logic contain the following:

- A translation buffer that converts virtual addresses from the CPU to physical addresses. The physical addresses are then applied to the memory array modules.
- A UNIBUS map which converts 18-bit addresses to physical memory addresses.
- A UNIBUS arbitrator that regulates activity on the UNIBUS and assigns the memory controller to the CPU if the controller is not being used by a UNIBUS device.
- A data rotating and substituting network for aligning memory data. Data retrieved from the arrays is always a 32-bit longword, but the requested byte is not necessarily in the desired position. The data rotator rearranges the data before it is sent to the CPU or the UNIBUS device.
- A control store that outputs 72-bit microwords that control operations within the memory controller.

- A microsequencer that selects the next 72-bit microword.
- ECC gate arrays.

64 K RAM Chips

The memory array modules of the VAX-11/730 use 64 K MOS (metal oxide semiconductor) RAM chips for data storage. 156 RAM chips are arranged in four banks of 39 chips. Each chip has a 256 X 256 matrix, providing 64 K one-bit locations and thus, 64 K 39-bit data locations per bank. The four banks yield 1 Mbyte with associated check bits.

BASIC MEMORY OPERATIONS

The physical address space contained in the memory controller is divided into two areas: physical memory addresses and I/O addresses. A physical address is 24 bits, allowing a total address space of 16 MB. Figure 4-2 illustrates the combined areas.

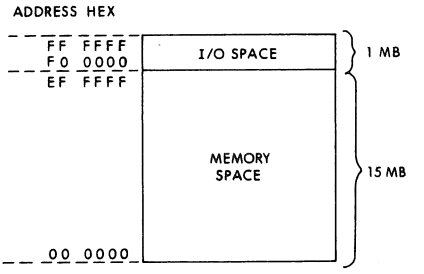


Figure 4-2 Physical Address Space

To access this address space, the system has two types of operations.

Read Operation

In a Read operation, the physical address on the array bus selects the desired module and addresses the desired location on the module. The addressed longword and its seven associated check bits are retrieved and placed onto the array bus.

The ECC logic takes the longword and check bits from the array bus and checks the longword for data errors. If a data error is found, ERROR is asserted to the error logic which then asserts ERR SUM (for a CPU operation) or UB ERR SUM (for a UNIBUS operation). The error is corrected in place and the CRD (correctable read data) bit is asserted in the CSR. If the error is uncorrectable, the cycle is aborted and the RDS (read data substitute) bit is asserted in CSR1. The ECC logic

returns the longword to the array bus for the data rotator. The data is aligned, if necessary, and output onto the MC (memory control) bus.

If the operation is a CPU READ, the longword is sent to the CPU. If the operation is a UNIBUS device Read, the byte or word is transferred to the data lines of the UNIBUS. If the operation is a CPU Read of a UNIBUS device, the Read data is taken off the UNIBUS data lines, transferred to the MC bus via the WCS (Writable Control Store) transceivers, then to the data rotator, and then to the CPU.

Write Operation

In a CPU Write operation, Write data is placed on the MC bus from the CPU and then applied to the data rotator. If the CPU Write is to a UNIBUS device, the rotator outputs the data back to the MC bus. From the MC bus, the data is placed on the data lines of the UNIBUS. If the CPU Write is to the memory arrays, the rotator outputs the data onto the data lines of the array bus. For a UNIBUS to memory Write operation, Write data is placed on the MC bus, applied to the data rotator, and then placed onto the data lines of the array bus.

The ECC logic takes the Write data off the array bus and uses it to generate seven ECC check bits. The data is returned to the array bus along with the check bits. The Write data and the associated check bits are taken from the array bus and written into the selected array module.

CONTROL AND STATUS REGISTERS

Three CSRs (control/status registers) are used in the main memory subsystem to input control signals into the memory controller and to report status to the CPU: CSR0, CSR1, and CSR2.

CSR0 is a Read Only register containing ECC check bits. The CPU reads the check bits to analyze data errors.

CSR1 is a Read/Write register. The CPU writes control bits into CSR1 for maintenance purposes and to regulate operation of the memory controller. Errors relating to a CPU/memory transfer are sensed by the error logic and set error bits in CSR1. When the logic asserts ERR SUM, the CPU reads the CSR1 error bits.

CSR2 is a Read Only register. Errors relating to a UNIBUS/memory transfer are sensed by the error logic and set error bits in CSR2. When the logic asserts UB ERR SUM, the microsequencer causes a timeout on the UNIBUS resulting in the CPU reading the CSR2 error bits.

CSR0 Bit Allocations

Figure 4-3 shows the organization of CSR0. The default values shown are those following a power-up sequence.

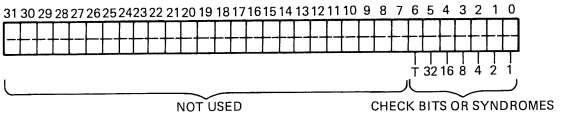


Figure 4-3 CSR 0

Bit: 31:7 **Name:** Not used.

Bit: 6:0 **Name:** Error Syndrome
Function: These bits are syndromes if a correctable error occurred during a CPU to memory transaction, or check bits if a noncorrectable error occurred. These bits are read-only.

CSR1 Bit Allocations
Figure 4-4 shows the organization of CSR1. The default values shown are those following a power-up sequence.

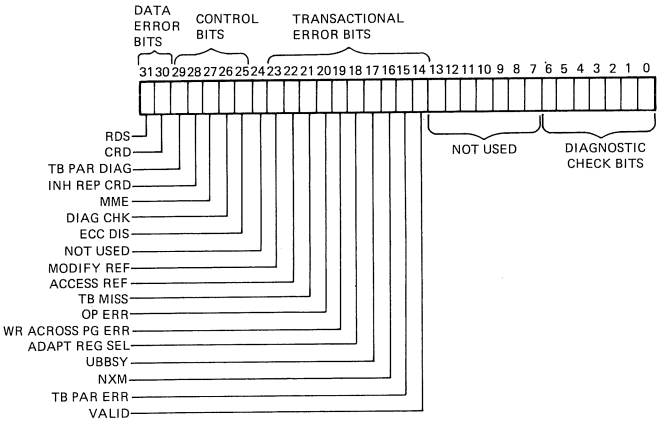


Figure 4-4 CSR1

Bit: 31 **Name:** RDS bit
Function: The RDS (read data substitute) bit is set if on any CPU read to memory an uncorrectable data error is encountered.

Bit: 30 **Name:** CRD bit
Function: This bit is set if on any CPU read to memory a single bit data error is detected and corrected.

Bit: 29 **Name:** TB PAR DIAG
Function: This bit is the translation buffer parity diagnostic bit. When this bit is set it will cause TB PAR ERR on any access to a TB entry in the translation buffer.

Bit: 28 **Name:** INH REP CRD
Function: This bit inhibits the reporting of the occurrence of CRD errors. When this bit is set, all CRDs will still be corrected by the ECC logic and syndromes will be logged in bits <6:0> of CSR0, however the CRD bit will not be set.

Bit: 27 **Name:** MME bit
Function: The Memory Management Enable bit enables translations via the translation buffer. If this bit is not set, the CPU address will map directly into physical memory. The UNIBUS map is always enabled.

Bit: 26 **Name:** DIAG CHK
Function: When this bit is set, the memory controller is in the diagnostic mode.

Bit: 25 **Name:** ECC DIS
Function: This bit disables the ECC correction logic.

Bit: 24 **Name:** Not used

Bit: 23 **Name:** MODIFY REF
Function: This bit means that the modify bit was not set in a TB entry referenced with a WCHK protection request.

Bit: 22 **Name:** ACCESS REF
Function: This bit signifies that the protection PROM has decided that the access requested by the CPU is not allowed. If the PAR ERR bit is set, this bit is meaningless. This bit will assert only if memory management is enabled.

Bit: 21 **Name:** TB MISS
Function: This bit indicates that the translation buffer tag didn't match bits <30:15> of the virtual address in the VAR, or the BYTE OFFSET bit was not set.

Bit: 20 **Name:** ILL UB OPER
Function: This bit is set by an operation error (OP ERR). OP ERR indicates that the CPU tried to perform a memory operation that was illegal in compatibility mode, or tried to perform an illegal UNIBUS reference.

Bit: 19 **Name:** WR ACROSS PG ERR
Function: This bit is set if a CPU write (with a WCHK) attempts to write across a page boundary.

Bit: 18 **Name:** ADAPT REG SEL
Function: This bit indicates that the physical address of a memory reference lies in the 768 Kbyte UNIBUS adapter space.

Bit: 17 **Name:** UBBSY
Function: This bit indicates that a CPU transaction has the UNIBUS as a target but the UNIBUS is busy.

Bit: 16 **Name:** NXM
Function: This bit indicates that the memory select decoder referenced a memory address where there is no memory array board.

Bit: 15 **Name:** TB PAR ERR
Function: This bit is set if a CPU access to the translation buffer results in a parity error.

Bit: 14 **Name:** VALID
Function: This bit is set if a CPU access to the translation buffer finds the TB entry VALID bit not set.

Bit: 13:7 **Name:** Not used.

Bit: 6:0 **Name:** Diagnostic Check bits
Function: These seven bits are used to check the ECC logic.

CSR2 Bit Allocations
Figure 4-5 shows the power-up state of CSR2. This register is read-only.

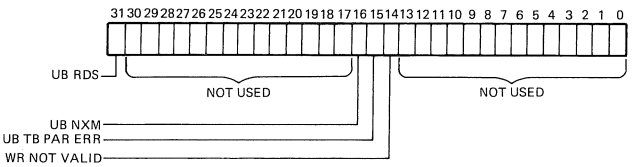


Figure 4-5 CSR2

Bit: 31 **Name:** UB RDS
Function: This bit is set if on any UNIBUS read to memory an uncorrectable data error is encountered.

Bit: 30:17 **Name:** Not used.

Bit: 16 **Name:** UB NXM
Function: This bit is set if a UNIBUS request referenced a nonexistent array card.

Bit: 15 **Name:** UB TB PAR ERR
Function: This bit is set if a UNIBUS/memory transaction results in a TB parity error.

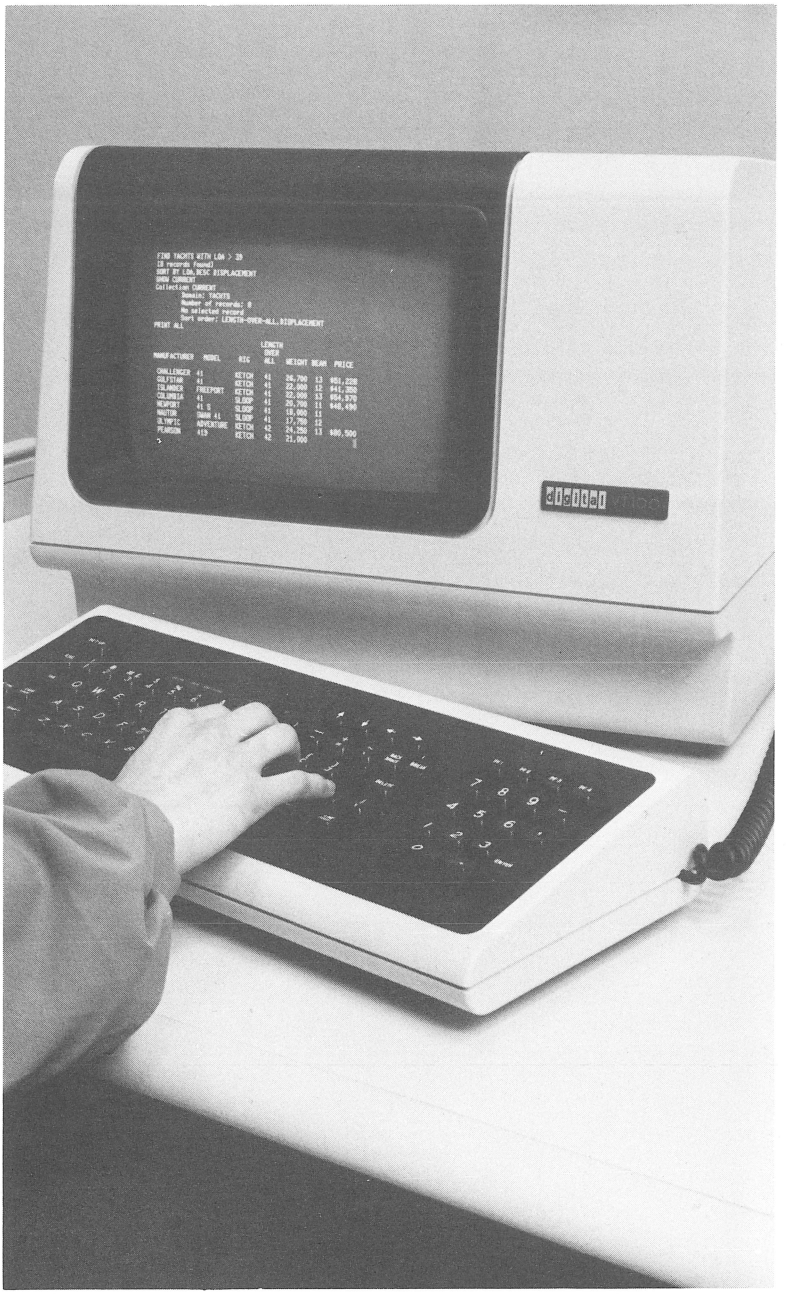
Bit: 14 **Name:** WR NOT VALID
Function: This bit is set if, during a two-cycle operation, an attempt is made to write into a page that does not have a valid entry in the translation buffer.

Bit: 13:0 **Name:** Not used.

ERROR CHECKING AND CORRECTION

The ECC scheme used in the memory subsystem is capable of detecting single and double bit errors. If a single bit error is detected, the ECC logic can be used to correct the error. The logic can detect, but not correct, multibit errors.

Each time a longword (32 bits) is written into memory, 7 check bits are generated from the longword data and are stored with the longword. When the data is retrieved from the arrays, seven syndrome bits are generated from the 39 retrieved bits. These seven syndrome bits indicate no error, a single error, or multiple errors. The single bit error is corrected using the seven syndrome bits and is reported as a CRD (correctable read data) error.


FIND YACHTS WITH LOA > 39
18 records found
SORT BY LOA,MISC DISPLACEMENT
SHOW CURRENT
Collection CURRENT
Domain: YACHTS
Number of records: 0
No selected records
Sort order: LENGTH-OVER-ALL,DISPLACEMENT
PRINT ALL
MANUFACTURER MODEL RIG LENGTH OVER ALL WEIGHT BEAM PRICE
CHALLENGER 41 KETCH 41 26,700 13 $91,220
GULFSTAR 41 KETCH 41 27,000 12 $41,365
ISLANDER FREEPORT KETCH 41 32,000 13 $54,970
COLUMBIA 41 SLOOP 41 32,500 13 $54,970
NEWPORT 41 S SLOOP 41 28,700 11 $46,450
NAUTOR SWAN 41 SLOOP 41 18,000 11
O'DAY JC ADVENTURE KETCH 42 17,700 12
PEARSON 419 KETCH 42 24,250 13 $80,500
21,000

CHAPTER 5

VAX-11/730 UNIBUS SUBSYSTEM

FEATURES	BENEFITS
Direct memory access (DMA) data transfers	Eliminates processor intervention for high data throughput
Communication among UNIBUS devices	Allows direct data transfer between UNIBUS devices without CPU involvement
Compatible with the PDP-11 UNIBUS	Supports a wide range of standard DIGITAL peripherals
Direct vectored hardware interrupts	The CPU avoids time-consuming polling tasks when servicing interrupt requests
UNIBUS device registers are addressed as memory locations	Simplifies I/O programming

INTRODUCTION

The UNIBUS subsystem connects most medium and low speed peripheral devices to the VAX-11/730 system. An asynchronous, bidirectional bus, the UNIBUS lets the user select from a range of existing peripherals (those supported by VAX/VMS and diagnostics) and also provides easy connection for customer-designed special devices. Although the UNIBUS was originally designed for implementations of the PDP-11 architecture, its capabilities have been expanded by the VAX architecture. Thus, existing UNIBUS peripheral devices can be used with the new VAX family architecture without hardware modification. (UNIBUS addresses in this chapter are listed in both octal and hexadecimal. The PDP-11 uses octal while the VAX family uses hexadecimal.)

The integral UNIBUS adapter (part of the CPU) connects the UNIBUS to the system and performs priority arbitration among UNIBUS devices. Furthermore, the UNIBUS adapter lets the processor access UNIBUS peripheral device registers. Figure 5-1 illustrates the UNIBUS subsystem configuration.

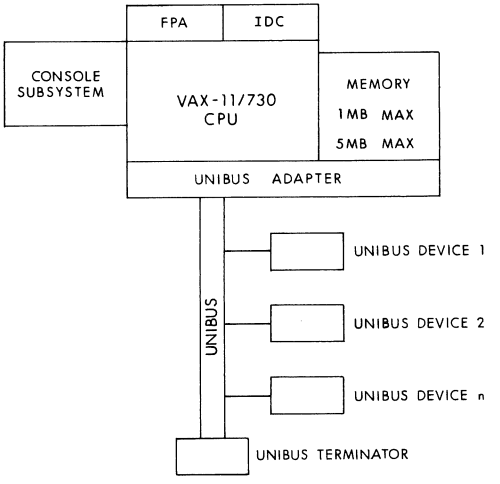


Figure 5-1 VAX-11/730 UNIBUS Configuration

VAX-11/730 UNIBUS SUMMARY

The UNIBUS is a communication path that links I/O devices to the UNIBUS adapter. Device addresses, data, and control information are passed along the 56 signal lines of the UNIBUS. The UNIBUS adapter handles all communication between the UNIBUS and the system, and fields device-generated interrupts.

Conceptually, the UNIBUS is designed around memory elements with ascending addresses starting at UNIBUS address zero, while registers storing I/O data or individual peripheral device status information have addresses in the highest 8 KB the 256 KB UNIBUS address space ($3E000_{16}$ to $3FFFE_{16}$, or 760000_8 to 777776_8).

Figure 5-2 illustrates the signal line configuration. 51 of these lines are parallel and 5 are serial; 42 lines are bidirectional and 14 are unidirectional.

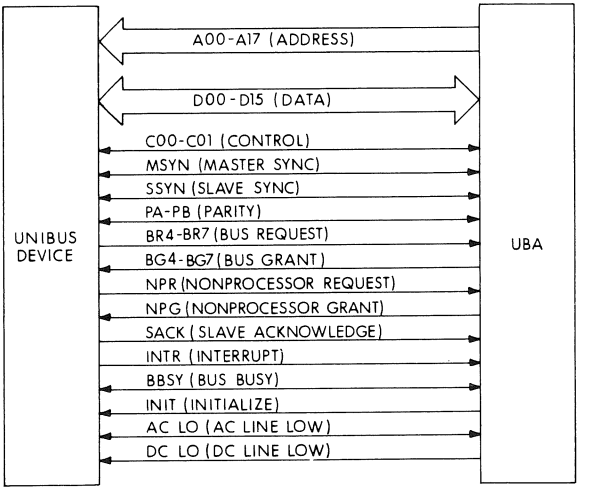


Figure 5-2 UNIBUS

Communication between any two devices on the bus is in a master/slave relationship. During any bus operation, one device, the bus master, controls the bus when communicating with another device on the bus, called the slave. For example, the processor, as master, can fetch from a peripheral device; or the disk, as master, can transfer data to memory. Master/slave relationships are dynamic; the processor, for example, may pass bus control to a disk, then the disk may become master and communicate with slave memory. On the VAX-11/730, the main memory that the processor deals with for instructions and data is not on the UNIBUS, but is attached to the processor. The UNIBUS adapter is the section of the processor that causes the VAX-11/730 memory to look like a slave to UNIBUS devices.

When two or more devices try to obtain control of the bus at once, priority circuits decide among them. Device priority levels are fixed at system installation. There are four priority levels among UNIBUS devices. A unit with a high priority level always takes precedence over one with a lower priority level. In the case of units with equal priority levels, the one closest electrically to the processor on the bus takes precedence over those further away.

For example, if the processor has control of the bus when three devices, all of higher priority than the processor, request bus control, and the requesting devices are of different priorities, the processor will grant use of the bus to the one with the highest priority. If they are all of

the same priority, all three signals come to the processor along the same bus line, so that it sees only one request signal. Its reply, granting control of the bus, travels down the bus to the nearest requesting device, passing through any intervening non-requesting devices. The requesting device takes control of the bus, executes one or more bus cycles and relinquishes the bus. The request grant sequence occurs again, this time going to the second device down the line, which has been waiting its turn. When all higher-priority requests have been granted, control of the bus returns to the processor.

The processor usually has the lowest priority because, in general, it can stop whatever it is doing without serious consequences. Peripheral devices may be involved with some kind of mechanical motion, or they may be connected to a realtime process, either of which requires immediate attention to a request to avoid data loss.

Priority arbitration takes place asynchronously in parallel with the data transfer.

Bus Communication

Communication is interlocked, so that each control signal issued by the master must be acknowledged by a response from the slave to complete the transfer.

Bus Control

There are two ways of requesting bus control: non-processor requests (NPRs) or bus requests (BRs).

An NPR is issued when a device wishes to perform a data transaction. An NPR does not use the CPU; therefore, DMA activity can occur while the CPU continues to execute instructions.

A BR is issued when a device needs to interrupt the CPU for service. A device can interrupt the CPU only if it has gained control via a BR.

Interrupts

Interrupt handling is automatic in the VAX-11/730. No device polling is required to determine which service routine to execute. When interrupting, the device supplies a vector which directs the CPU to a memory location which contains the starting address of an interrupt service routine. When servicing the interrupt, the processor automatically raises its priority to the level of the interrupting device. For a complete discussion of how the CPU responds to interrupts, see the VAX Architecture Handbook.

Priority Control

The UNIBUS priority system determines which device obtains the bus. Figure 5-3 illustrates UNIBUS priority control.

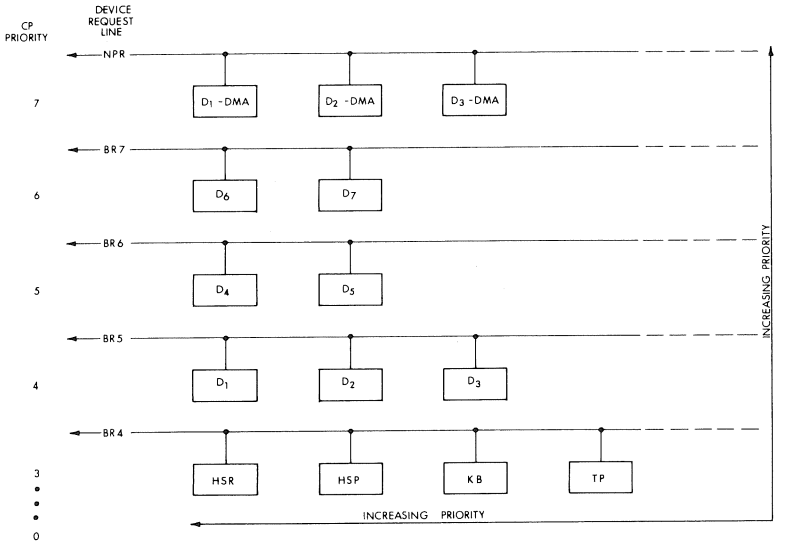


Figure 5-3 Priority Control

There are five UNIBUS vertical priority levels—NPR, and BR7, BR6, BR5, and BR4. To accommodate several peripheral devices, it may be necessary to connect more than one device to a single level.

Priority Assignments

When assigning priorities to a device, three factors must be considered: operating speed, ease of data recovery, and service requirements.

Data from some devices is available for only a short time period. Therefore, highest priorities are usually assigned to these devices for efficient data transfers. Lower priorities are assigned to devices with data available for longer periods of time, and to devices with automatic data recovery features. For example, a disk or magnetic tape device would be assigned a higher priority than a line printer or paper tape device. These priorities are assigned at installation time.

CPU Priority Level

In addition to device priority levels, the CPU has a programmable

priority. The CPU can be set to any one of 32 priority levels. Levels 14_{16} through 17_{16} correspond to UNIBUS levels BR4 through BR7.

Data Transactions

There are four types of UNIBUS data transactions:

- DATO—a data *word* is transferred from the master to the slave.
- DATOB—a data *byte* is transferred from the master to the slave.
- DATI—a data *word* or *byte* is transferred from the slave to the master.
- DATIP—is used to allow a read from slave or write from slave sequence to occur as a single bus cycle without any other UNIBUS activity intervening. This feature allows devices to synchronize their activities with the processor. DATIP must be followed by DATO or DATOB to the same location.

VAX-11/730 UNIBUS ADAPTER

There are several characteristics of the VAX architecture that require more than a "straight-through" connection from the UNIBUS to the memory.

Addresses that are contiguous in the virtual address space may be noncontiguous in the physical address space on 512-byte boundaries. Because UNIBUS devices often use sequential addresses, a means is provided to break up these sequential addresses into disjoint 512-byte blocks.

The VAX architecture imposes no restrictions on the alignment of data in memory. UNIBUS word transfer NPR devices, however, only transfer data on even addresses. The memory controller provides a mechanism that allows the transfer to be effectively shifted by one byte to accommodate requests for I/O buffers on odd byte addresses.

A method is also provided that allows a UNIBUS device access to all of the physical address space. This is necessary because the memory has 24-bit addresses, whereas the UNIBUS has 18-bit addresses.

PROCESSOR ACCESS TO UNIBUS

Any processor access with a physical address in the range of $FC0000_{16}$ through $FFFFFF_{16}$ will map directly to the UNIBUS in the range 0 through 777777_{8}. Any VAX-11/730 internal device (other than the processor) that references that address range will be ignored by the UNIBUS Interface. The device will receive a time-out error.

Processor Operations

The memory controller is designed to respond only to byte or aligned word transactions. Any other type of access will yield UNPREDICTA-

BLE results. The different types of operations are mapped into UNIBUS operations as follows:

Transaction	UNIBUS
Read	DATI (Data In)
Read with modify intent	DATIP (Data In Pause)
Read lock	DATIP
Write	DATO or DATOB (Data Out)
Write unlock	DATO(B)

When an operation that causes a DATIP occurs, BBSY is asserted and held until the end of the next operation that does not cause a DATIP to occur.

The choice of DATO or DATOB is made based on the number of bytes written.

UNIBUS Responses

A processor read to the UNIBUS that causes a no-response timeout will result in a machine check abort with a non-existent memory (NXM) bus error indicated. A processor write that causes such a timeout to occur will generate a write bus error interrupt request. If a UNIBUS device asserts PB in response to a processor access to it, it will cause a machine check abort, indicating an uncorrectable bus error.

UNIBUS INITIATED DATA TRANSFERS

As mentioned earlier in this chapter, the UNIBUS adapter performs a number of functions to make the UNIBUS architecture compatible with the VAX architecture. A key element in this matching operation is the UNIBUS Map. This map translates UNIBUS device-generated addresses into physical memory addresses. It also identifies UNIBUS transfers so that other features, such as odd byte addressing, can be used appropriately.

The UNIBUS has 18 address lines, creating an address space of 256 Kbytes. This is conceptually divided into two regions: the bottom 248 KB used to address UNIBUS memory, and the top 8 KB used to address UNIBUS peripheral device control and status registers (CSRs). A UNIBUS NPR device typically does a transfer to memory by doing a series of transactions placing addresses in the lower range on the bus. In the VAX-11/730, these are not the actual memory addresses, but serve as pointers to the UNIBUS Map, which in turn provides the actual physical memory address. If more than one device is doing

transfers at the same time, each one usually is set up to transfer into a different range within the 248 KB address space.

The UNIBUS address space is divided into 512 pages of 512 bytes each. When an address arrives at the UNIBUS adapter, the page number (the upper 9 bits of the 18-bit UNIBUS address) is taken as an index into the map, with the map then providing the actual Page Frame Number (PFN) of the physical memory address. This is concatenated with the address of the word or byte within the page from the UNIBUS address (the lower 9 bits of the 18-bit UNIBUS address) to create the final memory address.

A UNIBUS device initiates a request for memory by asserting UNIBUS NPR (Non-Processor Request). If memory (LOCK) is not busy, the arbitrator asserts NPG (Non-Processor Grant) to the requesting device. If the CPU is accessing memory, the UNIBUS is locked out. In this case, the UNIBUS device must wait until the lock is released to proceed with the memory access.

When NPG is asserted on the UNIBUS, the requesting device normally responds by asserting SACK to the controller. If SACK is not received from the device within a specified timeout period (12.8 to 25.6 μs), the arbitrator negates NPG thereby terminating the device's attempt to access memory.

If SACK is received within the timeout period, the arbitrator negates NPG causing the device to assert BBSY and to negate SACK. The assertion of BBSY signifies that the UNIBUS is busy and that the device is now bus master.

The device presents the two control bits (C1, C0) specifying the type of operation and the target address to the memory controller. If the device is to write to memory (a DATO(B)), it places the write data onto the data lines of the UNIBUS and raises MSYN to the controller. When the controller receives MSYN, it checks for the presence of CPU GRANT. The true state of CPU GRANT indicates a CPU/memory transaction is in progress, in which case the transaction must complete and CPU GRANT must be negated before the UNIBUS access to memory can continue. With CPU GRANT false, the memory controller replies to the device with SSYN whereupon the device negates MSYN. This is followed by the negation of SSYN by the controller and the negation of BBSY by the device to complete the UNIBUS/memory transaction.

If the C1 and C0 control bits specified a DATI(P), the device asserts MSYN to the memory controller which responds by placing the data onto the UNIBUS and raising SSYN. Otherwise, the read transaction is identical to a write.

UNIBUS Adapter Operating Detail

UNIBUS address bits <17:9> are used to enter a 512-byte by 23-bit wide memory location. The data coming out of that memory location determine how the transaction will be handled. The map data field is divided into three sections: Page Frame Number (PFN), Offset Bit, and Valid Bit. The format of the map data field is shown in Figure 5-4.

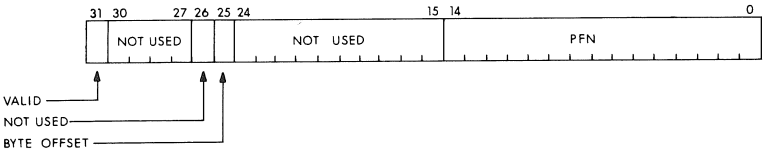


Figure 5-4 Map Data Field

Bit: 31 **Name:** VALID Bit
Function: This bit indicates map data is a genuine entry.

Bit: 30:26 **Name:** Not used
Function:

Bit: 25 **Name:** BYTE OFFSET
Function: This bit indicates UNIBUS reference is to an odd byte location (e.g., 1001, 1003). UNIBUS protocol allows UNIBUS address bits to present only even addresses to the memory controller during a word transfer.

Bit: 24:15 **Name:** Not used
Function:

Bit: 14:0 **Name:** PFN
Function: These bits select physical storage location.

On UNIBUS initiated transactions that cause a memory read or write cycle to occur, the map PFN is concatenated with UNIBUS address bit <8:0> to form a 24-bit wide physical address. Figure 5-5 shows this address translation process.

If the Offset Bit is a one, it causes the transaction to behave as if the UNIBUS address supplied by the DMA device were incremented by one. This allows devices that only produce even byte addresses to access buffers on odd byte boundaries. A transaction that causes a word to cross page boundaries because the Offset Bit is set must have the Offset Bit and Valid Bit identical in both map entries. Any differences will yield UNPREDICTABLE results. If this is a DATI(P) or a DATO and the two bytes of UNIBUS data fall across a longword boundary, two memory cycles will occur.

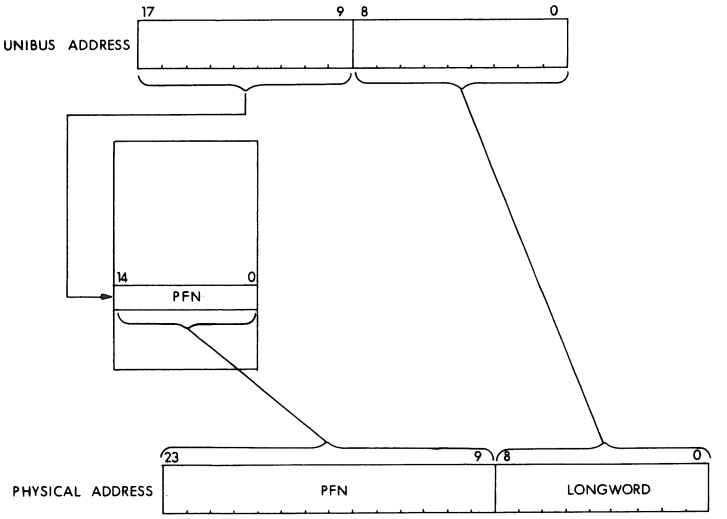


Figure 5-5 UNIBUS to Physical Address Translation

If the Valid Bit is a one, the VAX-11/730 integral UNIBUS adapter processes the transaction as described above. When it is zero, the integral UNIBUS adapter ignores UNIBUS requests. The Valid Bit must be set to zero for map entries that correspond to sections of UNIBUS address space in which there are slaves expected to respond to transactions originating on the UNIBUS. Transactions from the CPU that cause a UNIBUS transaction to occur are always ignored by the integral UNIBUS adapter and can never wrap back through the UNIBUS adapter to memory.

Interrupts

Interrupting devices on the UNIBUS are directly vectored through the System Control Block (SCB). The address of the vector is found at System Control Block Base (SCBB) + 200_{16} + device vector. The device vectors are the standard PDP-11 UNIBUS vectors. The VAX-11/730 integral UNIBUS adapter interrupt mechanism will not operate properly with any UNIBUS device vector at or above 200_{16} (1000_8). The VAX-11/730 UNIBUS adapter itself never generates an interrupt.

UNIBUS Control and Status Register

See Figure 4-5 and the corresponding bit definitions of CSR2 in Chapter 4 of this handbook.

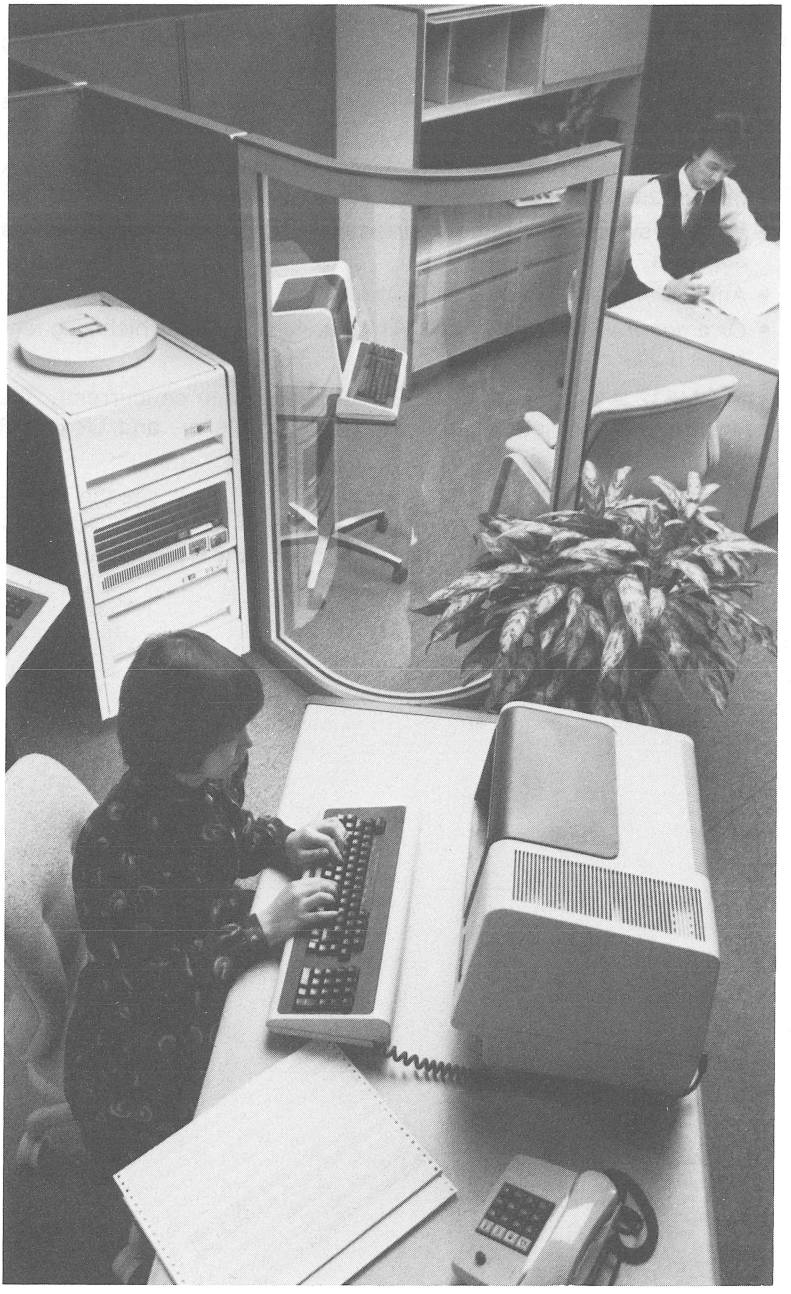
Optional DMF32 Communication Board

The VAX-11/730 UNIBUS accepts a number of VAX/VMS supported device and communication controllers, including the DMF32 communication board. The DMF32 consists of 4 UNIBUS controllers on one hex board:

- An 8-line asynchronous multiplexer featuring programmed DMA mode transmit lines
- A DMA synchronous line supported by DECnet communications software
- And *either* a DMA lineprinter controller
- *Or* a general purpose 16-bit DR11 parallel interface which also features buffer or DMA transfers

The DMF32 is programmed for 3 interfaces to run concurrently; the asynchronous multiplexer, synchronous multiplexer, and LP or DR device.

For more details on the DMF32 communication board, see the Terminals and Communications Handbook. For VAX-11/730 UNIBUS configuration guidelines, refer to the VAX Systems and Options Summary.



CHAPTER 6

VAX-11/730 PRIVILEGED REGISTERS

INTRODUCTION

The processor register space provides access to many types of CPU control and status registers such as the memory management base registers, console registers, clock registers, and the multiple stack pointers. In a VAX/VMS environment, the operating system conveniently manages these registers for the user; therefore, the detailed privileged register information contained in this chapter will be useful to system programmers.

A complete list of VAX-11/730 internal processor registers may be found in the back of this book.

SYSTEM IDENTIFICATION REGISTER (SID)

The system identification register is a read-only constant register that specifies the processor type. The entire SID register is included in the error log and the type field may be used by software to distinguish processor types. Figure 6-1 illustrates the system identification register.

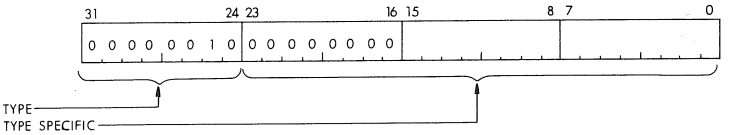


Figure 6-1 System Identification Register IPR#40_{16}

Type	A unique number assigned by engineering to identify a specific processor:
0	Reserved to DIGITAL
1	VAX-11/780
2	VAX-11/750
3	VAX-11/730
4 through 127	Reserved to DIGITAL
128 through 255	Reserved to CSS and customers

For the VAX-11/730, the type-specific format is shown in Figure 6-2.

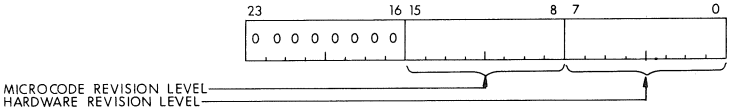


Figure 6-2 Type-Specific Format

CONSOLE TERMINAL REGISTERS

The console terminal is accessed through four internal registers. Two are associated with receiving from the terminal and two with writing to the terminal. In each direction there is a control/status register and a data buffer register. Figure 6-3 illustrates the console receive control/status register, and the bit assignments are described.

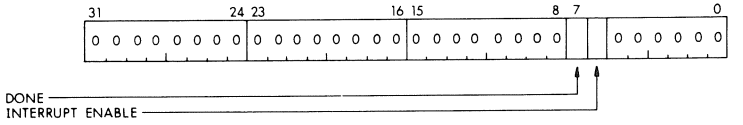


Figure 6-3 Console Receive Control/Status Register (RXCS)
IPR#20_{16}

Bit: 31:8 **Name:** MBZ
Function: Must be zero.

Bit: 7 **Name:** Done
Function: This bit is read-only and is set by the console whenever a datum is received. Done is initialized to 0 at bootstrap time and is cleared whenever MFPR #RXDB,dst is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 20 when Done becomes set. Similarly, if Done is already set and the software sets IE, an interrupt is generated. This bit is set to 0 by hardware initialization, and can be read or written by software.

Bit: 5:0 **Name:** MBZ
Function: Must be zero.

Figure 6-4 illustrates the read-only console receive data buffer register. The bit assignments follow.

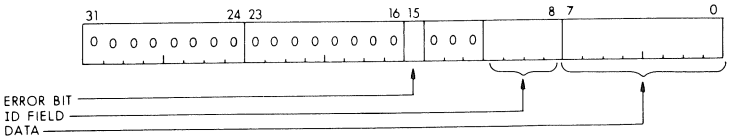


Figure 6-4 Console Receive Data Buffer Register (RXDB) IPR#21_{16}

Bit: 31:16 **Name:** MBZ
Function: Must be zero.

Bit: 15: **Name:** ERR
Function: Error bit. If the received data contained an error such as overrun or loss of connection, then ERR is set.

Bit: 14:12 **Name:** MBZ
Function: Must be zero.

Bit: 11:8 **Name:** ID
Function: If ID is zero, then the data is from the console terminal. If ID is nonzero, then the entire register is implementation-dependent.

Bit: 7:0 **Name:** Data
Function: This field contains the actual data received by the console.

Figure 6-5 illustrates the console transmit control/status register; the bit descriptions are also provided.

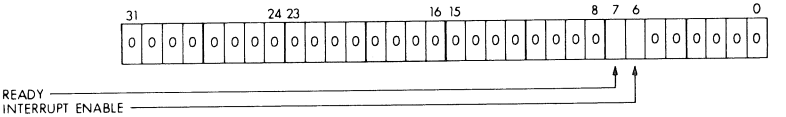


Figure 6-5 Console Transmit Control/Status Register (TXCS) IPR#22_{16}

Bit: 31:8 **Name:** MBZ
Function: Must be zero.

Bit: 7 **Name:** RDY
Function: Ready. This bit is read-only and is set at bootstrap time. It is also set whenever the console transmitter is not busy. This bit is cleared when MTPR src,#TXDB is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 20 when RDY becomes set. If RDY is already set and software sets IE, an interrupt is also generated. This bit is cleared when MTPR src,#TXDB is executed.

Bit: 5:0 **Name:** MBZ
Function: Must be zero.

Figure 6-6 illustrates the read-only console transmit data buffer register. The bit assignments are described.

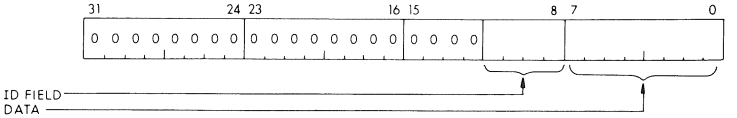


Figure 6-6 Console Transmit Data Buffer Register (TXDB) IPR#23_{16}

Bit: 31:12 **Name:** MBZ
Function: Must be zero

Bit: 11:8 **Name:** ID
Function: These bits act as a secondary decode to specify a number of different places that the data can be set.

<11:8>=0	Console terminal
<11:8>=1-E	Reserved operand abort
<11:8>=F	Boot control functions. The function is dependent on the data value written as follows:
0	No operation
1	Software Done
3	Clear Restart in Progress flag
2	Boot CPU
4	Clear Bootstrap in Progress flag
>4	Reserved operand abort

Bit: 7:0 **Name:** Data
Function: This field contains the actual data transmitted by the console. If ID is F, then 0, 1 or 3 is a no-operation; a 2 will cause a boot, and a 4 will clear a cold start flag. With ID = F, any other value will cause a reserved operand fault.

TU58 REGISTERS

The console TU58 tape cartridge subsystem is accessed through four internal registers. Two are associated with receiving from the TU58 and two with writing to it. In each direction there is a control/status register and a data buffer register. Figure 6-7 illustrates the console storage receive status register.

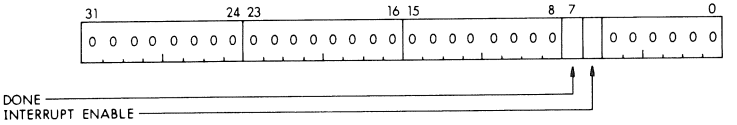


Figure 6-7 Console Storage Receive Status (CSRS) IPR#$1C_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero.

Bit: 7 **Name:** Done
Function: This bit is read-only and is set by the TU58 whenever a datum is received by the TU58 from the console storage receive data register. Done is initialized to 0 at bootstrap time and is cleared whenever MFPR #CSRD, dst is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 23. Similarly, if Done is already set and the software sets IE, an interrupt is generated. This bit is set to 0 by hardware initialization, and can be read or written to by software.

Bit: 5:0 **Name:** MBZ
Function: Must be zero.

Figure 6-8 illustrates the console storage receive data register.

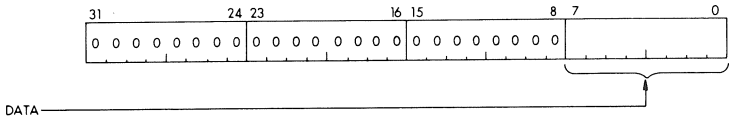


Figure 6-8 Console Storage Receive Data (CSRD) IPR# $1D_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero.

Bit: 7:0 **Name:** Data
Function: This field contains the actual data received from the TU58 subsystem.

Figure 6-9 illustrates the console storage transmit status register.

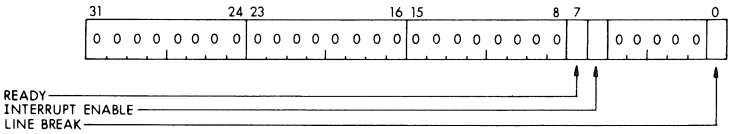


Figure 6-9 Console Storage Transmit Status (CSTS) IPR#1E$_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero.

Bit: 7 **Name:** RDY
Function: Ready. This bit is read-only and is set at bootstrap time. It is also set whenever the TU58 transmitter is not busy. This bit is cleared when MTPR src, #CSTD is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 23. If RDY is already set and software sets IE, an interrupt is also generated. This bit is cleared when MTPR src, #CSTD is executed.

Bit: 5:1 **Name:** MBZ
Function: Must be zero.

Bit: 0 **Name:** LB
Function: Line Break. When this bit is written to a 1, a line break is issued to the TU58. This bit is write-only.

Figure 6-10 illustrates the console storage transmit data register.

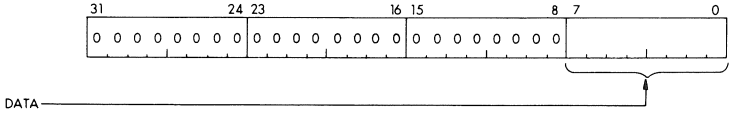


Figure 6-10 Console Storage Transmit Data (CSTD) IPR#1F$_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero.

Bit: 7:0 **Name:** Data
Function: This field contains the actual data transmitted by the TU58 subsystem.

TIME-OF-YEAR CLOCK AND INTERVAL TIMER REGISTERS

The interval timer is used for accounting, for time-dependent events, and to maintain the software date and time. The time-of-year clock is used by software to perform various timekeeping functions.

Time-of-Year Clock

The time-of-year clock consists of one longword register. The register forms an unsigned 32-bit binary counter that is driven by a precision clock source. The least significant bit of the counter represents a resolution of 10 ms. Thus, the counter cycles to zero after approximately 497 days.

If power has been lost so that time is not accurate, the register is cleared on power-up. It is held at zero until software writes a nonzero value to it. Thus, if software initializes this clock to a value corresponding to a large unit of time (e.g., a month), it can check for loss of time after a power restore by checking the clock value. The time-of-year clock register is illustrated in Figure 6-11.

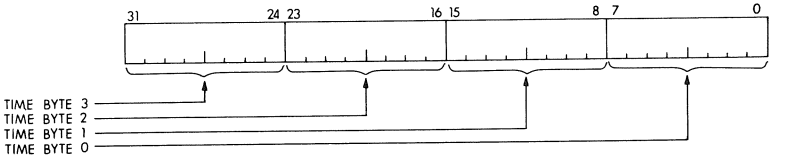


Figure 6-11 Time-of-Year Clock Register (TODR) IPR#$1B_{16}$

Interval Timer

The interval timer provides an interrupt at IPL 24 at programmed intervals. The counter is incremented at 1 μs intervals, with a typical accuracy of .01% or 8.64 seconds per day. (Accuracy will vary slightly with ambient temperature.) The clock interface consists of three registers in the privileged register space: the read-only Interval Count Register, the write-only Next Interval Count Register, and the Interval Count Control/Status Register.

Interval Count Register (ICR)

The Interval Count Register is a read/write register incremented once every microsecond. It is automatically loaded from NICR (Next Interval Count Register) upon a carry out from bit 31 (overflow) which also causes an interrupt request at IPL 24 if the interrupt is enabled. Figure 6-12 illustrates the Interval Count Register.

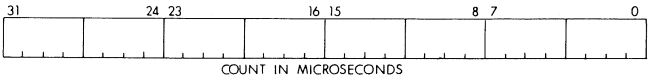


Figure 6-12 Interval Count Register (ICR) IPR#$1A_{16}$

Next Interval Count Register (NICR)
The reload register is a write-only register that holds the value to be loaded into ICR when it overflows. The value is retained when ICR is loaded. NICR is capable of being loaded regardless of the current values of ICR and ICCS (Interval Count Control/Status Register). Figure 6-13 illustrates the Next Interval Count Register.

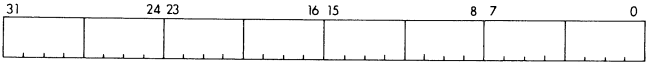


Figure 6-13 Next Interval Count Register (NICR) IPR#19_{16}

Interval Count Control/Status Register (ICCS)
The ICCS register contains control and status information for the interval counter. Figure 6-14 illustrates the Interval Count Control/Status Register with the bit assignments described below.

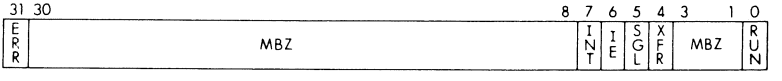


Figure 6-14 Interval Count Control/Status Register (ICCS) IPR#18_{16}

Bit: 31 **Name:** ERR
Function: Whenever ICR overflows, if INT is already set, then ERR is set. Thus, ERR indicates one or more missed clock ticks. Attempts to set this bit via MTPR clears ERR.

Bit: 30:8 **Name:** MBZ
Function: Must be zero

Bit: 7 **Name:** INT
Function: This bit is set by hardware every time ICR overflows. If IE is set then an interrupt is also generated. An attempt to set this bit via MTPR clears INT, thereby re-enabling the clock tick interrupt (if IE is set).

Bit: 6 **Name:** IE
Function: When this bit is set, an interrupt request at IPL 24 is generated every time ICR overflows (INT is set). When clear, no interrupt is requested. Similarly, if INT is already set and the software sets IE, an interrupt is generated (i.e., an interrupt is generated whenever the function (IE .AND. INT) changes from 0 to 1).

Bit: 5 **Name:** SGL
Function: This bit is write-only. If Run is clear, each time this bit is set, ICR is incremented by one.

Bit: 4 **Name:** XFR
Function: This bit is write-only. Each time this bit is set, NICR is transferred to ICR.

Bit: 3:1 **Name:** MBZ
Function: Must be zero

Bit: 0 **Name:** Run
Function: When this bit is set, ICR increments each microsecond. When clear, ICR does not increment automatically. At bootstrap time, Run is cleared.

Thus, to set up the interval timer, load the negative of the desired interval into NICR. Then an MTPR #↑X51,#ICCS will enable interrupts, reload ICR with the NICR interval and set Run. Every "interval count" microseconds will cause INT to be set and an interrupt to be requested. The interrupt routing should execute an MTPR #↑XC1,#ICCS to clear the interrupt. If INT has not been cleared (i.e., if the interrupt has not been handled) by the time of the next ICR overflow, the ERR bit will be set.

At bootstrap time, bits <6> and <0> of ICCS are cleared. The rest of ICCS and the contents of NICR and ICR are UNPREDICTABLE.

VAX-11/730 OPTIONAL FLOATING POINT ACCELERATOR

The VAX-11/730 has an optional accelerator for a subset of the instructions. The ACCS, an internal read/write register, controls the accelerator.

ACCS is the accelerator control/status register. It indicates whether an accelerator exists, controls whether it is enabled, identifies its type and reports errors and status. At bootstrap time, the type and enable are set; the errors are cleared. Figure 6-15 illustrates the accelerator control/status register.

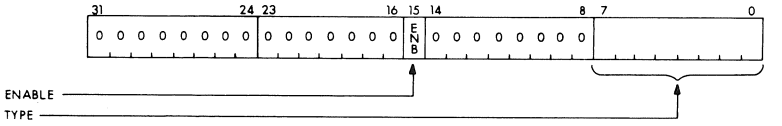


Figure 6-15 Accelerator Control/Status Register (ACCS)

Bit: 31:16 **Name:** MBZ
Function: Must be zero.

Bit: 15 **Name:** ENB
Function: Read-only field specifying whether the accelerator is enabled. This is set if the accelerator is installed and functioning. An attempt to set this is ignored.

0 = No accelerator
1 = Floating Point Accelerator

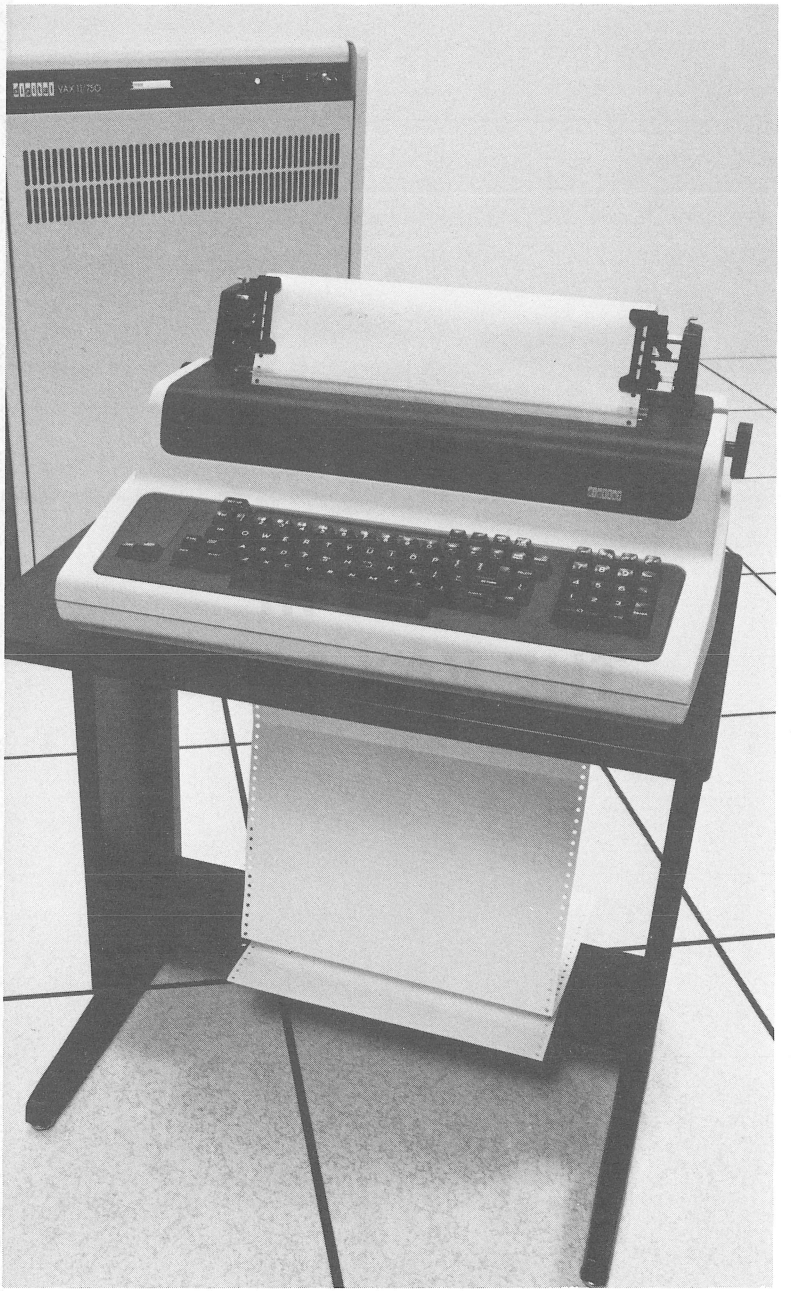
Bit: 14:8 **Name:** MBZ
Function: Must be zero.

Bit: 7:0 **Name:** TYPE
Function: Read-only field specifying the accelerator type as follows:

0 = No accelerator
1 = Floating Point Accelerator

Numbers in the range 2 through 127 are reserved to DIGITAL. Numbers in the range 128 through 255 are reserved to CSS/customers.

PART III
THE VAX-11/750



CHAPTER 7

VAX-11/750 CONSOLE SUBSYSTEM

FEATURES	BENEFITS
Console terminal	Dual function as console and user terminal results in hardware savings
Console command language	Gives the user a powerful, yet easy-to-use, debugging tool
Console terminal is a standard ASCII device	Provides a high degree of flexibility
EIA communications interfacing	Allows standard industry-compatible communications
Front panel switches and indicator lights	Offer the user easy system access and control
TU58 cartridge tape drive	Provides an inexpensive, reliable device and medium for booting, diagnostics, field updates to any software, and convenient personal data storage on cartridge
Unattended restart	The system restarts or reboots itself upon recovery of electricity after a power failure or other system crash

INTRODUCTION

The VAX-11/750 console subsystem has four major elements:

1. A front panel on the CPU cabinet with switches and indicator lights.
2. A separate ASCII terminal with keyboard, called the console terminal.
3. A TU58 tape cartridge drive in the CPU cabinet.
4. A special console command language with simple commands the user types from the console terminal.

This approachable console subsystem is designed to allow the user to interactively communicate instructions to the CPU using the console terminal and the console command language. The integral TU58 tape cartridge drive can be used optionally to boot the system, as well as to

load diagnostics, updates to the operating system software and compilers. DIGITAL also supports the TU58 under VAX/VMS for data storage and retrieval.

Figure 7-1 illustrates the hardware elements of the VAX-11/750 console subsystem.

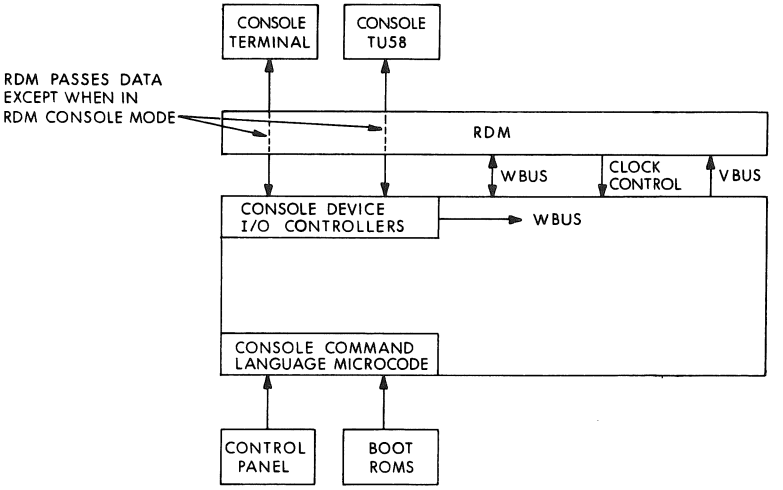


Figure 7-1 VAX-11/750 Console Subsystem

CONSOLE MODES

The VAX-11/750 console runs in three modes: program I/O mode, console I/O mode, and RDM console mode. These modes are mutually exclusive. One of the three will always be enabled while there is power to the machine. In the program I/O mode (also known as the system terminal mode), the console functions like the other terminals on the VAX-11/750 system. In this mode the console passes characters between the terminal and the instruction set processor running in the CPU.

In the console I/O mode (also called console mode), the console program interprets and acts on commands typed on the console terminal. The VAX-11/750 will enter the console I/O mode whenever the instruction set processor halts.

Table 7-1 lists the actions that cause the CPU to halt and enter console I/O mode. Pressing the reset button simulates the power-

down/power-up sequence. When the keylock switch is in either "secure" position, CTRL-P and the reset button are disabled.

When the processor enters the console I/O mode it types on the console terminal the address contained in the program counter (PC), a two-digit code which identifies the reason for the halt, and the console prompt symbol, >>>. The prompt symbol shows that the console program is looping, waiting for a command.

Table 7-1 Console Halt Codes

Code	Meaning
0	Halt or Single Step from console
1	Successful T command
2	CTRL-P was typed on the console
4	Interrupt stack not valid or unable to read SCB
5	Double bus write error halt
6	Halt instruction with PSL<CM> = 0
7	SCB vector <1:0> = 3
8	SCB vector <1:0> = 2 and no UCS
A	CHMX while on the interrupt stack
B	CHMX SCB vector <1:0> .NE.0
11	Power up and can't find RPB, FPS1 at RESTART/HALT
12	Power up and warm start flag false FPS1 at RESTART/HALT
13	Power up and can't find good 64K of memory
14	Power up and booting, but bad Boot ROM or no ROM
15	Power up and cold start flag set during boot subroutine
16	Power up halt; Power On Action Switch at HALT position
FF	Micro Verify test failure

The third console mode is the RDM console mode. This mode is functional when the Remote Diagnosis Module (RDM) option is present in

the system configuration. In the RDM console mode the microprocessor on the RDM board interprets and acts on the commands typed on the console terminal.

VAX-11/750 FRONT PANEL

The front panel of the VAX-11/750 has four switches and seven indicator lights which allow the processor to be halted, restarted, booted, and initialized. If a user wants the CPU to restart in case of a power failure, the POWER ON ACTION switch located on the front panel can be set to automatically restart the system. The OFF-LOCAL-REMOTE switch determines the operating mode of the console terminal: *system terminal mode* or *console mode*. Furthermore, by removing the key from the OFF-LOCAL-REMOTE switch, the user can fix the operating state, preventing anyone else from changing that state. The operator can use the console terminal as a user terminal in a protected environment once the system is running. Additional user terminals on the system cannot function as console terminals.

Figure 7-2 illustrates the VAX-11/750 front panel controls and indicators.

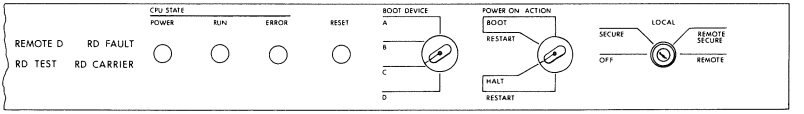


Figure 7-2 VAX-11/750 Front Panel Controls and Indicators

Off-Local-Remote

This is a five-position keylock switch.

- OFF—No power to the CPU (except battery backup to the time-of-year clock) or to memory.
- LOCAL—The CPU responds to console commands and the remote diagnostic unit is completely disabled.
- SECURE—Console commands are ignored. The remote diagnostic unit is completely disabled, and the RESET switch is disabled.
- REMOTE—The console functions are enabled and may be activated from the remote line only. However, the remote line does have the capability to return control to the local terminal.
- REMOTE/SECURE—Console commands are ignored. The remote line replaces the console terminal, and the RESET switch is disabled. With the switch in this position, the remote diagnosis module may not be used to identify the source of system failures.

Power On Action

This four-position rotary switch controls the machine on a power-up sequence.

- HALT—The machine comes up halted, in console mode.
- BOOT—The machine bootstraps from the device selected by the boot device switch.
- HALT/RESTART—A check is made to see if there is a valid Restart Parameter Block (RPB). If so, a restart sequence is initiated. Otherwise, the machine halts.
- BOOT/RESTART—Similar to Halt/Restart except that a bootstrap sequence occurs if RESTART is not successful.

Boot Device

This four-position rotary switch selects the device from which to boot. The VAX-11/750 CPU contains four sockets for built-in Read-Only Memory chips (ROMs) that contain the VAX-11/750 code needed to bootstrap a device. The switch selects which of the four ROMs is to provide the bootstrap code when a boot sequence is initiated. Typically, this switch is left in the position corresponding to the VAX/VMS system disk. Position A always causes a boot from the TU58 tape cartridge.

Reset

This pushbutton switch activates a system power-down sequence followed by a power-up sequence unless the keylock switch is on SECURE. The system will come up in the state selected by the POWER ON ACTION switch. It is usually used only if the machine appears to be hung and does not respond to console commands.

CPU State Indicator Lights

- POWER—This green light indicates that DC power is present inside the CPU and that the keylock switch is not in the OFF position.
- RUN—When lit, this green light indicates the machine is in the program I/O mode.
- ERROR—When brightly lit, this red light indicates that the machine is stopped because of an unrecoverable control store parity error. To reset the machine, the RESET switch must be used because console commands are not operational.

  When glowing softly, the red light means that the CPU is functioning normally.

Remote Diagnostic Indicators

The following four lights are back-lit words, illuminated during various

stages of remote diagnostic procedures. It should be noted that the REMOTE D, RD CARRIER, RD TEST, and RD FAULT lights only apply to systems having the optional remote diagnosis module installed by Field Service.

- REMOTE D—Remote diagnostics (RD) software illuminates this green light whenever the OFF-ON-LOCK switch is in one of the two remote positions.
- RD CARRIER—This green light is lit by the RD software whenever it detects that the remote port carrier is present. It indicates that the Remote Diagnostic Center (RDC) has established connection.
- RD TEST—The RDC software illuminates this amber light while tests are in progress.
- RD FAULT—This red light is lit by the Remote Diagnosis Module (RDM) if it detects a fault in its own logic. No tests should be attempted when the fault indicator is lit.

CONSOLE TERMINAL

The console terminal and the console command language are two powerful tools within the VAX-11/750 console subsystem. By typing in simple console command language instructions through the console terminal, the user can instruct the CPU to perform a wide variety of functions. These instructions to the CPU can only be communicated when the console terminal is in console mode. In this mode, the console terminal is dedicated exclusively to controlling CPU functions such as examining and depositing data in memory. Console mode also allows the processor to be started, self-tested, initialized to a known state, single-stepped through instructions, or halted. In contrast, system terminal mode dedicates the console terminal to user application processes and the VAX/VMS operating system. In system terminal mode, the CPU ignores console commands.

Console Terminal Communications

The electrical interface for the console terminal is an industry-standard full-duplex EIA RS232-C line. The speed of the line is jumper-selectable to 300, 400, 600, 1200, 2400, 3600, 4000, 4800, 7200, 9600, 19,200, or 38,400 bits per second. This interface is a special port connected directly to the central processor.

CONSOLE COMMAND LANGUAGE

The VAX-11/75O console command language gives the user an efficient way of communicating with the console subsystem—using simple commands entered through the console terminal instead of the traditional toggle switches and lights. Thus, the VAX-11/750's console

command language is a powerful tool, allowing the user to EXAMINE, DEPOSIT, INITIALIZE, CONTINUE, START, HALT, SINGLE STEP, SELF TEST, and BOOT as briefly described below.

COMMAND	DESCRIPTION
Examine	Allows the user to look at physical and virtual memory, the processor registers, the general registers, and the Processor Status Longword
Deposit	Allows the user to make an entry into physical or virtual memory, the processor registers, the general registers, and the Processor Status Longword
Initialize	Allows the user to put the processor into a known state
Continue	Allows the user to restart a halted program without altering the state of the machine
Start	Initializes the processor and enables it to begin
Halt	No operation
Single Step	Allows the user to execute one instruction at a time
Boot	Allows the user to load the operating system and start the CPU
Self Test	Runs micro verify routines and initializes the processor
Binary Load	Permits the user to deposit a block of binary data in physical memory
Binary Unload	Allows the user to examine a block of binary data in physical memory

NOTE

The Binary Load/Unload commands are for special diagnostic use, and are not normally useful to the customer.

The console terminal utilizes the console command language to communicate instructions to the CPU. Commands are specified by a single letter with optional modifiers. When the CPU is not executing

instructions, it is halted, and the console terminal is in console mode. In this mode, the CPU is receptive to console commands and can perform the functions previously listed. Direct Memory Access (DMA) activity to memory can also occur with the terminal in console mode, and all DMA transactions in progress will continue even when the CPU is halted. This allows the machine to be halted without destroying these transactions; however, interrupts will not be serviced while the machine is halted. They will be serviced following a CONTINUE from the console.

Console Command Syntax and Semantics

Symbol	Function
< >	Angle brackets are used to denote category names. For example, the category name <ADDRESS> may be used to represent any valid address, instead of actually listing all the strings of characters that can represent an address.
[ ]	Parts of expressions in brackets do not need to be typed if defaults are used.
<SPACE>	Represents one typed space.
<COUNT>	Represents a numeric count in hexadecimal, 32 bits. Leading zeros may be omitted.
<ADDRESS>	Represents an address argument. Valid <ADDRESS> types are explained later in relation to specific commands. However, virtual addresses that reference nonexistent or nonresident pages will cause the console to abort execution of the console command which referenced that address. An appropriate error message will be displayed. Leading zeros may be omitted.
<DATA>	Represents a numeric argument. Leading zeros may be omitted.
<QUALIFIER>	Represents a command modifier (also called a *switch*). Valid <QUALIFIER> types are explained later in relation to specific commands.
<INPUT-PROMPT>	Represents the console's input prompt string — >>>.

<CR>	Carriage return.
<LF>	Line feed.

Typing Errors and Illegal Characters

Typing errors may be corrected (before a <CR> sends the entire command to the CPU) using the DELETE key. When the key is typed, the console will echo the character being deleted (after printing a backslash, \, upon receipt of the first deletion). The console will also add a backslash between the last deletion and the next input character.

Example:

operator types	1278<del><del>34
console prints	1278\87\34
console sees	1234

The console attempts to interpret each character as it is typed. If the console cannot interpret the next character in the context of the current command, it sounds a "beep" (bell) and ignores the character. This does not abort the entire command; instead, the command may be completed by typing the correct character(s).

Control Characters

CTRL-P	This is typed by holding down the CTRL key while typing a P. This command puts the machine in console I/O mode, halts the processor (see the HALT command), and causes the console terminal to type a halt message (PC<SPACE>02). Typing a CTRL-P while already in console I/O mode causes the console defaults to be reinitialized and the halt message to be typed on the console terminal.
CTRL-U	Tells the system to ignore all characters typed since the last <CR> and types \<CR><LF><INPUT-PROMPT>.
CTRL-D	CTRL-D causes the console to enter the RDM console mode, if the Remote Diagnosis Module is installed.

Console Command Language Instructions

BOOT Command

B[/X] [/n] [<SPACE><ddcu>] <CR>

This command boots the operating system or diagnostic software from the device specified. The hexadecimal number, <n>, specifies the boot control flags. The command deposits this number in register R5 before booting. For example, 200 sets bit 9 in R5, specifying boot control flag 9. <n> may be from one to four digits in length. If <n> is omitted, the default value for R5 is 0. The software boot control flags are actually implemented by VMS, not the VAX-11/750 CPU.

If /X is typed, the BOOT command will inhibit Micro Verify. This means that the Micro Verify tests are normally run at the beginning of the boot sequence.

<ddcu> represents the boot device, where dd is a two letter device mnemonic. Table 7-2 shows the boot device codes. The I/O channel adapter is specified by c. A, B, C, and D are possible values. U is a one-digit number that specifies the device drive number. If this is omitted, the device will be selected by the boot device switch, the channel adapter is A, and the drive number is 0. If /n or /X is used, then <ddcu> must be supplied.

Table 7-2 Boot Device Codes

Device Code (dd)	Device
DL	RL02
DM	RK06/7
DB	RP04/5/6/7, RM03
DD	TU58

Following a successful boot, the console enters program I/O mode (system terminal mode). If Micro Verify has been executed (/X was not typed), the console also prints a message telling whether the test revealed any errors (see the Test command).

CONTINUE Command

C**<CR>**

The CONTINUE command restarts execution of a halted program at the address currently contained in the Program Counter. The CPU is not initialized, and the console terminal enters system terminal mode once the CONTINUE command is issued.

DEPOSIT and EXAMINE Commands

D[<QUALIFIER-LIST>]<SPACE><ADDRESS><SPACE><DATA><CR>

E[<QUALIFIER-LIST>][<SPACE><ADDRESS>]<CR>

DEPOSIT and EXAMINE commands will be treated together because their formats are quite similar. Both commands require definition of the address space and the size of the operand in addition to the address. To make multiple EXAMINEs or DEPOSITs easier, there is a system of defaults for each of the items that must be specified for these commands. Some of the defaults are automatic (such as longword size for general registers), and some are set up by the immediately preceding EXAMINE or DEPOSIT. All other console commands (except <CR>) cause the defaults to be set to their initial values. DEPOSIT and EXAMINE qualifiers are listed in Table 7-3.

Table 7-3 DEPOSIT and EXAMINE QUALIFIERS

Size Qualifiers	
/B	Byte
/W	Word
/L	Longword
Space Qualifiers	
/V	Virtual Address
/P	Physical Address
/I	IPR
/G	GPR
Address Values	
nnnnnnnn	Hex Number
*	Last Location
P	PSL
+	Next Location
(DEPOSIT only)	
Data	
nnnnnnnn	Hex Number

DEPOSITs (writes) or EXAMINEs (reads) <DATA> at the <ADDRESS> specified. The address space and size used will depend upon the qualifier or qualifiers specified with the command. If no address space qualifier is used, the default is physical address space; following another DEPOSIT or EXAMINE, the same space as that of the previous command will be used as the default.

If no size qualifier is typed, the default for a physical or virtual DEPOSIT or EXAMINE is longword; however, following another EXAMINE or DEPOSIT, the size used in the previous command will be the default. The size for an IPR or GPR DEPOSIT/EXAMINE is always longword, and these commands do not change the current size default.

<ADDRESS> must be one to eight hexadecimal digits. The initial default is zero; however, the default is unpredictable when the address space is changed. Following another virtual or physical DEPOSIT or EXAMINE, the default is the sum of the address from the last EXAMINE/DEPOSIT plus size from the last EXAMINE/DEPOSIT. Typing a + for <ADDRESS> (for DEPOSIT only) will also get this default. Following another IPR or GPR EXAMINE/DEPOSIT, the default is the sum of the address from the last EXAMINE/DEPOSIT plus one. Using a P for <ADDRESS> does a longword reference of the Processor Status Longword, independent of address space and size. Using P has no effect on any of the defaults.

<DATA> must be represented by one to eight hex digits. If more digits than specified by the size are supplied, the extra digits on the left are ignored; if fewer digits are supplied, zeros are appended to the left.

Sample DEPOSIT response: A successful DEPOSIT always receives <INPUT-PROMPT>.

Sample EXAMINE responses: console output underlined

```
>>>     E/P 1234            ! Examine physical
                            ! address 1234

        P 00001234
        ABCDEF89

>>>     E/V 1234            ! Examine virtual
                            ! address 1234

        P 00005634          ! Note that
        01234567            ! virtual
                            ! EXAMINEs display
                            ! the translated
                            ! physical
                            ! address

>>>     E/G 0               ! Examine general
                            ! register
                            ! R0

        G 00000000
        98765432
```

Index of EXAMINEs and DEPOSITs

E*<CR>	Examine the last location examined or deposited into
E<CR>	Examine the next location
E<SPACE><ADDRESS><CR>	Examine <ADDRESS>; all switches are defaulted to last EXAMINE or DEPOSIT
E/G<SPACE><ADDRESS><CR>	Examine GPR; register number is <ADDRESS>; <ADDRESS> must be a value from 0 to †XF
E/I<SPACE><ADDRESS><CR>	Examine IPR; register number is <ADDRESS>
E/P<SPACE><ADDRESS><CR>	Examine physical <ADDRESS>
E/V<SPACE><ADDRESS><CR>	Examine virtual <ADDRESS>.
E<SPACE>PSL<CR>	Examine PSL
E/W/P<SPACE><ADDRESS><CR>	Examine a word at physical <ADDRESS>
E/P/W<SPACE><ADDRESS><CR>	Examine a word at physical <ADDRESS>
E/L/V<SPACE><ADDRESS><CR>	Examine a longword at virtual <ADDRESS>
D*<DATA><CR>	Deposit <DATA> in the last location that was deposited into or examined
D+<DATA><CR>	Deposit <DATA> in the next sequential address
D/G<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in GPR <ADDRESS>
D/I<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in IPR <ADDRESS>
D/P<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in physical <ADDRESS>
D/V<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in virtual <ADDRESS>
D<SPACE>PSL<SPACE><DATA><CR>	Deposit <DATA> in PSL
D<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit <DATA> in <ADDRESS> Switches are defaulted to previous switches
D/V/W<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit a word of <DATA> in virtual <ADDRESS>
D/L/P<SPACE><ADDRESS><SPACE><DATA><CR>	Deposit a longword of <DATA> in physical <ADDRESS>

INITIALIZE Command

I<CR>

This command performs the following functions:

- Initialize the processor
- Clear the translation buffer
- Clear the cache; turn on the cache
- Set the program counter to 0
- Set the processor status longword to 041F0000

HALT Command
H<CR>

The Halt command is implemented in the VAX-11/750 for the sake of consistency with the VAX-11/780. It does not actually halt the CPU since the CPU must already be halted to respond to the command. However, the Halt command does reset the console defaults.

Table 7-1 defines the various console halt codes.

NEXT Command
N<CR>

The NEXT command causes the CPU to execute one ISP level instruction. The CPU then halts and re-enters the console mode.

START Command
S[<SPACE><ADDRESS>]<CR>

The START command is normally used to start execution of programs that run without the operating system and without the diagnostic supervisor. START performs the equivalent of the following console commands:

1. Initialize the CPU.
2. Deposit <address> into the Program Counter (PC). If no address is specified, the current value of the PC is used.
3. Perform the Continue function to begin ISP level CPU execution.

Programs which may be started this way must be loaded into main memory before the START command is given. For example:

```
>>>S 1000          ! Start the program that
                   ! begins at
                   ! address 1000.
```

TEST Command

T<CR>

This command runs the Micro Verify program and initializes the processor. The Micro Verify program types a percent sign (%) on the terminal when it starts the test.

If all the tests run successfully, Micro Verify types a second %. If Micro Verify detects a failure, it types a single error character and then halts the processor in the console I/O mode. When the processor halts, the console types an error code in place of the PC and a halt code. The halt code shows that this halt is due to an error condition in Micro Verify.

Response if successful:
<CR><LF>%%<CR><LF><INPUT-PROMPT>

Response if unsuccessful:
<CR><LF>%<ERROR INDICATOR><CONSOLE HALT MESSAGE>

BINARY LOAD/UNLOAD Command
X<SP><ADDRESS><SPACE><COUNT><CR><CHECKSUM>

<ADDRESS>	starting address of load
<COUNT>	number of bytes to be transferred
<CHECKSUM>	2's complement checksum of command string or binary data

The BINARY LOAD/UNLOAD command instructs the console to load binary data into or unload binary data from physical memory, starting from the location specified by <ADDRESS>. A <COUNT> with bit <31> set indicates BINARY UNLOAD (data sent to the register). The remaining bits in <COUNT> indicate the number of bytes to Load or Unload. After a correct 2's complement checksum calculation, the console issues an <INPUT-PROMPT>, but remains in binary mode and either sends data to the user or prepares to receive data. If the checksum shows an error, a message and an <INPUT-PROMPT> are issued.

In LOAD, a binary string of data, of length <COUNT> + 1, will be sent once the <INPUT-PROMPT> indicates that the console has accepted the command. When <COUNT> is exhausted, the final byte is a console-calculated block checksum of all the data. A successful transfer issues an <INPUT-PROMPT>. An error in the checksum will generate an error message (see Table 7-4 for error codes).

For UNLOAD, the console processes the command and <CHECKSUM>. Then, if the checksum is correct, the console responds with <INPUT-PROMPT>, followed by a string of bytes which is the binary data requested. A second checksum is calculated and processed as for the LOAD sequence.

Console Command Errors
When a command is given that the console cannot properly process, it responds by typing ?nn. nn is an error code that describes the nature of the problem. Error codes are listed in Table 7-4.

Table 7-4 Console Command Error Codes

10	Illegal GPR register number
11	Illegal access of an IPR
19	Reference to next location was preceded by an EXAMINE or DEPOSIT to an IPR or PSL
20	ACV, TNV, or Machine Check during a read or write
30	Binary transfer checksum error
33	Unrecognizable boot device

NOTE

The console ignores illegal characters and echoes them as bell codes.

INTEGRAL TU58 CARTRIDGE TAPE DRIVE

The TU58 tape cartridge drive is an important part of the console subsystem. Because the TU58 is connected directly to the CPU, it retains the ability to run diagnostics even with some system components inoperative. This feature significantly increases system maintainability. The TU58 may also be used to boot the system, to load files into physical memory, and to store files which describe and execute site-specific bootstrap procedures (see BOOTING THE VAX-11/750 SYSTEM later in this chapter).

The tape cartridge is preformatted and contains 256 KB, normally formatted in 512 B records. The controller provides random access to any record. The TU58 searches at 60 inches per second (i/s) to find the file requested, then reads at 30 i/s. Data read from the tape are verified through checksums at the end of each record or header.

BOOTING THE VAX-11/750 SYSTEM

Initializing or booting the VAX-11/750 system can be viewed from two different perspectives. First, there are the actual steps that the system operator must perform in order to boot the system. And second, there are the actual events that occur within the system during the boot process.

From the system operator's standpoint, the booting process is quite simple. For a typical system boot:

1. Turn on power to the console terminal.

2. Turn the POWER ON ACTION switch on the VAX-11/750 front panel to the HALT position. The POWER ON light should now be lit. The console terminal will print:

```
%%
00000000  16
>>>
```

3. Check that the VMS operating system disk pack is in drive 0 and the READY light is on. (The disk READY light can take from 20 to 60 seconds to light from Power On until a Ready condition exists.)
4. Set the BOOT DEVICE switch to the position corresponding to the system device. This action matches the correct boot ROM with the VMS system device. The BOOT DEVICE switch settings are: A (TU58), B (RK07), C (RL02), and D (MASSBUS).
5. Set the POWER ON ACTION switch to the RESTART/BOOT position.
6. Press the RESTART button.
7. The console terminal should type:

```
%%
00000000  16
%%
```

   This is followed by the VAX/VMS boot message.

If this procedure does not work, it is possible that either drive 0 is not functioning properly or that the VAX system pack is no longer valid.

Console Subsystem Action on Boot

The overall flow of control for booting passes through a number of stages on the VAX-11/750. An overview of this flow is followed by a detailed specification for each part that is part of the VAX-11/750 hardware.

The console subsystem initializes the CPU, finds 64 KB of good memory, and passes the address of that memory to the Device ROM which brings in block 0 (the bootblock) of the boot device into the first page of good memory and then transfers control to it.

The bootblock code uses the Device ROM as a callable driver for the device and then uses it to bring in whatever file it needs to bring the system up.

There are five distinct ways that a boot sequence may be initiated on the VAX-11/750. They are:

1. Power-up sequence (RESET button or power restoration) with

switch set to BOOT.

2. Typing the "B" command while in console mode.
3. Execution of a HALT instruction when the processor is in kernel mode and the POWER ON ACTION switch is in the BOOT position.
4. Executing the MTPR (Move to Processor Register) to the console register that invokes a boot.
5. Failure of a warm start (restart) while the POWER ON ACTION switch is in the RESTART/BOOT position.

All five of these mechanisms will initiate the following sequence of actions. The first two will not perform the first step listed. This implies that the first two will cause a boot independent of the state of the cold start bit, while the last three will not.

1. Clear the cold start flag.
2. Micro Verify.
3. Initialize PSL to $041F0000_{16}$.
4. Locate 64 KB of page-aligned usable memory to be used during bootstrapping.
5. Initialize UBIO.
6. Load and validate the first 128 UBIO map registers to address the first 64 KB of usable memory.
7. Load contents of all four boot ROMS into memory (base + XFA00).
8. Check the cold start flag. If set, Halt. If clear, set it. This prevents cold start looping.
9. Load input arguments to be used by the Device ROM code and by VMS.
10. Select the boot ROM specified by the BOOT DEVICE switch.
11. Check for non-existent ROM; halt if no ROM.

The general registers receive the input arguments from the console subsystem.

R1	System bus address of a MASSBUS adapter (MBA0 unless otherwise specified in the BOOT command)
R2	Physical address of the UNIBUS I/O page associated with the UNIBUS adapter (UBIO unless otherwise specified in the BOOT command)
R3	Device unit number (0 unless otherwise specified in the BOOT command)

R5	Software boot control flags (0 unless otherwise specified in the BOOT command)
SP	<base address + ↑X200> of the 64 KB of good memory

The console subsystem then transfers control to the second word in the selected ROM code to begin macro instruction execution of the ROM code. The table below shows the four starting addresses.

Device ROM	Starting Address
A	FA02
B	FB02
C	FC02
D	FD02

Device ROM Code Function on Boot

The Device ROM code consists of four components:

- An ASCII description of the device type
- A control routine: gains control from microcode
- A driver subroutine
- A checksum

ASCII device description:

- Located at bytes 0-1 of the ROM
- Consists of two ASCII characters describing device type; characters are written in reverse order

Examples:

```
.ASCII /MD/     ; RK06/7 disk drive
.ASCII /BD/     ; MASSBUS disk drive
```

Control Routine, Located at bytes 2-n of the ROM

Function:	To prepare inputs for and then call the driver subroutine to bring in the bootblock from the device; when the driver subroutine returns, check for errors; if no error, prepare inputs for and call bootblock code; if error, halt
Inputs:	R1: address of a MASSBUS adapter R2: address of a UNIBUS I/O page R3: unit number of a boot device R5: software boot control flags SP: base address of usable memory + ↑X200
Implicit inputs:	UBIO map registers are marked as valid and mapped to the 64 KB of usable memory

The bootblock is at LBN 0

The bootblock will be brought into the first page of good memory

The address stored in SP is mapped to the second UBI map register

Base of good memory + C: transfer address of bootblock code

Outputs:

R0: type of boot device

0	MASSBUS Device
1	RK06/7
2	RL02
3-63	Reserved for the future
64	TU58

R1: (UNIBUS) address of the UNIBUS I/O page
(MASSBUS) address of the boot device's adapter

R2: (UNIBUS) address of the boot device's CSR
(MASSBUS) controller number of the boot device

R6: address of driver subroutine in ROM

Preserves Registers: R3, R5, R10, R11, AP, SP

Driver Subroutine:

Located after control subroutine in ROM.
Function is to read 1 block of data from device into memory. Called with JSB.

Inputs:

R1:	(UNIBUS) address of a UNIBUS I/O page (MASSBUS) address of the boot device's adapter
R2:	(UNIBUS) address of the boot device's CSR (MASSBUS) controller number of the boot device
R3:	Unit number of a boot device
R5:	Offset from beginning of 64 KB block at which data are to be written
R8:	LBN of block to be read on the boot device

4(SP) address in memory at which data are to be written

Outputs: R0: <0>@1 indicates success
0 indicates failure

Preserves: R1-R6, R10, R11,AP,SP

Devices which use the UNIBUS map require the offset from the base of good memory since the console will set up the UNIBUS map relative to that base. This is provided in R5. Devices which need the actual physical address in memory get it at 4(SP).

Checksum
Located at byte 255 (the last byte in the ROM)
Consists of the sum (discarding carries) of the first 255 bytes in the ROM.

Figure 7-3 represents a memory map of boot process.

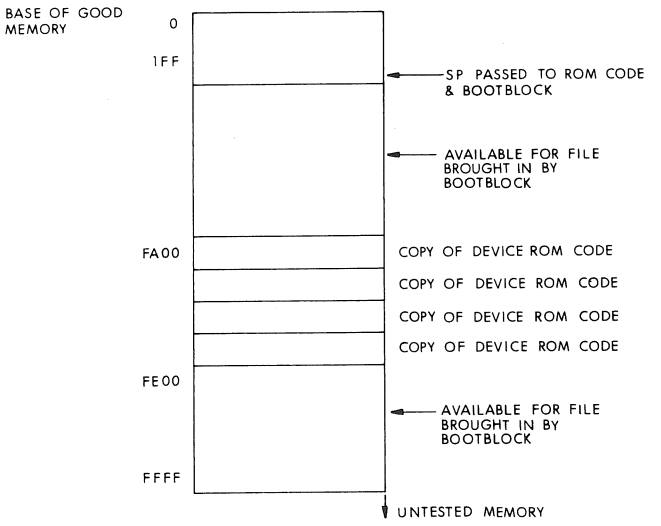


Figure 7-3 Memory Map of Boot Process

Console Subsystem Action on a Warm Start

When the console subsystem gains control of the VAX-11/750 following a power restoration, activation of the RESET switch, or CPU Halt, it examines the POWER ON ACTION switch. The console subsystem will

attempt to restart the CPU in either of two cases.

1. When the POWER ON ACTION switch is set to the RESTART/HALT position.
2. When the POWER ON ACTION switch is set to the RESTART/BOOT position.

If the POWER ON ACTION switch is set to either RESTART/HALT or RESTART/BOOT, the console subsystem searches through physical memory for a valid restart parameter block (RPB), shown as follows.

```
physical address of the RPB                              0:
physical address of the restart routine                  4:
checksum of ↑X1F longwords of restart routine            8:
Bit 0;                    !  warm start flag             C:
```

Parameter Block, First Four Longwords

A valid RPB is defined as a block of four longwords, starting on a page boundary. The first longword points to itself. The second is a pointer to the address of the restart code and must not be zero. The third longword contains the checksum (sum, throwing away carries) of the 31 longwords pointed to by the second longword. The console subsystem starts at address zero and searches all available memory for a valid RPB. If it doesn't find one it performs the second action specified by the power on switch (HALT or BOOT).

If it does find an RPB, it examines bit 0 of the fourth longword of the RPB. If this bit is 1, it will halt or boot as specified by the power-on switch.

If this bit is 0, it sets it and then:

- Loads the stack pointer (SP) with the address of the RPB plus ↑X200.
- Loads the argument pointer (AP) with a value that indicates the cause of the restart. If the restart is occuring because of power restoration or reset switch activation, the AP is loaded with a 3. If it is because of a CPU Halt, it is loaded with one of the codes specified in Table 7-1.
- Starts execution of the restart routine, whose address is located at the second longword of the RPB.


digital VAX 11/750

CHAPTER 8

VAX-11/750 CENTRAL PROCESSOR

FEATURES	BENEFITS
32-bit microprogrammed processor	Provides the user with high-performance data throughput
Memory management hardware	Allows the user to directly address up to four billion bytes of virtual address space using a smaller physical memory
Gate array technology	Reduces the physical size and power consumption of the CPU and increases system reliability
4 KB direct mapped memory cache	Significantly improves memory access time, increasing overall performance
PDP-11 compatibility mode	Gives the PDP-11 user an easy migration path to the VAX/VMS architecture
User control store (optional)	Increases throughput by allowing the user to create routines in microcode tailored to specific applications
Optional Floating Point Accelerator	Decreases instruction execution time of floating point arithmetic and some integer arithmetic operations

INTRODUCTION

The VAX-11/750 central processing unit (CPU) performs the logic and arithmetic operations requested by the computer system user. The processor is a high-performance, microprogrammed computer that executes a large set of variable-length instructions in native mode, and nonprivileged PDP-11 instructions in compatibility mode.

The CPU uses 32-bit virtual addresses, allowing access to over four gigabytes (4 GB, 2^{32}) of virtual address space. These addresses are called virtual because each address is not necessarily the actual address in physical memory. The processor's memory management hardware translates virtual addresses to physical addresses.

The processor provides sixteen 32-bit registers that can be used for temporary storage, as accumulators, index registers, and base registers. Four of these registers have special significance: the Program Counter, and three registers that are used to provide an extensive CALL facility.

The native instruction set is highly versatile and bit-efficient. It includes integer, packed decimal, character string, bit field, and floating point instructions, as well as program control and special instructions. Instructions and data are variable in length and can start on any byte boundary or, in the case of bit fields, at any arbitrary bit in memory.

The CPU can process the following kinds of data:

- Bits (up to 32)
- Bytes (8 bits)
- Words (16 bits)
- Longwords (32 bits)
- Quadwords (64 bits)
- 32-bit floating point (single precision)
- 64-bit floating point (double precision)
- Packed decimal (up to 31 digits)
- Character strings (up to 64 KB)

The remainder of this chapter is divided into two sections. The first section discusses the gate array technology used in the VAX-11/750, and the second section describes the hardware elements listed below.

- CPU control store
- Internal data paths
- 4 KB direct mapped memory cache
- 512-entry address translation buffer
- 8-byte prefetch instruction buffer
- Time-of-year clock and programmable realtime clock
- Optional floating point accelerator (FP750)
- Optional 10 KB (80 bits wide) User Control Store (UCS)

Figure 8-1 illustrates the central processing unit.

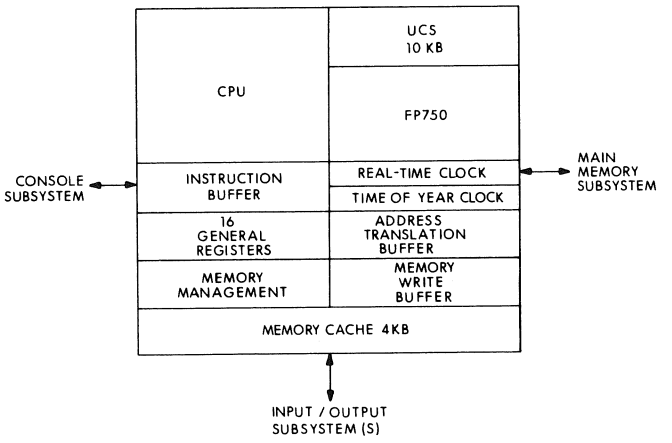


Figure 8-1 VAX-11/750 Central Processing Unit

GATE ARRAY TECHNOLOGY

The VAX-11/750 central processor uses state-of-the-art gate array technology. Three-quarters of the logic circuits in the CPU are custom LSI (Large Scale Integrated) circuits designed by DIGITAL. In the basic CPU and memory controller there are 55 LSI devices using 27 different circuit designs, with each LSI device having up to 488 logic gates. With gate array technology, the physical size of the processor and its power consumption can be reduced to approximately half of what they would be using conventional technology. In addition, this technology requires fewer components, for enhanced system reliability.

HARDWARE ELEMENTS

CPU Control Store

The control store is a programmable read-only memory containing 6K 80-bit microwords. The control store contains the program that describes the operation and sequencing of the central processing unit. It also contains the native and PDP-11 compatibility mode instruction sets. In addition, the control store contains an 80-bit buffer, enabling it to execute one microword while simultaneously fetching the next.

Internal Data Paths

The data path subsystem consists of three parallel sections used to

process addresses and data specified by the instruction set. The arithmetic section is used to perform both arithmetic and logical operations on data and addresses, and is used for fast processing of floating point instructions. The data shift and rotate section packs and unpack floating point and decimal string data. Finally, the address section calculates virtual addresses for the translation buffer.

4 KB Direct Mapped Memory Cache

Cache memory is a small, high-speed memory that maintains a copy of frequently selected portions of main memory for faster access to instructions and data. The cache memory is loaded by memory management so that there is a high probability the desired data will be in the cache. Because the processor always looks for data in the cache first, the cache offers faster system speed for the cost of a small quantity of fast memory and associated logic.

The VAX-11/750 memory cache is a 4 KB, direct mapped, write-through cache. It is used for all data coming from memory, including addresses and instructions. The write-through feature protects the integrity of memory because memory contents are updated immediately after the processor performs a write. Instead of tying up the processor while main memory is updated, VAX-11/750 processors buffer commands to avoid waiting while main memory is updated from the cache.

The memory cache (typically 90% hit rate) provides the central processor with high-speed access to main memory. The memory cache typically cuts memory access time to half what it would be with no cache feature. For increased integrity, the cache memory system carries byte parity for both data and addresses. Cache locations are allocated when data are read from memory or when a longword is written to memory. This cache also watches I/O transfers and updates itself appropriately. Thus, no operating system overhead is needed to synchronize the cache with I/O operations, and applications software does not need to be concerned with the cache.

Address Translation Buffer

The address translation buffer is a cache of frequently used physical address translations. It significantly reduces the amount of time the CPU spends on the repetitive task of dynamic address translation. The cache contains 512 virtual-to-physical page address translations which are divided into equal sections: 256 system space page translations and 256 process space page translations. Each of these sections is two-way associative and has parity on each entry for increased integrity.

8-Byte Prefetch Instruction Buffer

The 8-byte instruction buffer improves CPU performance by prefetching data in the instruction stream. The control logic fetches longword data from memory to keep the 8-byte buffer full.

Processor Clocks

The VAX-11/750 processor contains a programmable realtime clock, and a time-of-year and date clock. The interval or realtime clock permits the measurement of finely resolved variable intervals. The realtime clock is based on a crystal oscillator with accuracy of 0.01%, and resolution (update rate) of one microsecond. The time-of-year clock is used by software to perform various timekeeping functions. It provides the correct time to the system without operator intervention in the event of a power failure or other system crash. To insure continuous running, the time-of-year clock is equipped with its own battery backup.

Optional Floating Point Accelerator (FP750)

The FP750 floating point accelerator is a hardware option that operates in conjunction with the VAX-11/750 to execute the standard floating point instruction set. Floating point representation permits a greater range of number values than is possible with a 32-bit integer.

Consisting of a single module, the FP750 is easily installed and is functionally transparent to the user. Hardware and software modifications are not required.

The FP750 receives an opcode from the CPU and decodes the information into a starting microaddress. The outputs of the control store ROM control the arithmetic operations and data path logic.

The FP750 executes addition, subtraction, multiplication, and division instructions which operate on single precision (32-bit) and double precision (64-bit). It executes Extended Multiply (EMOD) and Polynomial Evaluation (POLY) instructions, and converts data between integer and floating point formats and between single and double precision floating point formats. The FP750 also executes the integer multiplication instruction.

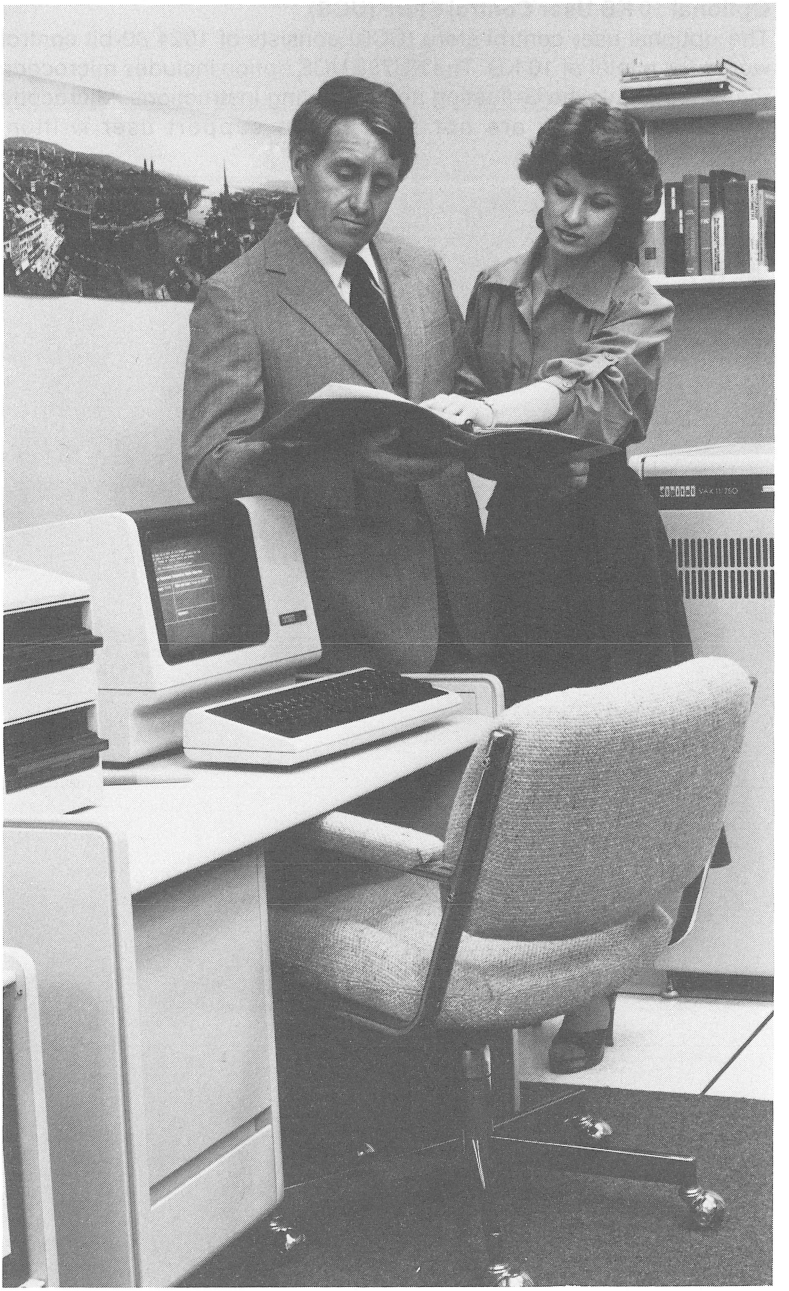
The floating point instructions performed by the FP750 are listed in Table 8-1.

Table 8-1 FPA Floating, Double, and Integer Instructions

OPCODE (HEX)	INSTRUCTION MNEMONIC	OPCODE (HEX)	INSTRUCTION MNEMONIC
40	ADDF2	62	SUBD2
41	ADDF3	63	SUBD3
42	SUBF2	64	MULD2
43	SUBF3	65	MULD3
44	MULF2	66	DIVD2
45	MULF3	67	DIVD3
46	DIVF2	68	CVTDB
47	DIVF3	69	CVTDW
48	CVTFB	6A	CVTDL
49	CVTFW	6B	CVTRDL
4A	CVTFL	6C	CVTBD
4B	CVTRFL	6D	CVTWD
4C	CVTBF	6E	CVTLD
4D	CVTWF	71	CMPD
4E	CVTLF	74	EMODD
51	CMPF	75	POLYD
54	EMODF	76	CVTDF
55	POLYF	7A	EMUL
56	CVTFD	A4	MULW2
84	MULB2	A5	MULW3
85	MULB3	C4	MULL2
60	ADDD2	C5	MULL3
61	ADDD3		

Optional 10 KB User Control Store (UCS)

The optional user control store (UCS) consists of 1024 80-bit control words for a total of 10 KB. The KU750 UCS option includes microcode that implements the G-floating and H-floating instructions. Microcode development tools are not available to support user written microcode.



CHAPTER 9

VAX-11/750 MAIN MEMORY SUBSYSTEM

FEATURES	BENEFITS
Expandable memory configuration	Allows the addition of 1 MB array cards up to a maximum of 8 MB.
Error correcting memory controller	Enhances data availability and reliability by correcting all single bit errors and detecting all double bit errors within the memory system.
	Permits write operations to any combination of bytes within an aligned longword.
Boot ROMs	Allow bootstrapping from devices including the integral TU58 cartridge tape drive.
Optional battery backup	Maintains power to preserve data for a full 8 MB memory for at least ten minutes.

INTRODUCTION

Main memory on the VAX-11/750 is implemented with array cards containing dynamic 64 Kbit array MOS memory devices interfaced to the system through an error correcting controller. The memory system is designed with a minimum capacity of 1 MB and is expandable in increments of 1 MB to a maximum capacity of 8 MB. The controller corrects all single bit errors and detects all double-bit errors that may occur within the memory system. This feature enhances both data availability and reliability. All errors are reported back to the CPU where they may be recorded for use in preventive and corrective maintenance operations. Figure 9-1 illustrates the basic memory subsystem.

For customers with the previously existing 256 KB array cards, an upgrade kit is available.

The memory controller module contains all of the logic needed to interface the array cards to the system. This includes the addressing logic, the refresh control, and the error correcting logic. The controller contains three longword registers that are accessible in I/O space.

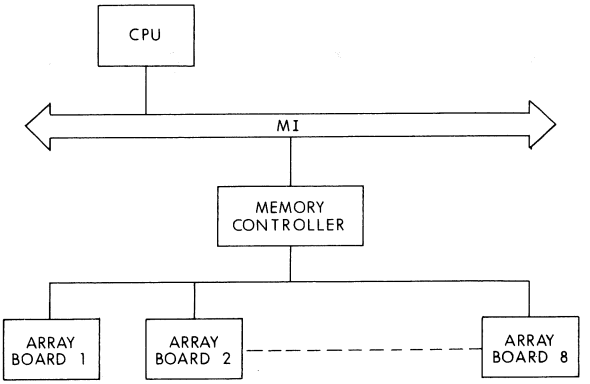


Figure 9-1 Main Memory Configuration

These registers allow a program to determine the size of the memory system, record the address of failing memory locations, isolate the error to an individual memory chip, and verify that the error correcting logic is working properly.

The memory system performance is optimized around longword read and write operations. The memory controller is designed to also allow write operations to any combination of bytes within an aligned longword; however, this operation reduces memory throughput.

BOOT ROMS

The memory controller module contains four read-only memories (ROMs) that are designed to allow bootstrapping from devices. One of the ROMs is always for the integral TU58 tape cartridge; the others are associated with various system devices used to boot the system. The ROMs are installed by Field Service to correspond to the positions of the BOOT DEVICE switch on the CPU cabinet front panel. Each ROM contains up to 256 bytes of code that are accessible in I/O space at the following physical addresses:

ROM	Physical Address (hex)
1	F20400 - F204FC
2	F20500 - F205FC
3	F20600 - F206FC
4	F20700 - F207FC

The CPU reads the data from these ROMs into main memory and executes it at bootstrap time. In addition, each ROM contains a checksum of the ROM's contents to allow verification of proper operation and associated logic.

BASIC MEMORY OPERATIONS

The memory controller is an extended length, hex height, multilayer board with both LSI and gate array devices.

The memory system provides several key features for the VAX-11/750: timing and control for the three types of memory cycles, logic for refresh operations required by MOS memory, and memory system diagnostic features (under software control). In addition, the memory system performs ECC functions and provides data for error logging, and contains bootstrap functions and initialization logic.

The VAX-11/750 physical address space contained within the memory controller is divided into two areas: physical memory addresses and I/O addresses. Figure 9-2 illustrates the combined areas.

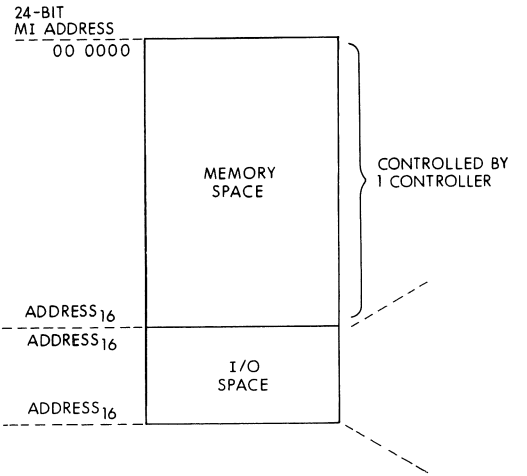


Figure 9-2 Physical Address Space

To access this address space, the system has three types of memory cycles.

Read

A READ cycle with no errors detected by ECC logic consists of four major steps. First, the device requesting the memory data presents ADDRESS and CONTROL information to the memory controller. Second, the memory controller initiates the memory cycle timing to the array cards. Third, the array cards present the memory data to the ECC logic. Finally, the memory controller puts out the data to the device requesting the READ and notifies the device that the data are available.

If, however, a correctable error is detected by the ECC logic, the corrected data are presented to the device doing the READ, and an interrupt is generated to notify the system that the error occurred. If an uncorrectable error is detected by the ECC logic, then the unmodified data are presented to the device along with an error message. If the READ reference came from an I/O adapter, then the adapter will record the error message. If the READ came from the CPU then the CPU will generate a machine check abort. When ECC detects errors during a READ cycle, the operation will be slightly slower than if no errors occurred.

Longword Write

For a LONGWORD WRITE, there are three major steps. First, the device requesting the LONGWORD WRITE presents ADDRESS and CONTROL information to the memory controller. Second, the memory controller initiates the memory cycle timing to the array cards. Third, while the array cards initiate their cycle, the memory controller generates seven check bits and presents them and the 32-bit longword to the array cards to complete the WRITE cycle.

Byte and Word Write

For these types of WRITE cycles, the sequence is the same as the first two steps of the READ cycle, with the added requirement of forming the new 32-bit data word and seven check bits into memory.

If a correctable error was detected by the ECC, the data bit will be corrected. However, if the error is in a byte or word being written, the error will be ignored. If an uncorrectable error is detected by the ECC during the READ portion of the cycle, the original longword will be rewritten into memory without any changes. This will cause reporting of the error during a subsequent READ to this location.

CONTROL AND STATUS REGISTERS

There are three control and status registers with the following format in the VAX-11/750 main memory system. CSR 0 (address $F20000_{16}$) provides information to allow the recording of both correctable and uncorrectable errors. CSR 1 ($F20004_{16}$) aids in the verification and diagnosis of the error correcting hardware. CSR 2 ($F20008_{16}$) provides memory size and configuration information.

CSR 0 Bit Allocations

The following illustration shows the organization and status of CSR 0 following a power-up sequence:

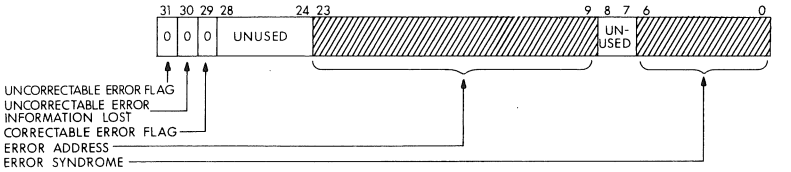


Figure 9-3 CSR 0

Bit: 31 **Name:** Uncorrectable Error Flag
Function: The occurrence of an uncorrectable error will set this bit to one. An uncorrectable error is a word which contains an even number of bits in error or an odd number of bits in error which map to an invalid syndrome. The address and syndrome of an uncorrectable error will overwrite the address and syndrome of a correctable error. This bit can only be cleared if the Uncorrectable Error Information Lost (CSR 0 <30>) is not set or if both bits are being cleared during the same write operation.

Bit: 30 **Name:** Uncorrectable Error Information Lost
Function: When this bit equals one, an uncorrectable error has occurred when the Uncorrectable Error Flag (CSR 0 <31>) was already set. The second uncorrectable error page address will not overwrite the first uncorrectable error page address. This bit is cleared by writing a one to it.

Bit: 29 **Name:** Correctable Error Flag
Function: This bit is set to one whenever a correctable (single-bit) error occurs during a READ operation. A single-bit error that occurs during the read portion of a BYTE WRITE cycle will have no effect on the flag. This bit is cleared by writing a one to it.

Bit: 28:24 **Name:** Not used

Bit: 23:9 **Name:** Page Address
Function: Identifies the page (512 bytes) on which an error occurred during a memory read cycle. The page address corresponds to the first error that occurs with the exception that the page address of an uncorrectable error will overwrite the page address of a correctable error. The page address of an uncorrectable error does not get overwritten by a more recent uncorrectable error. These bits are read-only.

Bit: 8:7 **Name:** Not used

Bit: 6:0 **Name:** Error Syndrome

Function: The error syndrome is stored in these bits. These bits are read-only.

CSR 1 Bit Allocations

The organization and status of CSR 1 following a power-up sequence is shown in the following illustration. All bits in CSR 1 are read/write.

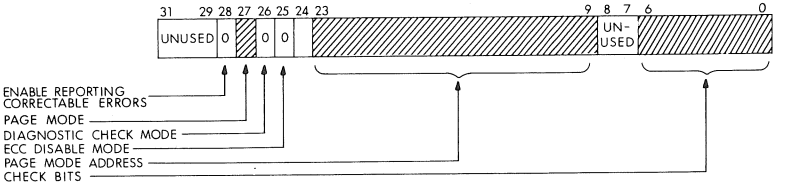


Figure 9-4 CSR 1

Bit: 31:29 **Name:** Not used

Bit: 28 **Name:** Inhibit Reporting Correctable Errors
Function: When CSR 1 <28> equals zero, correctable (single-bit) errors will request the appropriate interrupt.

When CSR 1 <28> equals one, correctable (single bit) errors will still be corrected by the ECC logic, but the status lines will report No Error. In addition, the page address of the correctable error will not be logged in CSR 0 <23:9> and the Correctable Error Flag (CSR 0 <29>) will not get set.

CSR 1 <28> has no effect on logging and reporting uncorrectable errors. This bit is set to one on power up.

Bit: 27 **Name:** Page Mode
Function: When CSR 1 <27> equals one, the ECC Disable Mode or the Diagnostic Check Mode operates on a 512-byte page whose address is contained in CSR 1 <23:9>.

The Diagnostic Check Mode must operate in the Page Mode.

The ECC Disable Mode can operate on a page or the entire memory (1 MB to 8 MB) depending on whether Page Mode is selected or not.

Bit: 26 **Name:** Diagnostic Check Mode
Function: When CSR 1 <26> equals one, the contents of CSR 1 <6:0> are substituted for the check bits that come from memory during a read operation.

The Diagnostic Check Mode is constrained to operate on a single page whose address is stored in CSR 1 <23:9>. Therefore the Page

Mode (CSR 1 <27>) must also be selected.

While operating in the Diagnostic Check Mode, read errors that occur in other pages of memory will not be logged into CSR 0.

Correct check bits are always put into memory during a write cycle.

The ECC Disable Mode and Diagnostic Check Mode cannot be selected at the same time.

Bit: 25 **Name:** ECC Disable Mode
Function: When CSR 1 <25> equals one, no detection of uncorrectable errors will occur, no correction of single bit errors will take place, no error logging in CSR 0 will occur and No Error will be reported on the status lines.

In the ECC Disable Mode (CSR 1 <25> equals one), a read operation stores the seven check bits read from the array in CSR <6:0> and during a write operation, the generated check bits that are written into the array are also stored in CSR 1 <6:0>.

The ECC can be disabled for a 512-byte page or for the entire memory (1 MB to 8 MB) depending on whether the Page Mode (CSR 1 <27>) is selected or not.

Correct check bits are always put into memory during a write cycle. The ECC Disable Mode and Diagnostic Check Mode cannot be selected at the same time.

Bit: 24 **Name:** Not used

Bit: 23:9 **Name:** Page Mode Address
Function: If the memory system is in the Page Mode (CSR 1 <27> equals one), CSR 1 <23:9> will contain the address of the 512-byte page to which the Diagnostic Check Mode or the ECC Disable Mode will be applicable.

Bit: 8:7 **Name:** Not used

Bit: 6:0 **Name:** Check Bits
Function: In Diagnostic Check Mode, the contents of CSR 1 <6:0> are substituted for the check bits that come from the memory during a read operation.

In ECC Disable Mode, a read operation takes the seven check bits read from the array and stores them in CSR 1 <6:0>.

CSR 2 Bit Allocations
This register is read-only.

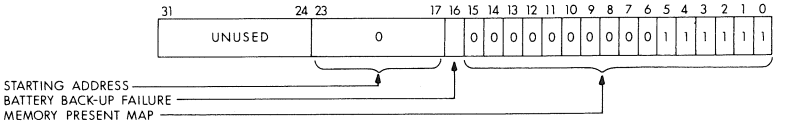


Figure 9-5 CSR 2

Bit: 31:24 **Name:** Not used

Bit: 23:17 **Name:** Starting Address
Function: The starting address for the memory systems will be contained in these bits. The starting address is defined by jumper lines within the system and normally is set to zero.

Bit: 16 **Name:** Battery Backup Failure
Function: This bit is set on a power-up sequence if the battery back-up power has been exhausted.

The bit is cleared on a LONGWORD WRITE to any location in memory.

Bit: 15:0 **Name:** Memory Present Map
Function: These bits represent the amount of memory present in the system and the locations in the memory backplane where the array boards are inserted. There are eight possible locations for the array boards and two possible array board configurations. The array boards can be 1 MB or .25 MB. The map is divided into eight pairs of two bits.

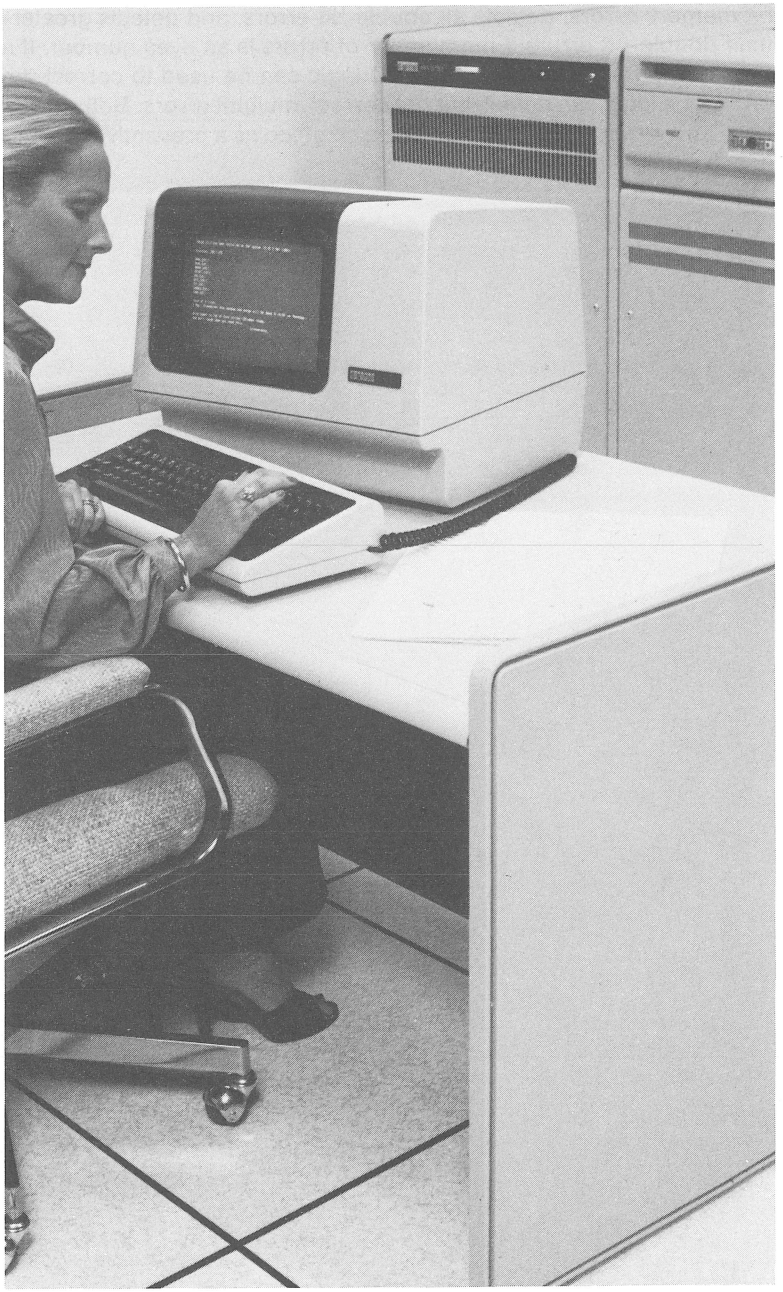
BATTERY BACKUP

The memory system has an optional battery backup unit designed to maintain the contents of memory during power failures. This system will hold data for the full 8 MB of memory for at least ten minutes. When power is restored, the memory controller checks to see if the contents of memory have been preserved by the battery unit. If they have not, the controller will write the contents of the entire memory system with zeros. On systems without the battery backup option, the controller always writes the memory system with zeros following a power-up sequence. The status of memory following a power-up sequence is noted in a control register so that the CPU can determine whether or not the contents of memory have been preserved throughout a power failure.

ERROR CHECKING AND CORRECTION

The ECC scheme used in the memory subsystem corrects all single-

bit memory errors, detects all double-bit errors, and detects greater-than double-bit errors if the number of errors is an even number. If a single bit error is detected, the ECC logic can be used to correct the error. The logic can detect, but not correct, multibit errors. Both detections and corrections are noted in the error log as a preventive maintenance aid.



CHAPTER 10

VAX-11/750 UNIBUS SUBSYSTEM

FEATURES	BENEFITS
Direct memory access (DMA) data transfers	Eliminates processor intervention for high data throughput
Peer communication between UNIBUS devices	Allows direct data transfer between UNIBUS devices without CPU involvement
Total compatibility with the PDP-11 UNIBUS data path	Gives the PDP-11 user an easy migration path to the VAX-11/750 processor
Direct vectored hardware interrupts to servicing routines	The CPU avoids time-consuming polling tasks when servicing interrupt requests
UNIBUS device registers are addressed like memory locations	Simplifies I/O programming
Supports a wide range of standard DIGITAL peripherals	Offers the user flexibility in peripheral selection to meet specific requirements
Three buffered data paths	Increases data throughput by buffering data traffic to memory
Optional second UNIBUS	Greater bandwidth available for UNIBUS devices

INTRODUCTION

The UNIBUS subsystem connects most medium and low speed peripheral devices to the VAX-11/750 system. An asynchronous, bidirectional bus, the UNIBUS is used with all devices other than high-speed disk drives and magnetic tape transports. The UNIBUS lets the user select from a range of existing peripherals (those supported by VAX/VMS and diagnostics) and also provides easy connection for customer-designed special devices. Although the UNIBUS was originally designed for implementations of the PDP-11 architecture, its features and capabilities have been expanded by the VAX architecture. Thus, existing UNIBUS peripheral devices can be used with the new VAX family architecture without hardware modification. (UNIBUS addresses in this chapter are listed in both octal and hexadecimal. The PDP-11 uses octal while the VAX family uses hexadecimal.)

The integral UNIBUS adapter (part of the CPU) connects the UNIBUS to the system and performs priority arbitration among UNIBUS devices. Furthermore, the UNIBUS adapter lets the processor access UNIBUS peripheral device registers.

In addition to the integral UNIBUS adapter, a second UNIBUS adapter may be added. One of the three general-purpose I/O adapter slots in the CPU may be used for a second UNIBUS adapter.

Figure 10-1 illustrates the UNIBUS subsystem configuration.

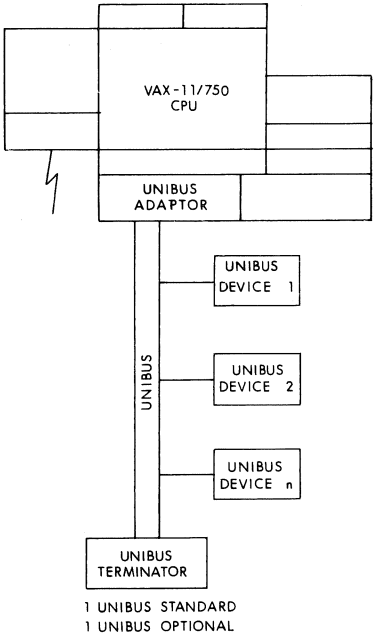


Figure 10-1 VAX-11/750 UNIBUS Configuration

VAX-11/750 UNIBUS SUMMARY

The UNIBUS is a communication path that links I/O devices to the UNIBUS adapter. Device-related addresses, data, and control information are passed along the 56 signal lines of the UNIBUS. The UNIBUS adapter handles all communications between the UNIBUS and the system, and fields device-generated interrupts.

Conceptually, the UNIBUS is designed around memory elements with ascending addresses starting at UNIBUS address zero, while registers storing I/O data or individual peripheral device status information have addresses in the highest 8 KB of addressing space ($3E000_{16}$ to $3FFFE_{16}$, or 760000_{8} to 777776_{8}).

The UNIBUS consists of 56 signal lines, to which UNIBUS peripheral devices are connected. Figure 10-2 illustrates the signal line configuration. 51 of these lines are parallel and 5 are serial; 42 lines are bidirectional and 14 are unidirectional.

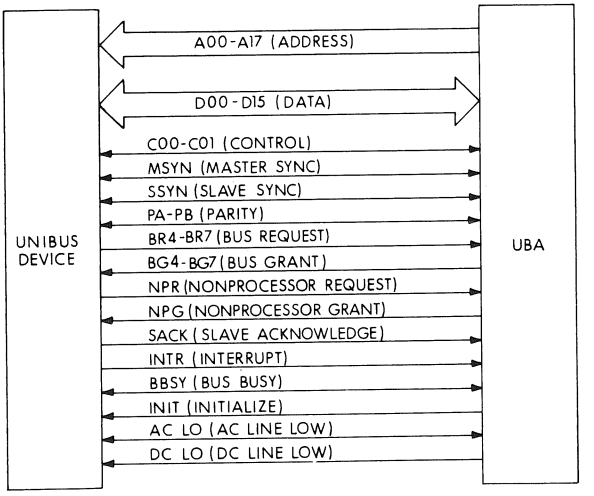


Figure 10-2 UNIBUS

Communication between any two devices on the bus is in a master/slave relationship. During any bus operation, one device, the bus master, controls the bus when communicating with another device on the bus, called the slave. For example, the processor, as master, can fetch from a peripheral device, as slave; or the disk, as master, can transfer data to memory, as slave. Master/slave relationships are dynamic; the processor, for example, may pass bus control to a disk, then the disk may become master and communicate with slave memory. On the VAX-11/750, the main memory that the processor deals with for instructions and data is not actually on the UNIBUS, but is attached to the processor. The UNIBUS adapter causes the VAX-11/750 memory to look like a slave to UNIBUS devices.

When two or more devices try to obtain control of the bus at once, priority circuits decide among them. Device priority levels are fixed at system installation. There are four priority levels among UNIBUS devices. A unit with a high priority level always takes precedence over one with a lower priority level. In the case of units with equal priority levels, the one closest electrically to the processor on the bus takes

precedence over those further away. The processor priority is raised and lowered according to the interrupt priority level (IPL).

Suppose the processor has control of the bus when three devices, all of higher priority than the processor, request bus control. If the requesting devices are of different priorities, the processor will grant use of the bus to the one with the highest priority. If they are all of the same priority, all three signals come to the processor along the same bus line, so that it sees only one request signal. Its reply, granting control of the bus, travels down the bus to the nearest requesting device, passing through any intervening non-requesting devices. The requesting device takes control of the bus, executes a single bus cycle and relinquishes the bus. Then the request grant sequence occurs again, this time going to the second device down the line, which has been waiting its turn. When all higher-priority requests have been granted, control of the bus returns to the processor.

The processor usually has the lowest priority because in general it can stop whatever it is doing without serious consequences. Peripheral devices may be involved with some kind of mechanical motion, or may be connected to a realtime process, either of which requires immediate attention to a request to avoid data loss.

Priority arbitration takes place asynchronously in parallel with the data transfer.

Bus Communication

Communication is interlocked, so that each control signal issued by the master must be acknowledged by a response from the slave to complete the transfer. This simplifies the device interface because timing is no longer critical.

Using the Bus

A device uses the bus if it needs to:

- Interrupt the processor. As a result, the processor stops what it is doing, enters a device service routine, and services the device.
- Transfer a word or byte of data to or from memory without involving the processor. Such functions—called NPRs or non-processor request transfers—are performed by direct memory access devices such as disks or tape units.

Whenever two devices communicate, it is called a bus cycle. Only one word or byte can be transferred per bus cycle.

Bus Control

There are two ways of requesting bus control: non-processor requests (NPRs) or bus requests (BRs).

An NPR is issued when a device wishes to perform a data transaction. An NPR does not use the CPU; therefore, DMA activity can occur while the CPU continues to execute instructions.

A BR is issued when a device needs to interrupt the CPU for service. A device can interrupt the CPU only if it has gained control via a BR.

Interrupts

Interrupt handling is automatic in the VAX-11/750. No device polling is required to determine which service routine to execute. When interrupting, the device supplies a vector which directs the CPU to a memory location which contains the starting address of an interrupt service routine. When servicing the interrupt, the processor automatically raises its priority to the level of the interrupting device.

Priority Control

The UNIBUS priority system determines which device obtains the bus. Each UNIBUS device is assigned a specific location in the priority structure. Priority arbitration logic determines which device obtains the bus according to its position in the priority structure. The priority structure is two-dimensional, i.e., there are vertical priority levels and horizontal priorities at each level. Figure 10-3 illustrates UNIBUS priority control.

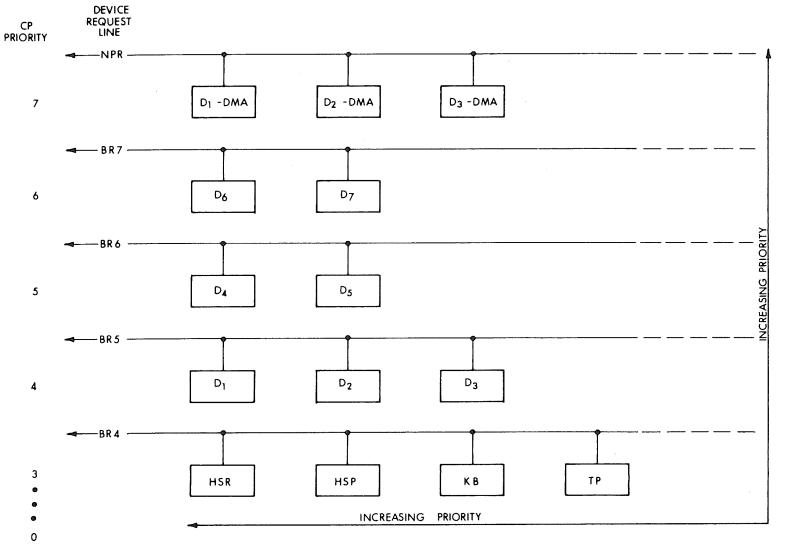


Figure 10-3 Priority Control

There are five UNIBUS vertical priority levels—NPR, and BR7, BR6, BR5, and BR4—which correspond to VAX interrupt priority levels (IPLs) 14_{16} through 17_{16}. To accommodate several peripheral devices, it may be necessary to connect more than one device to a single level. When a number of devices are connected to the same level, the device electrically closest to the CPU has the highest priority.

Priority Assignments

When assigning priorities to a device, three factors must be considered: operating speed, ease of data recovery, and service requirements.

Data from some devices are available for only a short time period. Therefore, highest priorities are usually assigned to these devices for efficient data transfers. Lower priorities are assigned to devices whose data are available for longer periods of time, and to devices which have automatic data recovery features. For example, a disk or magnetic tape device would be assigned a higher priority than a line printer or paper tape device. These priorities are assigned at installation time.

CPU Priority Level

In addition to device priority levels, the CPU has a programmable priority. The CPU can be set to any one of 32 priority levels. Levels 14_{16} through 17_{16} correspond to UNIBUS levels BR4 through BR7.

Data Transactions

There are four types of UNIBUS data transactions:

- DATO—a data *word* is transferred from the master to the slave.
- DATOB—a data *byte* is transferred from the master to the slave.
- DATI—a data word *or* byte is transferred from the slave to the master.
- DATIP—used to allow a read from slave/write from slave sequence to occur as a single bus cycle without any other UNIBUS activity intervening. This feature allows devices to synchronize their activities with the processor. DATIP must be followed by DATO or DATOB to the same location.

VAX-11/750 UNIBUS ADAPTER

The VAX-11/750 UNIBUS adapter serves three purposes. It allows the processor to access registers on the UNIBUS; it allows devices on the UNIBUS to perform DMA transfers to the VAX-11/750 memory; and it allows UNIBUS devices to interrupt the processor.

There are three characteristics of the VAX architecture and the VAX-11/750 memory system that require more than a straight-through connection from the UNIBUS to the system. First, addresses that are contiguous in the virtual address space may be noncontiguous in the physical address space on 512-byte boundaries. Since all UNIBUS non-processor request (NPR) devices broadcast sequential addresses, these device addresses must be broken up into noncontiguous 512-byte blocks by the UNIBUS adapter.

Second, the VAX architecture imposes no restrictions on the alignment of data in memory. There are restrictions placed on UNIBUS word transfer NPR devices, however, in that data words are only transferred on even addresses. The VAX-11/750 UNIBUS adapter provides an offset mechanism that allows the transfer to be effectively shifted by one byte to accommodate requests for I/O buffers on odd byte addresses. Third, because UNIBUS devices can only address a maximum of 256 KB, the UNIBUS adapter allows these addresses to be expanded. This permits the devices to address the VAX-11/750's full 16 MB of physical address space.

The adapter also provides three buffered data paths (similar to small caches) which allow UNIBUS devices to take advantage of the memory's four-byte accesses. This efficiency results in higher UNIBUS throughput and enhanced overall system operation.

PROCESSOR ACCESS TO UNIBUS

Any processor access with a physical address in the range of $FC0000_{16}$ through $FFFFFF_{16}$ (for the first UNIBUS) or $F80000_{16}$ through $FBFFFF_{16}$ (for the optional second UNIBUS) will map directly to the UNIBUS in the range 0 through 777777_8. Any VAX-11/750 internal device (other than the processor) that references that address range will be ignored by the UNIBUS adapter. The device will receive a nonexistent memory confirmation.

Processor Operations

The VAX-11/750 UNIBUS adapter responds only to aligned word or byte transactions. Any other type of access will yield UNPREDICTABLE results. The different types of operations mapped into UNIBUS operations are: processor reads, processor reads with modify intent, processor read lock, write, and write unlock. Processor reads become UNIBUS Data In (UNIBUS read or DATI operations); processor reads with modify intent and processor read lock become Data In Pause (DATIP); and write and write unlock become UNIBUS Data Out operations (UNIBUS write, DATO or DATOB).

UNIBUS Responses

A processor read to the UNIBUS that causes a no-response timeout

will result in a machine check abort with a non-existent memory (NXM) bus error indicated. A processor write that causes such a timeout to occur will generate a write bus error interrupt request. If a UNIBUS device asserts PB in response to a processor access to it, it will cause a machine check abort, indicating an uncorrectable bus error.

UNIBUS INITIATED DATA TRANSFERS

Overview

As mentioned earlier in this chapter, the UNIBUS adapter performs a number of functions to make the UNIBUS architecture compatible with the VAX architecture. A key element in this matching operation is the UNIBUS Map. This map translates UNIBUS device-generated addresses into physical memory addresses. It also identifies UNIBUS transfers so that other features, such as buffered data paths and odd byte addressing, can be used appropriately.

The UNIBUS has 18 address lines, creating an address space of 256 KB. This is conceptually divided into two regions: the bottom 248 KB used to address UNIBUS memory, and the top 8 KB used to address UNIBUS peripheral device control and status registers (CSRs). A UNIBUS NPR device typically does a transfer to memory by doing a series of transactions placing addresses in the lower range on the bus. In the VAX-11/750, these are not the actual memory addresses, but serve as pointers to the UNIBUS Map, which in turn provides the actual physical memory address. If more than one device is doing transfers at the same time, each one should be set up to transfer into a different range within the 248 KB address space.

The UNIBUS address space is divided into 512 pages of 512 bytes each. When an address arrives at the UNIBUS adapter, the page number (the upper 9 bits of the 18-bit UNIBUS address) is taken as an index into the map, with the map then providing the actual Page Frame Number (PFN) of the physical memory address. This is concatenated with the address of the word or byte within the page from the UNIBUS address (the lower 9 bits of the 18-bit UNIBUS address) to create the final memory address. The map also provides a bit that specifies whether this transaction should have its address incremented by one, allowing a device to address memory at an odd address. In addition, the map specifies which of the four data paths the transaction is to use—the direct data path or one of the three buffered data paths.

In using a direct data path, the UNIBUS adapter insures that every UNIBUS transaction results directly in a memory transaction and that the UNIBUS is kept busy until the memory transaction is completed. However, the direct data path is not as efficient as the buffered data path; this is because memory can do four-byte transfers whereas the UNIBUS is limited to two-byte transfers.

Benefits of Buffered Data Paths

The buffered data paths allow UNIBUS devices to optimize use of memory. Because most NPR devices do sequential address transfers, every pair of two-byte transactions can be merged into a single long-word transfer. Therefore, the buffered data paths use only half the number of main memory transactions that the direct data path uses. The buffered data paths also increase performance for a large class of devices which access memory sequentially, but not exclusively so. Examples of such devices are graphics and intelligent communications devices. Furthermore, because requests are fulfilled from the buffer faster than a memory access can occur, the buffered data paths improve UNIBUS access time.

The buffered data path consists of an address register and a four-byte data buffer that together act as a cache. Only when a request cannot be met from this cache is a memory cycle required. If more than one NPR device is doing sequentially addressed transfers, the address stream that arrives at the UNIBUS adapter will not be sequential. This is why three buffered data paths are provided. The UNIBUS Map directs different transfers to different data paths so that each data path sees only the activity of a single device and receives a sequential address stream.

Under VAX/VMS, these features are transparent to I/O drivers. VAX/VMS supplies standard system routines to allocate and release buffered data paths and blocks of map registers. Standard routines also set up the allocated map registers for a transfer and convert the buffer address to a UNIBUS space address for loading into the device buffer address register. For customer-developed operating systems and I/O drivers, the user would need to program these functions to manage the buffered data paths and map registers. A set of addresses is provided in I/O space for loading the map and controlling the buffered data paths.

UNIBUS Adapter Operating Detail

UNIBUS address bits <17:9> are used to enter a 512-byte by 19-bit wide memory location. The data coming out of that memory determine how the transaction will be handled. The map data field is divided into four sections: Page Frame Number (PFN), Data Path Number, Offset Bit, and Valid Bit. The format of the map data fields is shown in Figure 10-4.

On UNIBUS initiated transactions that cause a memory read or write cycle to occur, the map PFN is concatenated with UNIBUS address bits <8:0> to form a 24-bit physical address. Figure 10-5 shows this address translation process.

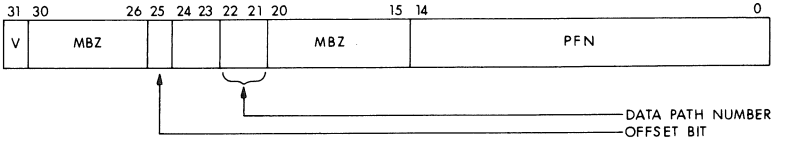


Figure 10-4 Map Data Field

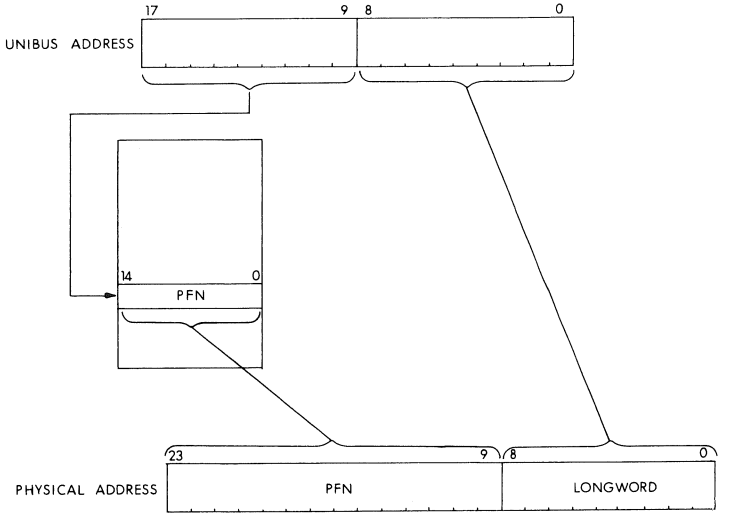


Figure 10-5 UNIBUS to Physical Address Translation

The Data Path Number is a two-bit field used to select among four data paths. Data paths one, two, and three are the buffered data paths, and data path zero is the direct data path.

If the Offset Bit is a one, it causes the transaction to behave as if the UNIBUS address supplied by the DMA device were incremented by one. This allows devices that only produce even byte addresses to access buffers on odd byte boundaries. A transaction that causes a word to cross page boundaries because the Offset Bit is set must have the Data Path Number, Offset Bit, and Valid Bit identical in both map entries. Any differences will yield UNPREDICTABLE results. If this is a DATI(P) or a DATO and the two bytes of UNIBUS data fall across a longword boundary, two memory cycles will occur.

If the Valid Bit is a one, the VAX-11/750 integral UNIBUS adapter processes the transaction as described above. When it is zero, the

integral UNIBUS adapter ignores UNIBUS requests. The Valid Bit must be set to zero for map entries that correspond to sections of UNIBUS address space in which there are slaves expected to respond to transactions originating on the UNIBUS. Transactions from the CPU that cause a UNIBUS transaction to occur are always ignored by the integral UNIBUS adapter and can never wrap back through the UNIBUS adapter to memory.

Map Access — The map is accessible for both reading and writing. Each entry uses a longword address from $F30800_{16}$ to $F30FFC_{16}$ (for the first UNIBUS) or $F32800_{16}$ to F32FFC (for the optional second UNIBUS). The logic that writes the map assumes that any write to the map addresses is a longword write. Any write other than a longword will make the contents of the map at the written location UNPREDICTABLE.

Direct Data Path — When the Data Path Number in the map is zero, the transaction uses the direct data path (DDP). This means that SSYN is not issued by the UNIBUS adapter until the corresponding memory transaction is completed. Listed below are the types of memory functions that are initiated as a result of a UNIBUS cycle:

DATI	Read
DATIP	Read Lock
DATO(B)	Write or Write Unlock

If a DATO(B) follows a DATIP, then a write unlock will be issued; otherwise an ordinary write will occur.

Offset — If the offset in the map is set, then the transaction will be treated as if the UNIBUS address were incremented by one. If this is a DATI or a DATO and if the address crosses a longword boundary, two cycles will occur. A device must not do a DATIP through the direct data path with the Offset Bit enabled.

Buffered Data Paths — When the Data Path Number in the map is one, two, or three, then a buffered data path has been selected. Each of the three BDPs consists of 4 bytes of data storage, 16 bits of address storage, 5 flag bits, and logic to make the BDP operate. These registers and flags are not accessible to software, but descriptions are included to permit precise definition of the BDP operation. The general intent of the BDP is that when UNIBUS transactions are occuring with sequential addresses (either ascending or descending), only one transfer is needed for every two UNIBUS transfers. When non-sequential transactions occur, then the correct data are provided, but no bandwidth saving occurs.

Data Buffer — Each buffered data path consists of a data storage

buffer of four bytes. This storage buffer can be loaded from the UNIBUS or memory, and its contents can be output to either the UNIBUS or to memory. Data can be loaded into the buffer one or two bytes at a time from the UNIBUS, but is always loaded four bytes at a time from memory.

Address Register — Each buffered data path has a 16-bit address register that can be loaded from UNIBUS addresses <17:2> (or addresses <17:2>+1 if offset is on). Circuitry compares the stored address with the address on the UNIBUS to see if there is a match. The address held in the register is the UNIBUS longword corresponding to the data in the data buffer.

Flags — There are five flags that keep track of the data in the data buffer, named CD and BF3 through BF0. If CD = 1, then the buffer has four bytes of data from memory and BF3 through BF0 are always zero. If CD = 0, then BF3 through BF0 indicate which bytes in the data buffer have valid UNIBUS data. If they are all zero, the buffer is considered empty.

Buffered Data Path Behavior

- DATI(P). The buffered data paths treat DATI and DATIP identically. The behavior of the UNIBUS adapter is primarily determined by the contents of the data and address buffers. If the buffer is empty or contains memory data but no address match, the UNIBUS adapter performs five operations in the following sequence: 1) does a memory read; 2) puts the read data in the buffer and on the UNIBUS; 3) puts the UNIBUS address in the address register; 4) sets the flags to indicate memory data in the buffer; and 5) sends the data to the requesting UNIBUS device.

  If there are UNIBUS data in the buffer, the UNIBUS adapter first performs a memory write, using the stored data as data, the stored address as an address, and the byte flags as the byte mask. Operation then continues with the sequence described above, beginning with step 1.

  If there are memory data in the buffer and an address match, the UNIBUS adapter puts the data on the UNIBUS and issues SSYN.
- DATI(P) with byte offset. If the map specifies byte offset, the behavior depends on whether the transaction crosses a longword boundary. If it does not, then the response is identical to that described above for DATI(P). If the transaction does cross a longword boundary, then the UNIBUS adapter acts as if two sequential transfers occurred—the first at the given UNIBUS address, the second at the address incremented by two. The two memory reads are pieced

together to form the UNIBUS data word. The data buffer and address register hold the information from the second read at the end of the transaction.

- DATO(B). For a DATO(B) transaction, the contents of the buffer determine the behavior of the UNIBUS adapter. If the buffer is empty or has memory data, the UNIBUS adapter performs four functions: 1) puts the UNIBUS data in the data buffer; 2) puts the UNIBUS address in the address register; 3) sets the flags to indicate UNIBUS data in the buffer; and 4) indicates to the UNIBUS device that the transaction is completed.

  If the buffer has UNIBUS data with an address match, behavior depends on whether the UNIBUS data, combined with the data in the buffer, form a full longword. If the data do not constitute a longword, the UNIBUS adapter performs the same four operations as it does when the buffer is empty or has memory data (steps 1 through 4 above). If the data form a longword, the UNIBUS adapter will do a memory write and clear the flags to show that the buffer is empty.

  If the buffer has UNIBUS data with no address match, the UNIBUS adapter will do a memory write and set the flags to show that the buffer is empty. Then the UNIBUS adapter will perform the same operations described above for DATO(B) with the buffer empty.
- DATO with byte offset. If this transaction crosses a longword boundary, the UNIBUS adapter treats it as two one-byte writes. If it does not cross a boundary, the UNIBUS adapter handles it as a DATO(B) as described above.
- DATOB with byte offset. If this transaction does not cross a longword boundary, the UNIBUS adapter treats it like a DATO(B) as described above, incrementing the address by one. If a longword boundary is crossed, the UNIBUS adapter treats it as if it were a DATOB in the next longword. In the latter case, address match is forced to no match.

Note: The behavior described for DATI(P) allows a device using a buffered data path to perform any sequence of transfers in either direction with any address sequence and still have the data end up in the desired location. A device that does repeated transfers within the same longword, however, will not cause any memory cycles. Thus, one may not use a buffered data path with a device that repeatedly reads one location in memory as a flag, waiting for the CPU to change it. In this case, the device would never see the change in memory since its reads will all be filled from the buffer.

Control and Status Registers — Proper transfers through BDPs require some intervention on the part of system software, aided by the implementation of control and status registers. The use of BDPs is not totally transparent. When a device has finished a series of DATO(B) transfers to memory, it is possible that some data will remain in the buffer if the transfer did not end on an even longword boundary. It is necessary for the software to initiate action to write this data to memory. When a device has finished a transaction that involves DATI(P)s, data are left in the buffer with a corresponding UNIBUS address in the address register. Should the contents of the map be changed at the location corresponding to the address in the address register, there will no longer be the correct association between address and data in the buffer. It is therefore necessary to clear the buffer following these transactions. A set of registers enable full control over these BDP transactions. The UNIBUS adapter is assigned a block of 8 KB of address space to map control and status registers. The bit assignments for these registers are described below and Figure 10-6 illustrates the format of a BDP control and status register.

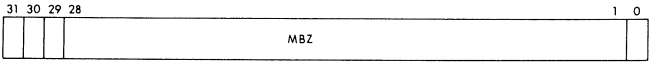


Figure 10-6 Buffered Data Path CSR

Bit: <31> **Name:** Error
Function: This bit on read is the OR of bits <30> and <29>. Writing to this bit has no effect.

Bit: <30> **Name:** NXM, Nonexistent Memory
Function: This bit is set when NXM status is received from memory. There is no response on the UNIBUS, and all future UNIBUS transactions through this BDP are ignored until this bit is cleared. This bit is cleared by writing a one to it.

Bit: <29> **Name:** UCE, Uncorrectable Error
Function: This bit is set when uncorrectable error status is received from memory. PB is asserted with the data that are passed back to the UNIBUS device on the first read from that location. It is not asserted on subsequent reads from this BDP. The bit is cleared by writing a one.

Bit: <28:1>
Name: Unused

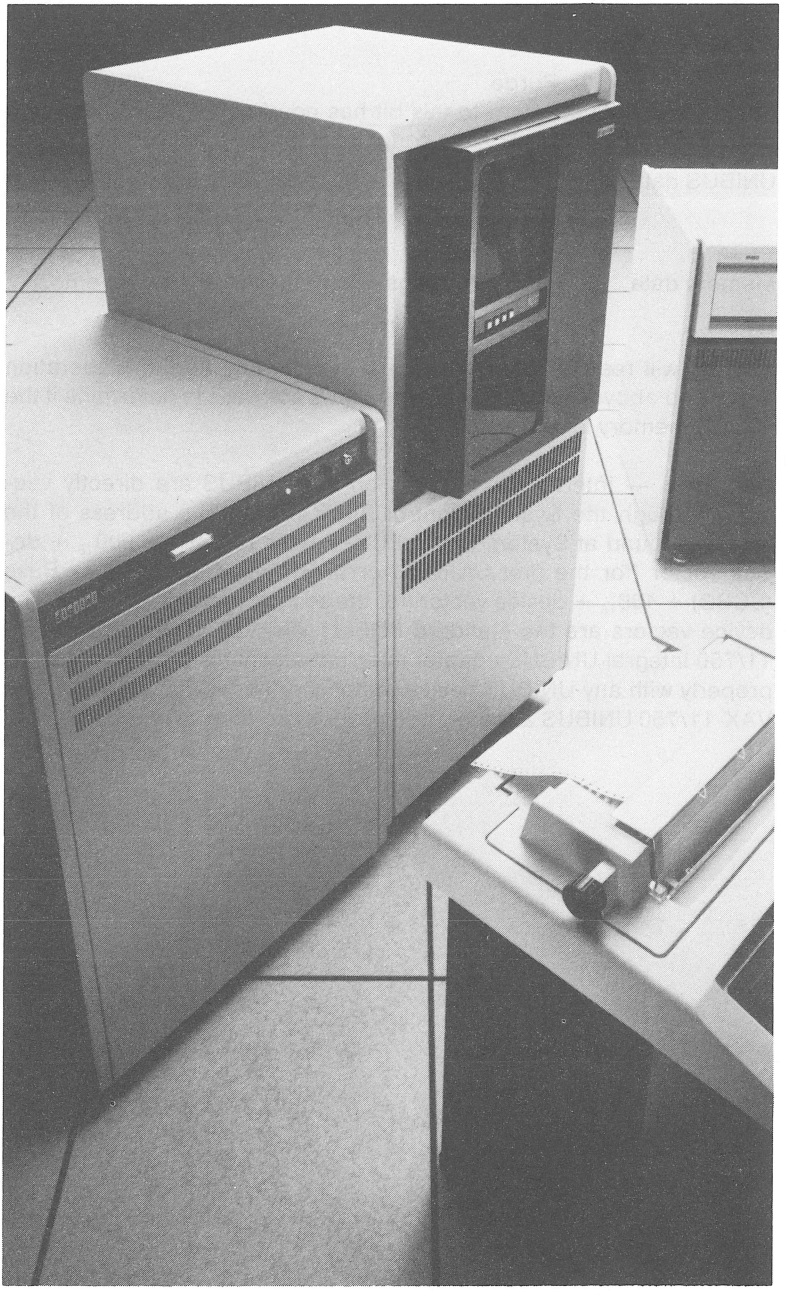
Function: These bits yield UNPREDICTABLE values when read and are ignored when written.

Bit: <0> **Name:** Purge
Function: Writing a zero to this bit has no effect. Writing a one to it produces a result based on the contents of the buffer:

UNIBUS data	The data are written to memory and the flags are set to mark the empty buffer empty
Memory data	The flags are set to mark the buffer empty
Empty	No action occurs

This bit will read as one following a write of one until the operation described above is completed. This allows software to determine if the write to memory is completed.

Interrupts — Interrupting devices on the UNIBUS are directly vectored through the System Control Block (SCB). The address of the vector is found at System Control Block Base (SCBB) + 200_{16} + device vector (for the first UNIBUS) or at System Control Block Base (SCBB) + 400_{16} + device vector (for the second optional UNIBUS. The device vectors are the standard PDP-11 UNIBUS vectors. The VAX-11/750 integral UNIBUS adapter interrupt mechanism will not operate properly with any UNIBUS device vector at or above 200_{16} (1000_{8}). The VAX-11/750 UNIBUS adapter itself never generates an interrupt.



CHAPTER 11

VAX-11/750 MASSBUS SUBSYSTEM

FEATURES	BENEFITS
Direct memory access (DMA) data transfers	Eliminate processor intervention for high data throughput
32-byte silo data buffer for each MASSBUS	Permits transfers at rates up to 2 MB/second (5 MB/second with three MASSBUSes).
Built-in diagnostic features	Allow on-line diagnosis of the MASSBUS and MASSBUS drives
MASSBUS device registers are addressed like memory locations	Simplifies I/O programming

INTRODUCTION

This chapter describes the hardware features of the MASSBUS subsystem. It outlines the kinds of CPU commands accepted by the MASSBUS adapter and describes in detail the MASSBUS adapter operation and the configuration of the various registers which control MASSBUS I/O operations. This INTRODUCTION and the section on MASSBUS ADAPTER OPERATION contain little technical detail and will be of interest to most readers. However, the section beginning with MBA REGISTERS will be of interest primarily to system designers not using VAX/VMS, who are writing their own operating systems or I/O device drivers. People writing their own device drivers under VAX/VMS should refer to the manual "How to Write a VMS Device Driver."

The MASSBUS adapter (MBA) is the hardware interface between the system and the high-speed MASSBUS storage devices on the VAX processors. The MASSBUS consists of two 16-bit wide communication paths linking the MASSBUS adapter to the mass storage device drives. One path, the data bus, is used for data during data transfer operations. The other bus, the control bus, is used for accessing drive registers. The MBA will handle a MASSBUS drive with a maximum burst data transfer speed of 900 ns per 16 bits and a bandwidth of 2 MB per second. Figure 11-1 illustrates the MASSBUS subsystem.

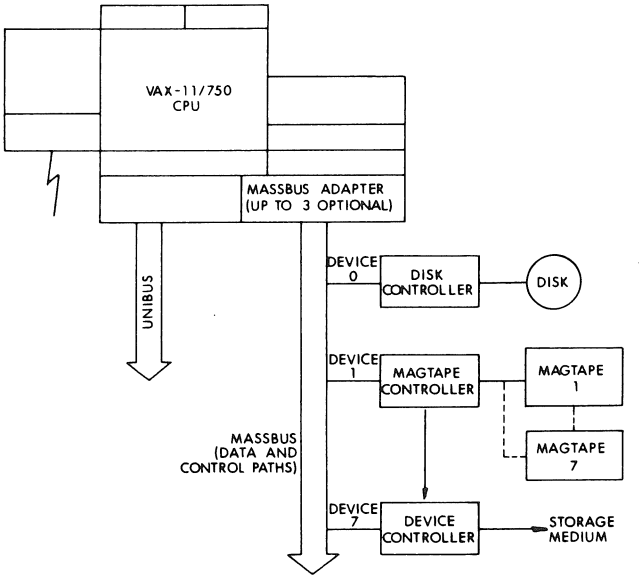


Figure 11-1 MASSBUS Subsystem

The MBA accepts and executes commands from the CPU, and reports the necessary status changes and fault conditions to the CPU. The MBA will accept the following commands from the CPU (see Appendix E of the VAX Architecture Handbook for command details).

- Read
- Read Lock (treated as Read)
- Read with Modify Intent (treated as Read)
- Write Unlock (treated as Write)

The MBA will send the following commands out to memory:

- Read
- Write
- Write Vector

The MBA will start a MASSBUS operation to transfer register data or a block of data to or from a MASSBUS device. 256 32-bit mapping registers store the page frame numbers of a block data transfer. A 32-

byte deep data buffer is used to smooth the data transfer between memory and MASSBUS devices. The 32-bit memory data will be sent in two 16-bit words to the MASSBUS device.

Special diagnostic features are built into the hardware to allow online diagnosis of MBA and MASSBUS devices, and can be exercised through diagnostic features with no device on the MASSBUS.

The VAX-11/750 MBA handles these functions:

- Mapping addresses from virtual (program) to physical memory
- Data buffering for transfers between main memory and the MASSBUS
- Transferring interrupts from MASSBUS devices to the system

The VAX-11/750 can support up to three MASSBUS adapters, (two if the optional UNIBUS is used) with each adapter supporting up to eight device controllers. A MASSBUS adapter supports any combination of mass storage devices, and is linked to devices by device *controllers*. There are controllers for different types of peripheral devices, with some controllers servicing several devices, while others support one device. For instance, each magnetic tape controller can support up to eight tape drives, but each disk controller supports a single disk drive. Regardless of the type of controller, only one controller can transfer data at any one given time. The data transfer rate depends on the particular mass storage device being accessed.

MASSBUS ADAPTER OPERATION

The MASSBUS adapter includes an interface, internal registers, control paths and data paths. The MBA accepts and executes commands from the CPU and reports the necessary status changes and fault conditions to the CPU.

The MBA handles a MASSBUS drive with a maximum burst data transfer speed of 900 ns per 16 bits via the 16-bit wide MASSBUS data path. The MASSBUS adapter controls data transfers between MASSBUS devices and physical memory. A MASSBUS adapter transfers 16 bits at a time to a mass storage device or it receives 16 bits at a time from a MASSBUS drive. The MBA contains a 32-byte buffer used to store data enroute to either main memory or mass storage. Transfers (data only) to or from main memory, occur in 32-bit (4-byte) increments. Therefore, each memory transaction requires two MASSBUS transfers (16 bits each). The MASSBUS adapter will accept only aligned longword reads and writes to its external or internal registers. An attempt to address a nonexistent register in the MASSBUS adapter will prompt a no-response confirmation.

MBA REGISTERS

The MBA address space contains two sets of registers: internal and external. The MBA internal registers are the registers which are physically located in the MBA. The external registers are located in the MASSBUS drives and are drive-dependent.

There are six internal registers and a 256 × 32-bit RAM. The primary function of the internal registers is to monitor MBA and operating status conditions. The internal registers also control phases of the data transfers between the CMI and the MASSBUS device, such as:

- Maintaining a byte count to ensure that all of the data to be transferred have been accounted for
- Converting virtual addresses to physical addresses for referencing data in memory

The six internal registers are:

MBA Control Register (CR)
MBA Status Register (SR)
MBA Virtual Address Register (VAR)
MBA Byte Count Register (BCR)
MBA Diagnostic Register (DR)
MBA Command Address Register (CAR)

NOTE

The command address register is read-only and is valid only during data transfers.

The MBA contains 256 32-bit map registers which are used to map program virtual addresses into physical addresses. The mapping registers allow transfers to or from contiguous or non-contiguous physical memory. Bits <30:15> of the map register are reserved and are not writable. Figure 11-2 illustrates mapping a virtual address to a physical address.

DATA PATH

The data path controls the data transferred to and from the MASSBUS device and memory. The 32-bit data word is divided into 16-bit (2-byte) segments required for data on the MASSBUS. When performing a read from a MASSBUS device, the data path assembles the two 16-bit segments from the MASSBUS into the 32-bit format. A silo and input/output data buffer provide the means for smoothing the data transfer rate. The data path also contains a write check circuit which can be used under program control to verify the accuracy of the data transfer function.

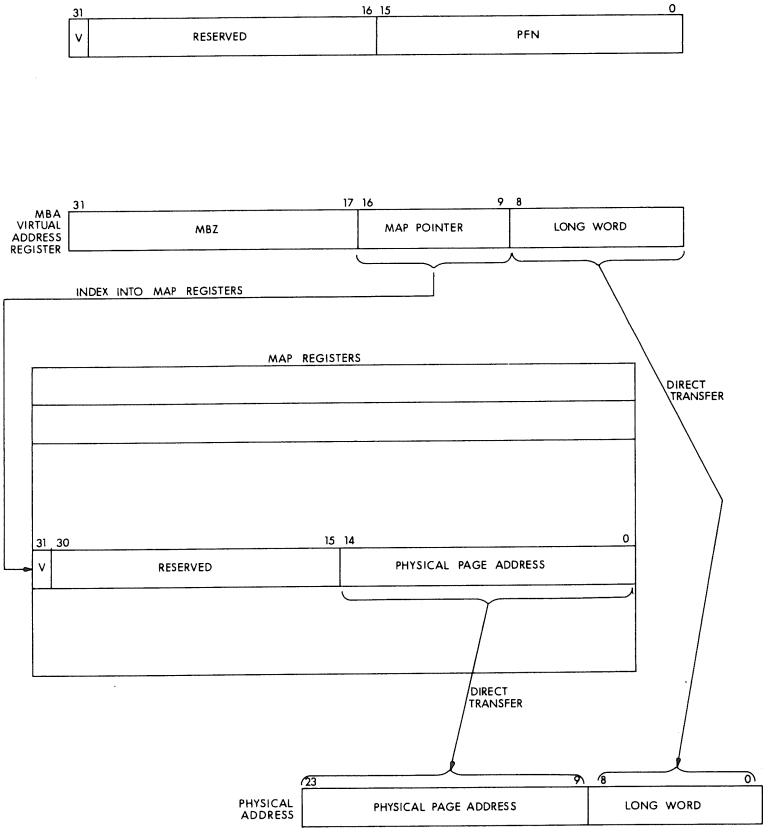


Figure 11-2 Virtual to Physical Address Translation

MBA ACCESS

Each device that interfaces to the VAX-11/750 has a block of addresses associated with it. Certain commonly addressed devices have preassigned address spaces, and other less frequently addressed devices use adapter codes. While the VAX-11/750 system handles up to three MASSBUS adapters, it has address space reserved for four. Each of the four blocks of addresses contains 8 KB, with the blocks beginning at addresses $F28000_{16}$, $F2A000_{16}$, $F2C000_{16}$, and $F2E000_{16}$. This space is accessible as part of the I/O address space. The command/address format used to access the MBA registers is illustrated in Figure 11-3.

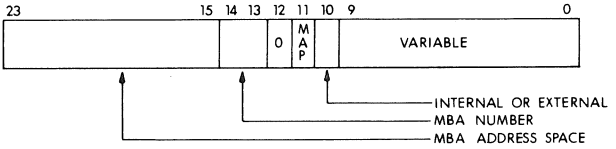


Figure 11-3 MASSBUS Adapter Addressing Format (Physical Byte Address)

Bit: <31:28>
Name: Bytes Mask
Function:

Bit: <27:25>
Name: Bus Function
Function:

Bit: <24> **Name:**
Function: Reserved for future use, zero.

Bit: <23:15>
Name: MBA Address
Function: To select MBA Address Space, this field must be $1E5_{16}$

Bit: <14:13>
Name: MBA Select
Function: Number of the MBA addressed.

Bit: <11:10>
Name: Register Select
Function:

0 0	MBA internal register Bit <9:5> must be zero Bit <4:2> = register select offset
0 1	MBA external register Bit <9:7> = device select Bit <6:2> = register select
1 0	MBA MAP Bit <9:2> = MAP address
1 1	Invalid (No response to an address with these bits on)

Bit: <9:0> **Name:**
Function: Reserved for future use.

MBA Control Register (Byte Offset = 4)

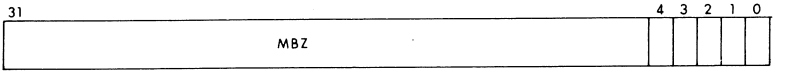


Figure 11-4 MBA Control Register (Byte Offset =4)

Bit: <31:5>
Name: MBZ
Function: Reserved for future use, all zeros.

Bit: <4> **Name:** IBC Mode
Function: Ignore Byte Count Mode. When this bit is set, a data transfer will not be terminated by byte counter overflow. Instead, the data transfer terminates when a signal is received telling the MASSBUS Adapter that the last byte has been transferred. This feature allows the system to read from magtapes with very long records (this requires a special device driver). In this mode, interrupts will be generated each time the byte counter overflows; the map registers are also writable. This bit is also cleared by writing a zero or by INIT.

Bit: <3> **Name:** MB Maintenance Mode (MMM)
Function: Setting this bit will put the MBA in the maintenance mode, which will allow the diagnostic programmer to exercise and examine the MASSBUS operations without using MASSBUS devices. When this bit is set, the MBA will send a signal to the MASSBUS so that all the devices on the MASSBUS will be detached from the MASSBUS. The MBA cannot be put in maintenance mode while a data transfer is in progress.

Bit: <2> **Name:** Interrupt Enable
Function: This bit is set by writing a one. This allows the MBA to interrupt the CPU when certain conditions occur. Cleared by writing a zero or by INIT.

Bit: <1> **Name:** Abort
Function: Abort data transfer. Write a one to set. Setting this bit will initiate the data transfer abort sequences which will stop sending commands, stop address and byte counter.

Setting this bit will also cause an interrupt to the CPU if the IE bit is one.

This bit will be cleared by writing a zero, or by INIT.

Bit: <0> **Name:** INIT
Function: Initialization. The bit is self-clearing. It will always read as zero. Setting this bit will:

1. Clear MBA Control register
2. Clear MBA Status register

3. Clear control and status bits of diagnostic registers
4. Cancel all pending commands
5. Abort data transfer
6. Assert MASSBUS INIT

MBA Status Register (Byte Offset = 8)

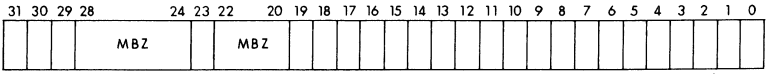


Figure 11-5 MBA Status Register (Byte Offset = 8)

Bit: <31> **Name:** DTBUSY
Function: Data Transfer Busy. Bit is set when a data transfer command is received. It is cleared when data transfer is terminated normally or when a data transfer is aborted.

Bit: <30> **Name:**
Function: Reserved for future use, zero.

Bit: <29> **Name:** CRD
Function: Corrected Read Data. This bit is set when the data received from memory were corrected. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command.

Bit: <28:24>
Name:
Function: Reserved for future use, all zeros.

Bit: <23> **Name:** CBHUNG
Function: Control Bus Hung. This bit is set if the TRANSFER (TRA) signal has been stuck in the asserted state for 1.5 μs after the MBA receives a read or write command to an external register and any previous external register operation is complete. When this bit is set, no data transfer operations are initiated because the user cannot address the device drives or registers. This bit is cleared by writing a one or by INIT.

Bit: <22:20 >
Name:
Function: Reserved for future use, all zeros.

Bit: <19> **Name:** PGE
Function: Programming Error. The PGE bit is set when one or more of the following conditions exists:

1. Program tries to initiate a data transfer when MBA is currently performing one.
2. Program tries to load MAP, VAR, or Byte Counter when MBA is currently performing a data transfer operation.
3. Program tries to set MB Maintenance Mode during a data transfer operation.

The bit is cleared by writing a one to it, or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will cause an interrupt to the CPU if Interrupt Enable is set.

Bit: <18> **Name:** NED
Function: Nonexistent Drive. This bit is set when a drive fails to assert the TRANSFER signal within 1.5 μs after asserting a DEMAND signal. The bit is cleared by writing a one or by INIT. Setting this bit will send zero read data back to memory and interrupt the CPU if Interrupt Enable is set.

Bit: <17> **Name:** MCPE
Function: MASSBUS Control Parity Error. This bit is set when a MASSBUS control parity error occurs. It is cleared by writing a one or by INIT. Setting this bit will cause an interrupt to the CPU if Interrupt Enable is set.

Bit: <16> **Name:** ATTN
Function: Attention from MASSBUS. Asserted when the attention line on the MASSBUS is asserted. Asserting this bit will cause an interrupt to the CPU if Interrupt Enable is set.

Bit: <15> **Name:**
Function: Reserved for future use, zero.

Bit: <14> **Name:** Silo Parity Error
Function: This bit is set when a silo parity error occurs during a data transfer operation. Cleared by writing a one to this bit or by INIT. Setting this bit will abort the data transfer operation. Subsequent receipt of a valid data transfer command will also clear this bit.

Bit: <13> **Name:** DTCMP
Function: Data Transfer Completed. This bit is set when the data transfer is terminated either due to an error or normal completion. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will cause an interrupt to the CPU if Interrupt Enable is set.

Bit: <12> **Name:** DTABT
Function: Data Transfer Aborted. This bit is set with the trailing edge

of the END OF BLOCK signal when the data transfer has been aborted. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will cause an interrupt to the CPU if Interrupt Enable is set.

Bit: <11> **Name:** DLT
Function: Data Late. This bit is set when:

1. The data buffer is empty and WCLK (the WRITE CLOCK signal indicates when data written to a drive are to be strobed) is sent to the MASSBUS during a Write Data Transfer or Write Check Data Transfer.
2. The data buffer is full when SCLK (SYNCHRONIZE CLOCK is asserted during a read operation to indicate when data read from the drive are to be strobed) is received from the MASSBUS during a read data transfer.

This bit is cleared by writing a one or by INIT. It is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <10> **Name:** WCK UP ERR
Function: Write Check Upper Error. This bit is set when a compare error is detected in the upper byte while the MBA is performing a write check operation. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <9> **Name:** WCK LWR ERR
Function: Write Check Lower Error. This bit is set when a compare error is detected in the lower byte while the MBA is performing a write check operation. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <8> **Name:** MXE
Function: Miss Transfer Error. This bit is set when an OCC (an OCCUPIED signal) is not received within 500 μs after data transfer busy is set. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will cause an interrupt to the CPU if Interrupt Enable is set.

Bit: <7> **Name:** MBEXC
Function: MASSBUS Exception. This bit is set when EXC (the EXCEPTION signal indicates an error condition during a data transfer) is received from MASSBUS. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <6> **Name:** MDPE
Function: MASSBUS Data Parity Error. This bit is set when a MASSBUS data parity error is detected during a read data transfer operation. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <5> **Name:** MAPPE
Function: Page Frame Map Parity Error. This bit is set when a parity error is detected on the page frame number read from the PF map. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <4> **Name:** INVMAP
Function: Invalid Map. This bit is set when the valid bit of the next page frame number is zero when the byte count is not zero. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <3> **Name:** ERR STAT
Function: Error Status. This bit is set when the MBA receives error status for the read command or write command. It is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will cause the data transfer operation to be aborted.

Bit: <2> **Name:**
Function: Reserved for future use, zero.

Bit: <1> **Name:** NRSTAT
Function: No Response Status. This bit is set when the MBA receives no response status for a read or write command to memory. This bit is cleared by writing a one or by INIT. This bit is also cleared by subsequent receipt of a valid data transfer command. Setting this bit will abort the data transfer operation.

Bit: <0> **Name:**
Function: Reserved for future use, zero.

MBA Virtual Address Register (Byte Offset = 12)

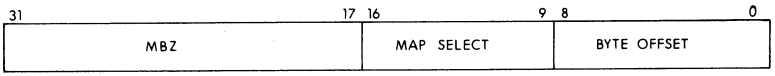


Figure 11-6 MBA Virtual Address Register (Byte Offset = 12)

The program must load an initial virtual address (pointing to the first byte to be transferred) into this register before a data transfer is initiated. Bits <16:9> select one of the 256 map registers. The contents of the selected map and the values in bits <8:0> are used to assemble a physical address to be sent to memory. Bits <8:0> indicate the byte offset of the current data byte into the page. Note that the MBA virtual address register is incremented by four after every memory read or write and will not point to the next byte to be transferred if the transfer does not end on a longword boundary. (It will point four bytes ahead.) Also, upon a write check error, the virtual address register will not point to the failing data in memory due to the preloading of the silo data buffer. The virtual address of the bad data may be found by determining the number of bytes actually transfered compared to the MASSBUS (the difference between bits <31:16> of the MBA Byte Counter and their initial value) and adding that difference to the initial virtual address.

MBA Byte Counter (Byte Offset = 16)

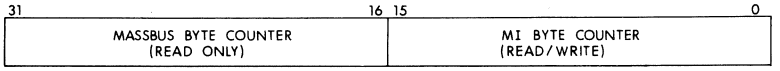


Figure 11-7 MBA Byte Counter (Byte Offset = 16)

The program writes the 2's complement of the number of bytes for the data transfer to bits <15:0> of this register. MBA hardware will load these 16 bits into bits <31:16> *and* bits <15:0>. Bits <31:16> serve as the byte counter for the number of bytes transferred to or from the drive. Bits <15:0> serve as the byte counter for the number of bytes transferred to or from memory. The starting byte count with 16 bits of zero is the maximum number of bytes of a data transfer.

MBA Diagnostic Register (Byte Offset = 20)

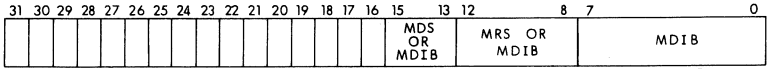


Figure 11-8 Diagnostic Register (Byte Offset = 20)

The diagnostic register may be written only while in maintenance mode.

Bit: <31> **Name:** IMDPG
Function: Invert MASSBUS Data Parity Generator.

Bit: <30> **Name:** IMCPG

Function: Invert MASSBUS Control Parity Generator.

Bit: <29> **Name:** IMAPP
Function: Invert Map Parity Checking.

Bit: <28> **Name:** BLKSCOM
Function: Block Sending Command. During a data transfer, setting this bit will eventually cause the DLT (Data Late—bit <11> of the MBA Status Register) bit to be set and interrupt the CPU.

Bit: <27> **Name:** SIMSCLK
Function: Simulate SCLK. When the MB Maintenance Mode bit is set, writing a one to this bit will simulate the assertion of SCLK (SYNCHRONIZE CLOCK); and writing a zero to this bit will simulate the deassertion of SCLK.

Bit: <26> **Name:** SIMEBL
Function: Simulate EBL. When the MB Maintenance Mode bit is set, writing a one and writing a zero to this bit will simulate the assertion and deassertion of EBL (END OF BLOCK).

Bit: <25> **Name:** SIMOCC
Function: Simulate OCC. When the MB Maintenance Mode bit is set, writing a one and writing a zero to this bit will simulate the assertion and deassertion of OCC (OCCUPIED).

Bit: <24> **Name:** SIMATTN
Function: Simulate ATTN. When the MB Maintenance Mode bit is set, writing a one or a zero to this bit will simulate the assertion and deassertion of ATTN. (ATTENTION is asserted by drives to signal the MBA of changes in a drive's status or abnormal conditions.)

Bit: <23> **Name:** MDIB SEL
Function: Maintenance MASSBUS Data Buffer Select. This bit selects what is to be sent out from bits <15:8> when the diagnostic register is read. When this bit is set to one, the upper byte of the MDIB (read-only) is selected. When this bit is set to zero, Maintenance Drive and Register Select (read-only) are selected.

Bit: <22> **Name:** ISPG
Function: Invert Silo Parity Generator.

Bit: <21> **Name:** SIMEXC
Function: Simulate EXC. When the MB Maintenance Mode bit is set, writing a one or a zero to this bit will simulate the assertion and deassertion of the EXCEPTION signal.

Bit: <20> **Name:** MFAIL
Function: MASSBUS Fail (read-only). Fail is asserted when MMM is set.

Bit: <19> **Name:** MRUN
Function: Maintenance MASSBUS Run (read-only).

Bit: <18> **Name:** MWCLK
Function: Maintenance MASSBUS WCLK (read-only).

Bit: <17> **Name:** MEXC
Function: Maintenance MASSBUS EXC (read-only).

Bit: <16> **Name:** MCTOD
Function: Maintenance MASSBUS CTOD (read-only).

Bit: <15:13>
Name: MDS or MDIB <15:13>
Function: Maintenance MASSBUS Device Select (read-only) or MDIB <15:13> (read-only) as controlled by bit <23>.

Bit: <12:8>
Name: MRS or MDIB
Function: Maintenance MASSBUS Register Select (read-only) or MDIB <12:8> (read-only) as controlled by bit <23>.

Bit: <7:0> **Name:** MDIB
Function: Maintenance MDIB.

MBA Command Address Register (Byte Offset = 28)
This register is read-only and valid only when DT Busy (bit <31> of the MBA Status Register) is set. The bit assignments are as follows:

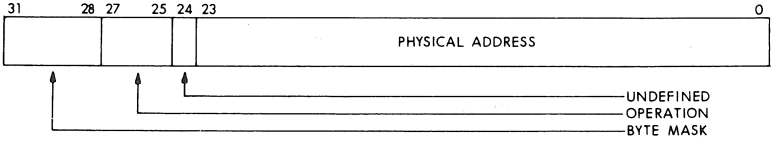


Figure 11-9 Command Address Register (Byte Offset = 28)

Bit: <31:28>
Name: Byte Mask
Function: On a write or write unlock, these bits correspond to bytes zero through three and tell memory which bytes to write.

Bit: <27:25>
Name: Operation
Function: These bits define the operation to be performed. Possible operations are:

000	Read to Memory
100	Write
110	Interrupt

Bit: <24> **Name:**
Function: Undefined

Bit: <23:0>
Name: Address
Function: These bits specify the physical address of the activity to memory.

MBA External Registers (Byte Offset = 400 to 7FC)

External registers are MASSBUS device-dependent. Each device has a maximum of 32 registers.

MBA Map Registers (Byte Offset = 800 to BFC)

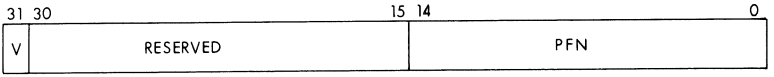


Figure 11-10 MBA Map Registers (Byte Offset = 800 to BFC)

Bit: <31> **Name:** Valid Bit
Function:

Bit: <30:15>
Name:
Function: Reserved for future use, all zeros.

Bit: <14:0>
Name: PFN
Function: Physical Page Frame Number. The MBA contains 256 map registers, each of which may be selected by address bits <9:2> when bits <11:10> are one and zero, respectively. Map registers can only be written when there is no data transfer operation in progress or the IBC bit (Ignore Byte Count—bit <4> of the MBA Control Register) is set. A write to a map register during a data transfer with IBC clear will be ignored and cause PGE to set.

DATA TRANSFER PROGRAM FLOW

1. Initialize MASSBUS Adapter
2. Mount pack into drive
3. Start drive spinning
4. Wait for ready light
5. Issue Pack ACK to drive
6. Load desired cylinder, sector, track, and registers in drive
7. Load starting virtual address into MBA's virtual address register
8. Load 2's complement of number of bytes to be transferred into byte count register in MBA

9. Load starting map (pointed to by bits <16:9> of VAR) with physical page address
10. Load successive maps with physical addresses to rest of pages
11. Issue read/write command to drive


digital VAX 11/750

CHAPTER 12

VAX-11/750 PRIVILEGED REGISTERS

INTRODUCTION

The processor register space provides access to many types of CPU control and status registers such as the memory management base registers, the Processor Status Longword, and the multiple stack pointers. The benefit of privileged registers is that these key registers are explicitly accessible only by the Move to Processor Register (MTPR) and Move from Processor Register (MFPR) instructions which are controlled by the kernel executive. In a VAX/VMS environment, the operating system conveniently manages these registers for the user; therefore, the detailed privileged register information contained in this chapter will be useful only to system designers who will not be using VAX/VMS.

A complete list of VAX-11/750 internal processor registers may be found in Appendix F.

SYSTEM IDENTIFICATION REGISTER (SID)

The system identification register is a read-only constant register that specifies the processor type. The entire SID register is included in the error log and the type field may be used by software to distinguish processor types. Figure 12-1 illustrates the system identification register.

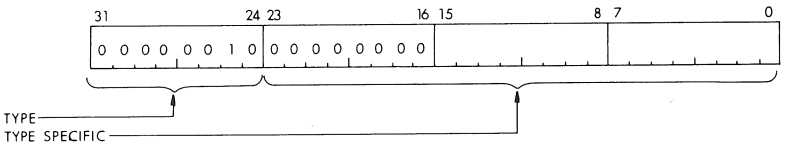


Figure 12-1 System Identification Register IPR #$3E_{16}$

Type	A unique number assigned by engineering to identify a specific processor:
0	Reserved to DIGITAL
1	VAX-11/780
2	VAX-11/750
3	VAX-11/730
4 through 127	Reserved to DIGITAL
128 through 255	Reserved to CSS and customers

For the VAX-11/750, the type-specific format is shown in Figure 12-2.

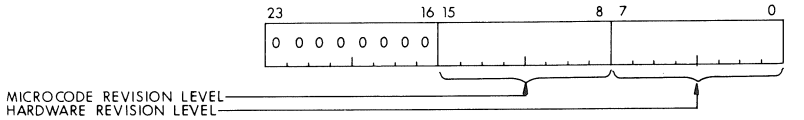


Figure 12-2 Type-Specific Format

CONSOLE TERMINAL REGISTERS

The console terminal is accessed through four internal registers. Two are associated with receiving from the terminal and two with writing to the terminal. In each direction there is a control/status register and a data buffer register. Figure 12-3 illustrates the console receive control/status register, and the bit assignments are described.

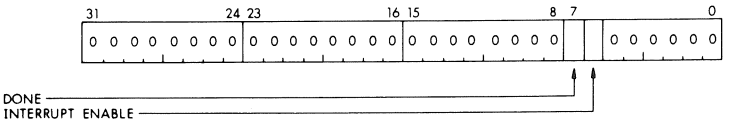


Figure 12-3 Console Receive Control/Status Register (RXCS)
IPR #20_{16}

Bit: 31:8 **Name:** MBZ
Function: Must be zero

Bit: 7 **Name:** Done
Function: This bit is read-only and is set by the console whenever a datum is received. Done is initialized to 0 at bootstrap time and is cleared whenever MFPR #RXDB,dst is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 20 when Done becomes set. Similarly, if Done is already set and the software sets IE, an interrupt is generated. This bit is initialized to 0 at bootstrap time, and can be read or written by software.

Bit: 5:0 **Name:** MBZ
Function: Must be zero

Figure 12-4 illustrates the read-only console receive data buffer register. The bit assignments follow.

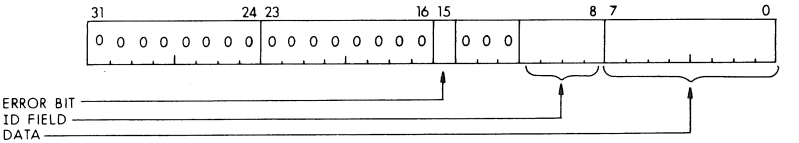


Figure 12-4 Console Receive Data Buffer Register (RXDB) IPR #11_{16}

Bit: 31:16 **Name:** MBZ
Function: Must be zero

Bit: 15: **Name:** ERR
Function: Error bit. If the received data contained an error such as overrun or loss of connection, then ERR is set.

Bit: 14:12 **Name:** MBZ
Function: Must be zero

Bit: 11:8 **Name:** ID
Function: If ID are zero, then the data is from the console terminal. If ID is nonzero, then the entire register is implementation-dependent.

Bit: 7:0 **Name:** Data
Function: This field contains the actual data received by the console.

Figure 12-5 illustrates the console transmit control/status register; the bit descriptions are also provided.

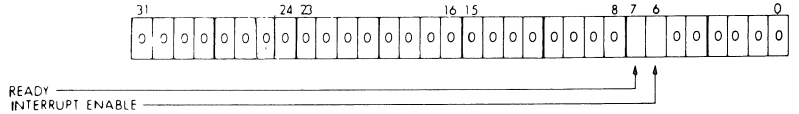


Figure 12-5 Console Transmit Control/Status Register (TXCS) IPR #22_{16}

Bit: 31:8 **Name:** MBZ
Function: Must be zero

Bit: 7 **Name:** RDY
Function: Ready. This bit is read-only and is set at bootstrap time. It is also set whenever the console transmitter is not busy. This bit is cleared when MTPR src,#TXDB is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 20 when RDY becomes set. If RDY is already set and software sets IE, an interrupt is also generated. This bit is cleared when MTPR src,#TXDB is executed.

Bit: 5:0 **Name:** MBZ
Function: Must be zero

Figure 12-6 illustrates the read-only console transmit data buffer register. The bit assignments are described.

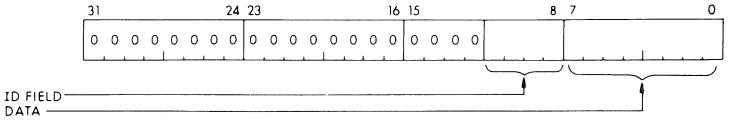


Figure 12-6 Console Transmit Data Buffer Register (TXDB) IPR #23_{16}

Bit: 31:12 **Name:** MBZ
Function: Must be zero

Bit: 11:8 **Name:** ID
Function: When an MTPR src,#TXDB is executed and these bits are written as 0, data will be sent to the console terminal. If ID is F_{16}, then the Data field can have five values. If ID is other than 0 or F_{16}, it indicates a reserved operand.

Bit: 7:0 **Name:** Data
Function: This field contains the actual data transmitted by the console. If ID is F, 0, 1 or 3 is a no-op; a 2 will cause a boot, and a 4 will clear a cold start flag. With ID = F, any other value will cause a reserved operand fault.

TU58 REGISTERS

The console TU58 tape cartridge subsystem is accessed through four internal registers. Two are associated with receiving from the TU58 and two with writing to it. In each direction there is a control/status register and a data buffer register. Figure 12-7 illustrates the console storage receive status register. Description of the bit assignments follow the figure.

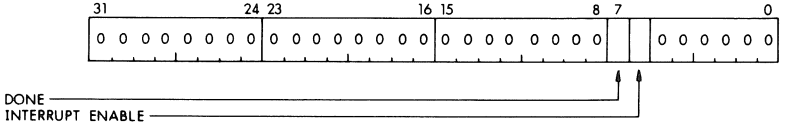


Figure 12-7 Console Storage Receive Status (CSRS) IPR #$1C_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero

Bit: 7 **Name:** Done
Function: This bit is read-only and is set by the TU58 whenever a datum is received by the TU58 from the console storage receive data register. Done is initialized to 0 at bootstrap time and is cleared whenever MFPR #CSRD, dst is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 23. Similarly, if Done is already set and the software sets IE, an interrupt is generated. This bit is initialized to 0 at bootstrap time, and can be read or written to by software.

Bit: 5:0 **Name:** MBZ
Function: Must be zero

Figure 12-8 illustrates the console storage receive data register. Bit assignments follow the illustration.

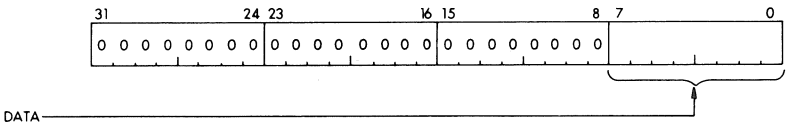


Figure 12-8 Console Storage Receive Data (CSRD) IPR #1D$_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero

Bit: 7:0 **Name:** Data
Function: This field contains the actual data received from the TU58 subsystem.

Figure 12-9 illustrates the console storage transmit status register. Bit descriptions follow the figure.

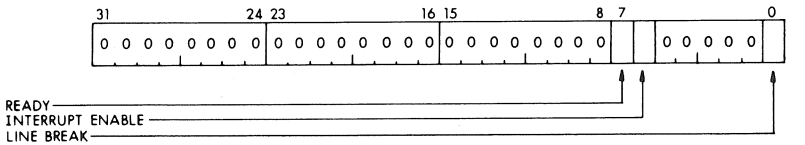


Figure 12-9 Console Storage Transmit Status (CSTS) IPR #1E$_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero

Bit: 7 **Name:** RDY
Function: Ready. This bit is read-only and is set at bootstrap time. It

is also set whenever the TU58 transmitter is not busy. This bit is cleared when MTPR src, #CSTD is executed.

Bit: 6 **Name:** IE
Function: Interrupt Enable. If this bit is set by software, an interrupt is generated at IPL 23. If RDY is already set and software sets IE, an interrupt is also generated. This bit is cleared when MTPR src, #CSTD is executed.

Bit: 5:1 **Name:** MBZ
Function: Must be zero

Bit: 0 **Name:** LB
Function: Line Break. When this bit is written to a 1, a line break is issued to the TU58. This bit is write-only.

Figure 12-10 illustrates the console storage transmit data register. Bit descriptions follow the illustration.

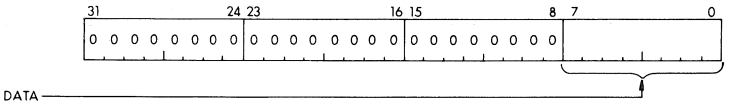


Figure 12-10 Console Storage Transmit Data (CSTD) IPR $\#1F_{16}$

Bit: 31:8 **Name:** MBZ
Function: Must be zero

Bit: 7:0 **Name:** Data
Function: This field contains the actual data transmitted by the TU58 subsystem.

CLOCK REGISTERS

The clocks consist of a time-of-year clock and an interval clock. The time-of-year clock is used by the operating system to reinitialize the time and date after power has been off, without requiring an operator to type in the information. The time-of-year clock is also used for time stamping of user and operating system programs. The interval clock is used for accounting, for time-dependent events, and to maintain the software date and time.

Time-of-Year Clock

The time-of-year clock consists of one longword register. The register forms an unsigned 32-bit binary counter that is driven by a precision clock source. (Clock accuracy is typically .0025% but will vary slightly with ambient temperature and remaining life of the battery backup

system.) The counter has a battery backup power supply sufficient for at least 100 hours of operation, and the clock does not gain or lose any ticks during transition to or from stand-by power. The battery is recharged automatically. The least significant bit of the counter represents a resolution of 10 ms. Thus, the counter cycles to zero after approximately 497 days.

If the battery has failed, so that time is not accurate, the register is cleared on power-up. It is held at zero until software writes a nonzero value to it. Thus, if software initializes this clock to a value corresponding to a large unit of time (e.g., a month), it can check for loss of time after a power restore by checking the clock value. The time-of-year clock register is illustrated in Figure 12-11.

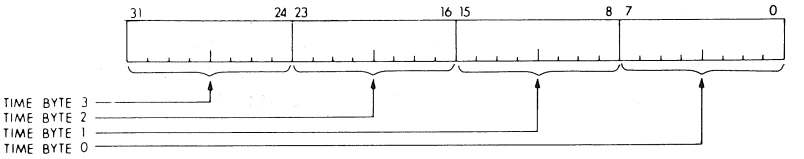


Figure 12-11 Time-of-Year Clock Register (TODR) IPR #$1B_{16}$

Interval Clock

The interval clock provides an interrupt at IPL 24 at programmed intervals. The counter is incremented at 1 μs intervals, with a typical accuracy of .01% or 8.64 seconds per day. (Accuracy will vary slightly with ambient temperature.) The clock interface consists of three registers in the privileged register space: the read-only Interval Count Register, the write-only Next Interval Count Register, and the Interval Clock Control/Status Register.

Interval Count Register

The Interval Count Register is a read-only register incremented once every microsecond. It is automatically loaded from NICR (Next Interval Count Register) upon a carry out from bit 31 (overflow) which also causes an interrupt request at IPL 24 if the interrupt is enabled. Figure 12-12 illustrates the Interval Count Register.

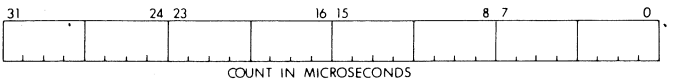


Figure 12-12 Interval Count Register (ICR) IPR #$1A_{16}$

Next Interval Count Register

The reload register is a write-only register that holds the value to be loaded into ICR when it overflows. The value is retained when ICR is loaded. NICR is capable of being loaded regardless of the current values of ICR and ICCS (Interval Clock Control/Status Register). Figure 12-13 illustrates the Next Interval Count Register.

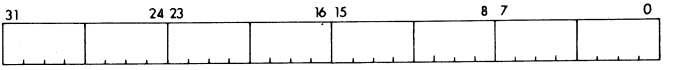


Figure 12-13 Next Interval Count Register (NICR) IPR #19_{16}

Interval Clock Control/Status Register (ICCS)

The ICCS register contains control and status information for the interval clock. Figure 12-14 illustrates the Interval Clock Control/Status Register, and the bit assignments are described in detail.

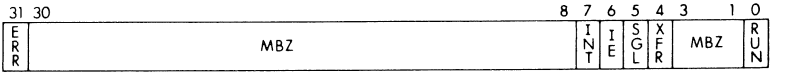


Figure 12-14 Interval Clock Control/Status Register (ICCS) IPR #18_{16}

Bit: 31 **Name:** ERR
Function: Whenever ICR overflows, if INT is already set, then ERR is set. Thus, ERR indicates one or more missed clock ticks. Attempts to set this bit via MTPR clears ERR.

Bit: 30:8 **Name:** MBZ
Function: Must be zero

Bit: 7 **Name:** INT
Function: This bit is set by hardware every time ICR overflows. If IE is set then an interrupt is also generated. An attempt to set this bit via MTPR clears INT, thereby re-enabling the clock tick interrupt (if IE is set).

Bit: 6 **Name:** IE
Function: When this bit is set, an interrupt request at IPL 24 is generated every time ICR overflows (INT is set). When clear, no interrupt is requested. Similarly, if INT is already set and the software sets IE, an interrupt is generated (i.e., an interrupt is generated whenever the function (IE .AND. INT) changes from 0 to 1).

Bit: 5 **Name:** SGL
Function: This bit is write-only. If Run is clear, each time this bit is set, ICR is incremented by one.

Bit: 4 **Name:** XFR
Function: This bit is write-only. Each time this bit is set, NICR is transferred to ICR.

Bit: 3:1 **Name:** MBZ
Function: Must be zero

Bit: 0 **Name:** Run
Function: When this bit is set, ICR increments each microsecond. When clear, ICR does not increment automatically. At bootstrap time, Run is cleared.

Thus, to set up the interval clock, load the negative of the desired interval into NICR. Then an MTPR #↑X51,#ICCS will enable interrupts, reload ICR with the NICR interval and set Run. Every "interval count" microseconds will cause INT to be set and an interrupt to be requested. The interrupt routing should execute an MTPR #↑XC1,#ICCS to clear the interrupt. If INT has not been cleared (i.e., if the interrupt has not been handled) by the time of the next ICR overflow, the ERR bit will be set.

At bootstrap time, bits <6> and <0> of ICCS are cleared. The rest of ICCS and the contents of NICR and ICR are UNPREDICTABLE.

MACHINE CHECK ERROR SUMMARY REGISTER (MCESR)

The Machine Check Error Summary Register (MCESR) logs information about causes of a machine check—for instance, a bus error. The entire register is read/write and all bits are set to 0 at bootstrap time. As shown in Figure 12-15, only bits <3:0> are implemented; descriptions are provided.

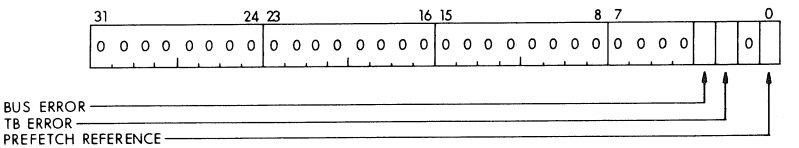


Figure 12-15 Machine Check Error Summary Register (MCESR)
IPR $\#26_{16}$

Bit: 3 **Name:** Bus Error
Function: This bit is set when a machine check results from either a read to nonexistent memory or a read of uncorrectable data.

Bit: 2 **Name:** TB Error
Function: This bit is set when a machine check results from a trans-

lation buffer error; it is cleared by writing a 1.

Bit: 0 **Name:** XB Error

Function: Execution Buffer Error. This bit is set when an error is detected trying to use data from the execution buffer. This bit is cleared by writing a 1.

MACHINE CHECK STATUS REGISTER (MCSR)

This register provides additional information about bus errors and translation buffer errors causing a machine check. The MCSR is illustrated in Figure 12-16 and the bit assignments are described.

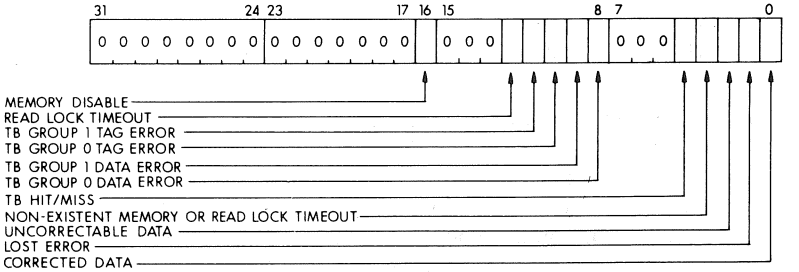


Figure 12-16 Machine Check Error Type Register (MCSR) IPR #17_{16}

Bit: 31:17 **Name:** MBZ

Function: Must be zero

Bit: 16 **Name:** Memory Disable

Function: This bit is read-only by software and is initially 0. If set, only the cache is read and write (i.e., main memory is not referenced or modified). It is highly unlikely that this bit would ever be set; it would indicate a serious CPU failure.

Bit: 15:13 **Name:** MBZ

Function: Must be zero

Bit: 12 **Name:** Read Lock

Function: This bit is read-only by software and is initially zero. It is set by a Read Lock Timeout and defines how bits <3:0> are interpreted.

Bit: 11:8 **Name:** TBGPE

Function: These bits are read-only and are set to 0 at bootstrap time. They define the type of translation buffer group parity error and are cleared when bit <2> of the MCESR is reset.

Bit: 7:5 **Name:** MBZ

Function: Must be zero

Bit: 4 **Name:** Hit/Miss

Function: This bit is read-only and is set to 0 at bootstrap time. It

indicates whether the last microcoded reference resulted in a hit or a miss.

Bit: 3:0 **Name:** Bus Error
Function: These bits are read-only and are set to 0 at bootstrap time. They define the type of bus error and are cleared when bit <3> of the MCESR is reset.

If bit <3> is set, a nonexistent memory or read lock timeout error is indicated. If bit<2> is set, an uncorrectable data error is indicated. If bit<1> is set, a lost bus error is indicated. If bit<0> is set, the indication is corrected data.

TRANSLATION BUFFER GROUP DISABLE REGISTER (TBDR)

The Translation Buffer Group Disable Register (TBDR) controls the enabling and disabling of the translation buffer and controls which half of the buffer is replaced. This register is illustrated in Figure 12-17 and the bit assignments are described following.

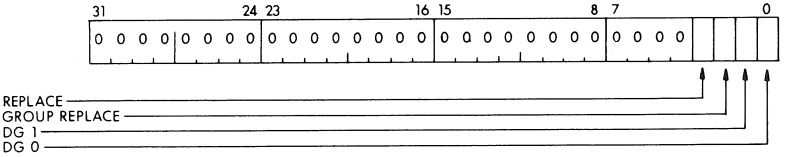


Figure 12-17 Translation Buffer Group Disable Register (TBDR) IPR $\#24_{16}$

Bit: 31:4 **Name:** MBZ
Function: Must be zero

Bit: 3 **Name:** Replace
Function: This bit is read/write and is set to 0 at bootstrap time. When this bit is set to 1, bit <2> controls which half of the translation buffer is replaced.

Bit: 2 **Name:** Group Replace
Function: This bit is read/write and is set to 0 at bootstrap time. When bit <3> is 1 and this bit is 0, group zero will be replaced. When bit <3> is 1 and this bit is also 1, group one will be replaced.

Bit: 1 **Name:** DG1
Function: This bit is read/write and is set to 0 at bootstrap time. When this bit is a 1, group one of the translation buffer will be disabled.

Bit: 0 **Name:** DG0
Function: This bit is read/write and is set to 0 at bootstrap time. When this bit is a one, group zero of the translation buffer will be disabled.

NOTE

The machine cannot be run with mapping enabled and DG1 and DG0 set.

CACHE DISABLE REGISTER (CADR)

This register is used to turn off the cache. This bit is cleared on power-up. Figure 12-18 illustrates the cache disable register.

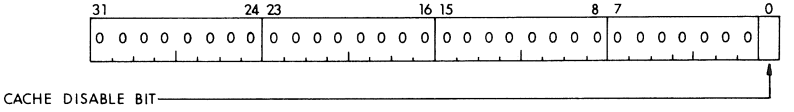


Figure 12-18 Cache Disable Register (CADR) IPR $\#25_{16}$

Bit: 31:1 **Name:** MBZ
Function: Must be zero

Bit: 0 **Name:** Cache Disable Bit
Function: If this read/write bit is 0, the cache is enabled to operate normally. If this bit is a one, the cache is turned off.

CACHE ERROR REGISTER (CAER)

The Cache Error Register (CAER) logs information about the nature of cache parity errors. This register is read/write, and only bits <3:0> are implemented. All bits are set to 0 at bootstrap time. This register is illustrated in Figure 12-19 and the bit assignments are described below.

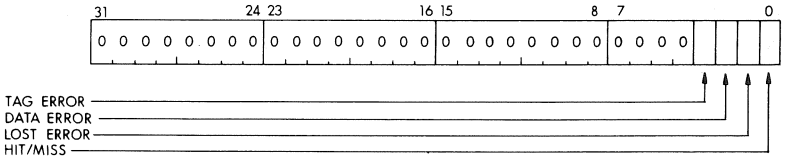


Figure 12-19 Cache Error Register (CAER) IPR $\#27_{16}$

Bit: 31:4 **Name:** MBZ
Function: Must be zero

Bit: 3 **Name:** Tag Error
Function: If this bit is set, it indicates that the parity error was in the tag portion of the cache.

Bit: 2 **Name:** Data Error

Function: If this bit is set, it indicates that the parity error was in the data portion of the cache.

Bit: 1 **Name:** Lost Error
Function: If this bit is set, it indicates that there were multiple cache errors.

Bit: 0 **Name:** Hit/Miss
Function: If this bit is 0, it indicates that the last microcoded reference resulted in a miss; if the bit is 1, the last microcoded reference resulted in a hit.

TRANSLATION BUFFER REGISTER

The ability to read and write locations in the VAX-11/750 translation buffer is important for diagnostic purposes. The MTPR and MFPR instructions are two-operand instructions. One operand specifies a longword source or destination, and the other operand specifies the IPR number of TB. For accessing the translation buffer, a third operand containing the address of the translation buffer location to be referenced is needed. The virtual address field of the P0BR supplies this virtual address. For MTPR, the P0BR contains the virtual address whose page table entry (PTE taken from source operand) is to be written into the translation buffer. For MFPR, the P0BR contains the virtual address whose PTE is to be read from the translation buffer into the destination.

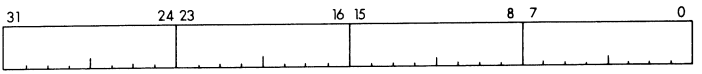


Figure 12-20 Translation Buffer Register (TB) IPR #$3B_{16}$

NOTE

For this process to work correctly, memory management **must** be disabled by clearing the Map Enable Register. Furthermore, only stand-alone diagnostics may use this feature.

VAX-11/750 OPTIONAL FLOATING POINT ACCELERATOR

The VAX-11/750 has an optional accelerator for a subset of the instructions. The ACCS, an internal read/write register, controls the accelerator.

ACCS is the accelerator control/status register. It indicates whether an accelerator exists, controls whether it is enabled, identifies its type and reports errors and status. At bootstrap time, the type and enable are

set; the errors are cleared. Figure 12-15 illustrates the accelerator control/status register.

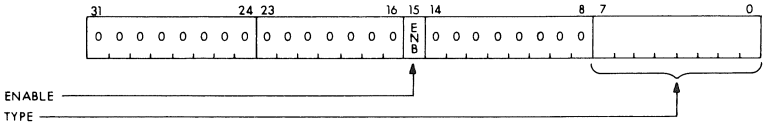


Figure 12-21 Accelerator Control/Status Register (ACCS)

Bit: 31:16 **Name:** MBZ
Function: Must be zero.

Bit: 15 **Name:** ENB
Function: Write-only field specifying whether the accelerator is enabled.

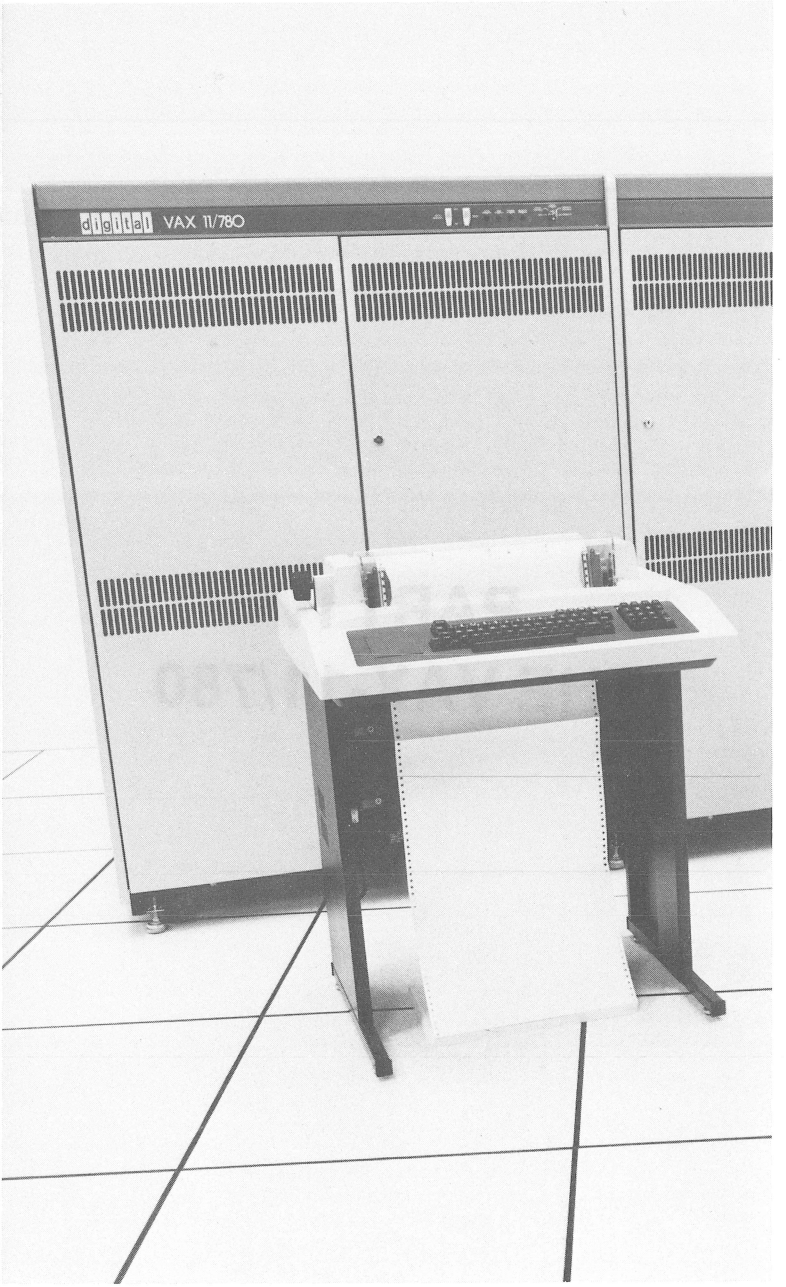
Bit: 14:8 **Name:** MBZ
Function: Must be zero.

Bit: 7:0 **Name:** TYPE
Function: Read-only field specifying the accelerator type as follows:

0 = No accelerator
1 = Floating Point Accelerator

Numbers in the range 2 through 127 are reserved to DIGITAL. Numbers in the range 128 through 255 are reserved to CSS/customers.

PART IV
THE VAX-11/780


digital VAX 11/780

CHAPTER 13

VAX-11/780 CONSOLE SUBSYSTEM

FEATURES	BENEFITS
LSI-11 console microcomputer	Performs diagnostics and simplifies bootstrapping and system initialization
Diagnostic console	Allows operator diagnostic operations through simple keyboard commands. Can be used as an operator console and as a user terminal
Console command language	Gives the user a powerful, yet easy-to-use, debugging tool
Console terminal is a standard ASCII device	Provides a high degree of flexibility
EIA communications interfacing	Allows standard industry-compatible communications
Front panel switches	Offer control over certain aspects of the machine operation
RX01 floppy disk drive	Provides an inexpensive, reliable device and medium for booting, diagnostics, and field updates to software
Unattended restart	The system restarts or reboots itself upon recovery of electricity after a power failure or other system crash

INTRODUCTION

The console subsystem serves as the interface between the operator and the VAX-11/780 system. The console subsystem provides the user with improved system maintenance features and greater operating system flexibility. The user interface to the subsystem is via the console command language, which is quite similar to the system command language. The traditional lights and toggle switch functions have been replaced by simple English language commands entered into the system terminal. The system terminal (OPA0) is the logical first terminal of the system. The floppy disk , an integral part of the subsys-

tem, stores microdiagnostics and system software. This facilitates fast diagnosis (initiated both locally and remotely), simplified system bootstrapping and initialization, and improved software update distribution. Figure 13-1 functionally illustrates the console subsystem.

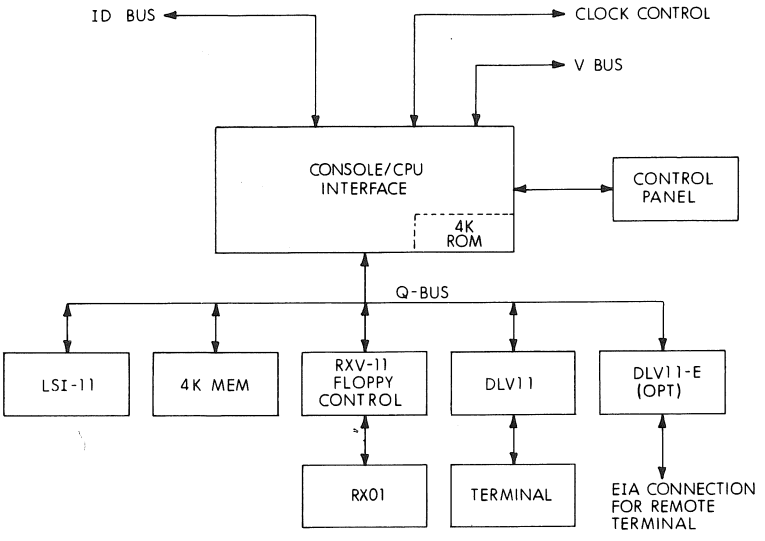


Figure 13-1 Console Subsystem

The console subsystem is comprised of six major components:

- An LSI-11 microprocessor (KD11-F) including a 4 K by 16-bit semiconductor random access memory (RAM).
- A floppy disk drive (RX01) and controller (RXV11).
- A system terminal and two serial line interface units, one serial line unit provided for optional remote diagnosis port.
- Console interface board (CIB) including 4 K by 16-bit read only memory (ROM) for the LSI-11 microprocessor.
- The control panel on the VAX-11/780 cabinet.
- Bus structure. The internal data (ID) bus is a high speed data path connecting major functional areas of the VAX-11/780 CPU.

CONSOLE INTERFACE BOARD

The Console Interface Board links the console subsystem to the VAX-11/780 central processor. The CIB contains interfaces to the console

subsystem bus structures; registers accessible to each bus; and all the hardware necessary to implement the console functions. In addition, the CIB contains a 4 K by 16-bit ROM which provides the core of the console LSI-11 software.

All data transfer operations between the VAX-11/780 processor and the console LSI-11 are routed via the TO Internal Data and FM Internal Data privileged registers on the CIB. The interaction of the console subsystem and the VAX-11/780 processor, however, is directly related to the states of the two processors. The VAX-11/780 processor may be either running or halted. When running, the VAX processor is executing normal VAX code. The processor can then be halted in one of two ways:

- Internal system error
- Halt command via console (console must be in console command mode to activate halt command)

If the processor is halted via an error detection, the console subsystem automatically enters the console command mode (e.g., CPU double-error halt).

The LSI-11 may perform in either the program I/O mode or the console command mode. When the LSI-11 is in the program I/O mode, it passes console terminal input character by character to the VAX-11/780 software. Data sent from the VAX-11/780 software to the console terminal is passed by the LSI-11 software directly to the terminal. When the LSI-11 is in the console command mode, it interprets all console terminal output in order to perform diagnostic and maintenance functions and to implement the console command language (CCL). Therefore, four possible system states could exist. They are:

- VAX-11/780 running—LSI-11 program I/O mode
- VAX-11/780 running—LSI-11 console command mode
- VAX-11/780 halted—LSI-11 program I/O mode
- VAX-11/780 halted—LSI-11 console command mode

Figure 13-2 illustrates the VAX-11/780 and LSI-11 interaction and operating mode combinations.

VAX-11/780 Running — LSI-11 in Program I/O Mode

In this mode of operation, the console terminal acts like any other user terminal and may be used in conjunction with normal user application programming. The Console Interface Board (CIB) passes character data between both processors. In this mode, the LSI-11 console software does not interpret commands typed at the console terminal.

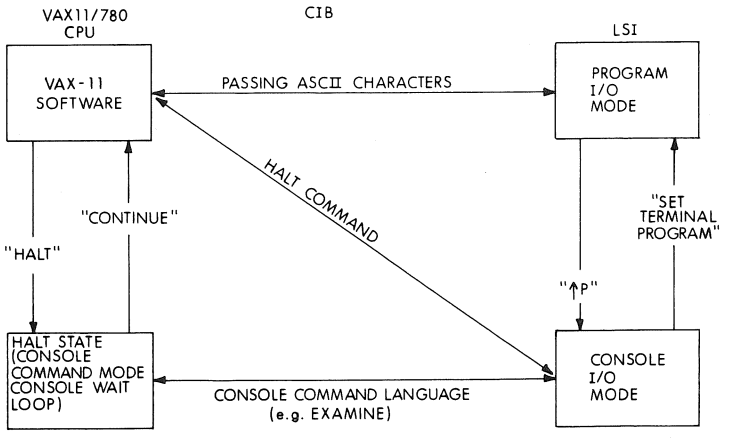


Figure 13-2 VAX-11/780 and LSI-11 Interaction and System Operating States

VAX-11/780 Running — LSI-11 in Console Command Mode

In this mode, the operator is able to halt the VAX-11/780 processor via the console terminal by typing the HALT command, and resume execution of the processor by entering the CONTINUE command. However, by entering the CONTINUE command, the console is automatically updated to program I/O mode. When the VAX-11/780 is executing instructions and the LSI-11 is in the program I/O mode, to halt the VAX processor, the operator must change console modes from program I/O mode to console command mode and then input the HALT command.

The system operator can enter the console command mode from the program I/O mode by typing control-P (↑P). Similarly, the operator can change from console command mode to program I/O mode by typing "Set Terminal Program". While the VAX processor is executing code, only the following subset of commands are permitted:

- SHOW
- SET
- WAIT DONE
- HELP
- EXAMINE /VBUS
- CLEAR
- HALT

Note that the functions which may be performed by the console are limited to those requiring no direct response by the VAX-11/780 processor (except HALT). The console software does not pass commands to the executing VAX processor software. Conversely, the console will not accept output from the executing software of the VAX-11/780 processor. Therefore, the VAX-11/780 software cannot communicate with the console floppy disk or console terminal.

VAX-11/780 Halted — LSI-11 in Program I/O Mode
This mode of operation contains no system functionality and should not be utilized.

VAX-11/780 Halted — LSI-11 in Console Command Mode
In this mode, the full functionality of the console command set is available to the system operator. Through the use of the console command language, the system operator has the capability to:

- Initiate and terminate software being executed by the VAX-11/780 processor.
- Display and modify memory elements including main memory, I/O, general register and process register address space.
- Control the processor clock to provide single step clock modes for use in basic hardware or program development.
- Initiate macro and micro diagnostics.

For further information regarding the console language, a complete listing of the console commands is included in this chapter.

CONSOLE BUS STRUCTURE

Communication between the elements of the console is achieved by three separate bus configurations. The ID (Internal Data) Bus links together the major functional areas of the central processor. The V bus interfaces the LSI-11 microprocessor and its peripheral hardware to the VAX CPU via the CIB (Console Interface Board). The Q bus is utilized by the console, while the LSI-11 is in the console I/O mode, to access the Central Processor's major buses and key control points.

INTERNAL DATA BUS

The Internal Data Bus is a high speed data path between the major functional areas of the CPU. The ID bus may be controlled from the console interface logic in a maintenance mode operation. This allows access to writable control store and internal registers from the console.

When the Console Interface Board generates the ID MAINT signal, it initiates a maintenance operation, allowing the console to assert ID

bus address and control signals (and data, if appropriate). The ID Bus Registers are described in Appendix E.

Q BUS

The Q bus (LSI-11 bus) connects the LSI-11 processor (and its ROM and RAM memories), the console terminal interfaces, and the floppy disk interface to the Console Interface Board, and thus to the VAX CPU. The 16 address signals and 16 data signals share the same bus lines. Fourteen other LSI-11 signal lines are used in the VAX-11/780 configuration for control signals (note that the DMA control lines are not used).

Note that the serial line interface and the floppy disk interface cannot communicate directly with the Console Interface Board, nor can the CIB communicate directly with them. All transfers initiated from the interfaces begin with interrupts to the LSI-11 processor.

V BUS

The V bus consists of eight serial data lines, a load signal line, a clock signal line, and a self test line. Each of the participating VAX CPU modules contains a V bus shift register. The data input lines to the shift register monitor specific test points on the CPU module, as shown in Figure 13-3. The LOAD signal causes the shift register to parallel load from the test points when the VAX CPU is in a stable condition. The clock signal can then be used to read the latched data serially from each of the shift registers into a register on the CIB. The LSI-11 must read the register before clocking in the next serial bit from each of the shift registers.

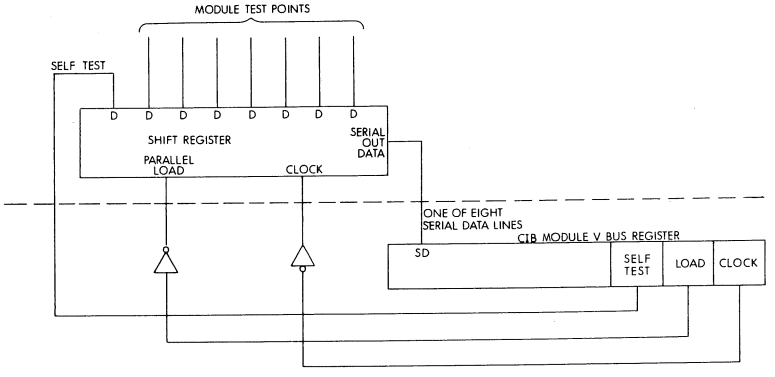


Figure 13-3 V Bus Block Diagram

CONSOLE/VAX-11 INTERACTION

All data transfer operations between the VAX CPU and the console LSI-11 are routed via the TO and FM ID Registers on the CIB. The LSI-11 Console may look at various points in the VAX CPU via the V Bus or it may look at data on the ID Bus. The TO ID Register is a data buffer, serving two functions. First, it may be loaded by the LSI-11 with data from the console terminal, one ASCII character to be read by the VAX-11 microcode. The low order eight bits of the TO ID register contain the ASCII character (RXDB <7:0>). Bits <11:8> specify the console unit at which the data originated. Logical unit 00 is reserved for the operator terminal. Second, the LSI-11 may write to any ID bus address through the TO ID register by executing an ID maintenance cycle.

The terms TO and FR (FROM) are used with respect to the VAX-11 CPU.

The FM ID Register is also a data buffer, serving a dual function. First, it may be loaded by the VAX-11 microcode with data to be passed to the console subsystem. The low order eight bits of the FM ID register contain the ASCII character to be passed to the LSI-11. Bits <11:8> specify one of the logic units in the console subsystem. Second, the LSI-11 may read any ID bus register through the FM ID register by executing an ID maintenance cycle when the VAX CPU is halted.

The TO and FM internal data registers are illustrated in Figure 13-4.

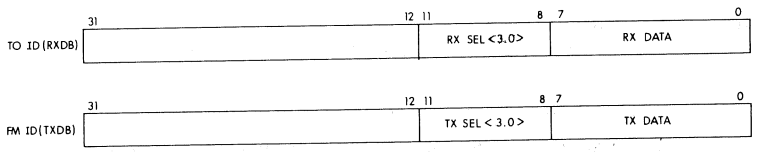


Figure 13-4 TO and FM ID Registers

READ ONLY MEMORY (ROM)

The Console Interface Board contains 4K words of ROM. This ROM contains the core of the LSI-11 console operating system, including the power up routines, the terminal and the floppy drivers. The LSI-11 begins executing instructions in the ROM when power is applied to the system.

VAX-11/780 PROCESSOR CONTROL PANEL

The VAX-11/780 processor control panel consists of four indicator lights, an AUTO RESTART switch, a BOOT switch, and a keylock rotary switch.Figure 13-5 illustrates the VAX-11/780 front panel controls and indicators.

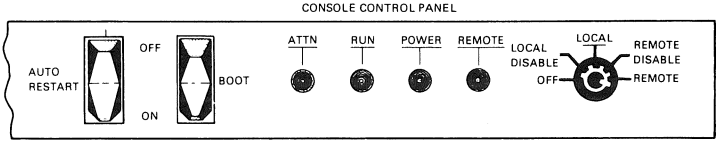


Figure 13-5 VAX-11/780 Front Panel Controls and Indicators

Indicator Lights

- ATTN—This light indicates that you have the attention of the console program.
- RUN—When lit, this light indicates that the processor is running.
- POWER—This light indicates that DC power is present inside the CPU and that the keylock switch is not in the OFF position.
- REMOTE—When lit, indicates that remote diagnostic procedures are being performed on the system.

AUTO RESTART Switch

- AUTO RESTART ON—Restarts system automatically.
- AUTO RESTART OFF—Halts the system and displays the console prompt (>>>) at the console terminal.

BOOT Switch

This switch bootstraps the system using the default bootstrap command procedure (DEFBOO.CMD)

Keylock Rotary Switch

This is a five-position keylock switch.

- OFF—No power to the CPU (except battery back up to the time-of-year clock) or to memory.
- LOCAL DISABLE—Local console terminal cannot issue console commands.
- LOCAL—The CPU responds to console commands and the remote diagnostic line is disabled.
- REMOTE DISABLE—The processor cannot respond to commands from remote line terminals.
- REMOTE—The processor can respond to commands from a remote line terminal and console commands are ignored.

Table 13-1 Console Halt Codes

Code	Meaning
0	Halt or Single Step from console
1	Successful T command
2	Control-P was typed on the console
4	Interrupt stack not valid or unable to read SCB
5	Double bus write error halt
6	Halt instruction with PSL<CM> = 0
7	SCB vector <1:0> = 3
8	SCB vector <1:0> = 2 and no UCS
A	CHMX while on the interrupt stack
B	CHMX SCB vector <1:0> .NE.0
11	Power up and can't find RPB, FPS1 at RE-START/HALT
12	Power up and warm start flag false FPS1 at RE-START/HALT
13	Power up and can't find good 64K of memory
14	Power up and booting, but bad Boot ROM or no ROM
15	Power up and cold start flag set during boot sub-routine
16	Power up halt; Power On Action Switch at HALT position
FF	Micro Verify test failure

CONSOLE COMMAND LANGUAGE

The console commands are English-language entries or their accepted abbreviations entered by the operator at the console terminal. These commands can only be entered when the console is in the console I/O mode of operation. When a valid command is entered, the indicated sequence of events is initiated. When that set of events is completed, another console command can be entered.

The console commands are described in the following paragraphs.

BOOT Command

SYNTAX: BOOT[<DEVICE-NAME>] <CR>

Where the optional <device-name> has the format: DDn, where DD is a two-letter device mnemonic and n is a one-digit unit number.

The BOOT command initiates a VAX-11/780 system bootstrap sequence. The command can support bootstrap operations from or for a set of VAX-11/780 system devices.

When an optional <device name> is not given with the BOOT command, the console performs the boot sequence for the device preset under the default listing by executing the boot program (indirect command file) named DEFBOO.CMD.

When an optional <device name> is given with the command, the console executes boot program DDNBOO.CMD where DDN is the device name given in the command.

Bootstraps (from devices other than the system default device) are performed by indirect command files containing the console commands necessary to boot from the device named.

CONTINUE Command

SYNTAX: CONTINUE<CR>

The CONTINUE command causes the CPU to begin operating system execution at the address currently contained in the CPU program counter (PC). CPU initialization is not performed. The console subsystem enters the program I/O mode after the CONTINUE command is executed.

DEPOSIT Command

SYNTAX: DEPOSIT[<qualifier-list><blank><address><blank><data><CR>

Where the qualifiers are /BYTE, /WORD, /LONG, /QUAD, /NEXT, /VIRTUAL, /PHYSICAL, /IDBUS, /VBUS, /INTERNAL, and /GENERAL.

The DEPOSIT command writes (deposits) <data> into the specified <address>. The address space used depends on the qualifiers specified with the command. If no qualifiers are used, the data and address defaults (entered via the SET DEFAULT command) are used to determine the address space.

Table 13-2 Symbolic Addresses—Used with the DEPOSIT and EXAMINE Commands

SYMBOL	DEFINITION
PSL	Deposits to or examines the processor status longword
PC	Deposits to or examines the program counter (PC)
SP	Deposits to or examines the stack pointer (SP)
+	Deposits to or examines the location immediately following the last location referenced. For physical and virtual references the location referenced is the last address plus n where n= (1 for byte, 2 for word, 4 for longword, 8 for quadword). For all other address spaces, n is equal to 1.
-	Deposits to or examines the location immediately preceding the last location referenced
*	Deposits to or examines the locations last referenced
ampersand	Deposits to or examines the address represented by the last data examined or deposited

EXAMINE Command

SYNTAX: EXAMINE[<qualifier-list>][<blank><address>]<CR>

Where the qualifiers are /BYTE, /WORD, /LONG, /QUAD, /NEXT,

/VIRTUAL, /PHYSICAL, /IDBUS, /VBUS, /INTERNAL, and /GENERAL.

The EXAMINE command reads and displays the contents of the specified <address>. If no <address> is specified, the contents of the address defined by the address default (entered via the SET DEFAULT command) are displayed.

The <address> argument may be one of the symbolic address in the preceding Table.

HALT Command
SYNTAX: HALT<CR>

The HALT command stops the CPU after the CPU completes execution of the instruction that is in process when the HALT command is entered.

INDIRECT (@) Command
SYNTAX: @<filename><CR>
This command causes the console to open the file specified by <filename> and begin executing console commands from the file. Execution continues until one of the following events occurs:

- A WAIT DONE command is read from the file
- The end of the indirect file is reached. The console prints @<EOF> and an <input prompt>
- The operator enters a CTRL C. Execution of the indirect file is aborted

INITIALIZE Command
SYNTAX: INITIALIZE<CR>

This command causes the CPU system to be initialized—set to a specified starting condition.

HELP Command
SYNTAX: HELP<CR>

The console opens and prints the contents of an indirect command file (help file). The help file contains a description of all console commands.

ABBREVIATION HELP Command
SYNTAX: ABBREV.HLP<CR>

The console opens and prints the contents of an indirect command file (abbreviation help file). The abbreviation help file contains a list of abbreviations and rules for the console command language.

ERROR HELP Command
SYNTAX: ERROR.HLP<CR>

The console opens and prints the contents of an indirect command file (error help file). The error help file contains a list of all error messages for the console command language.

LOAD Command
SYNTAX: LOAD[<qualifier-list>]<blank><file-specification><CR>

Where the <qualifier list> is the storage location for the file being loaded; and where the <file specification> is the identity of the file to be loaded.

The LOAD command is used to read or transfer file data from the console floppy disk to main memory, or the the Writable Control Store (WCS). If no qualifier is entered, physical main memory, location 0, is the default location (entered via the SET DEFAULT command) to begin loading. The load defaults are described in Table 13-3.

Table 13-3 LOAD Qualifiers

/START:<address>
The START qualifier is used to specify a starting address for the load. If no START qualifier is given, the console will start loading at Address 0.

/WCS
The WCS is loaded with the specified file.

/PHYSICAL
The physical main memory is loaded with the specified file.

NEXT Command
SYNTAX: NEXT[<blank><count>]<CR>

With the NEXT command, the CPU clock is stepped the number of times indicated by <count>. The type of step performed by the clock is determined by the previous SET STEP command. If a NEXT command is issued while the CPU is in normal clock mode, it defaults to single instruction step mode for the duration <count> of the command.

The console enters program I/O mode immediately prior to performing the STEP command. The console I/O mode is re-entered as soon as the STEP command is complete.

QUAD CLEAR Command
SYNTAX: QCLEAR<blank><physical address><CR>

The QUAD CLEAR command clears the quadword at the specified <address>. The command is used to clear an uncorrectable ECC error.

REPEAT Command
SYNTAX: REPEAT<console command><CR>

With a REPEAT command, the specified console command is repeatedly executed until terminated by a CTRL C. Any console command, except the REPEAT command, may be specified.

SET DEFAULT Command
SYNTAX: SET;,blank>DEFAULT[<blank><default option>]<CR>

The SET DEFAULT command is used to set the defaults for address, data length, and radix. The <default options> and their definitions are contained in Table 13-4. The console applies the defaults from those set under this command when a console command does not contain a qualifier.

Table 13-4 SET DEFAULT Command Options

Address defaults (Set the default to the specified address):	Virtual, Physical, General, Internal, ID bus, V bus
Data defaults (Set the default to the specified data length):	Byte, Word, Long, Quad
Radix defaults (Set the default for the terminal I/O to the specified radix):	Hex, Octal

SET STEP Command
SYNTAX: SET<blank>STEP[<blank><step option>]>CR>

The SET STEP command sets the CPU clock mode. The CPU clock modes (<step options>) are defined is Table 13-5.

Table 13-5 SET STEP Command Options

OPTION GROUP	ITEM	SPECIFICATION
STEP	Instruction	Sets the CPU clock

	to the single instruction step mode
Bus	Sets the CPU clock to the single Synchronous Backplane Interconnect (SBI) cycle step mode
State	Sets the CPU clock to the single SBI time state step mode

SET TERMINAL FILL Command
SYNTAX: SET<blank>TERMINAL FILL<count><CR>

The SET TERMINAL FILL command sets the number (<count>) of blanks that are transmitted to the console terminal after a <CR> or a line feed (<LF>).

SET TERMINAL PROGRAM Command
SYNTAX: SET TERMINAL PROGRAM<CR>

The SET TERMINAL PROGRAM command places the console terminal in the console I/O mode of operation.

SET CLOCK Command
SYNTAX: SET<blank>CLOCK[<blank>(SLOW! FAST! NORMAL!)]<CR>

The SET CLOCK command sets the CPU clock to the frequency specified by the command argument (SLOW, FAST, NORMAL).

SET SOMM Command
SYNTAX: SET<blank>SOMM<CR>

The SET SOMM command sets the Stop On Microbreak Match (SOMM) enable (on the console interface board). The CPU clock is stopped if SOMM is enabled and the contents of the microbreak match register match the contents of the CPU micro-PC.

SET RELOCATION Command
SYNTAX: SET RELOCATION:<data><CR>

The SET RELOCATION command deposits data into the console's relocation register. The value <data> set in the relocation register is added to the effective address of all virtual and physical memory EX-

AMINEs and DEPOSITs.

SHOW Command

SYNTAX: SHOW<CR>

With the SHOW command, the console terminal displays the following:

- The default settings for data length, address type and radix, and data inputs and outputs
- The terminal fill character count
- The CPU status including the run halt state and current clock mode setting

START Command(s)

There are two formats used with the START command:

SYNTAX: START[<blank><address>]<CR>
or
START/WCS<blank><address><CR>

The START command of the first format performs the equivalent of the following sequence of console commands:

1. Initialize the CPU
2. Deposit the <address> data in the PC
3. Issue a continue instruction to the CPU

The START command of the second format performs the equivalent of the following sequence of commands:

1. Deposit the <address> data in the micro-PC
2. Start the CPU clock in the normal mode of operation

TEST Command

TEST[/com]<CR>

The TEST command enables the microdiagnostic monitor program. Microdiagnostic execution begins immediately if no /com qualifier is entered. If microdiagnostic testing is completed successfully (i.e., no errors detected), the console I/O mode is re-entered. A microdiagnostic floppy must be loaded in the RX01.

With a [/com] qualifier, the microdiagnostic monitor enters its command mode (MIC>) and waits for an operator command.

UNJAM Command

SYNTAX: UNJAM<CR>

The UNJAM command initiates a clearing (UNJAM) of the SBI.

WAIT Command

SYNTAX: WAIT<blank>DONE<CR>

The WAIT command is executed from an indirect command file. When

executed, further execution of the command file is suspended until one of the following occurs:

- A DONE signal is received from a program running in the CPU. On receipt of DONE, the console resumes execution of the command file.
- The console prints <@EXIT> and aborts execution of the remainder of the console command file if the CPU halts and a DONE signal is not received.
- The operator enters a CTRL C. The console aborts execution of the remainder of the command file.

CONSOLE ERROR MESSAGES

This section lists all console error messages and defines their format and meaning. All console error messages are prefixed by a question mark, to distinguish them from informational messages. Where user interaction is required, the necessary steps appear in parentheses following the respective error description.

Syntactic Errors

?'<TEXT-STRING>' IS INCOMPLETE	The <TEXT-STRING> is not a complete console command.
?'<TEXT-STRING>' IS INCORRECT	The <TEXT-STRING> is not recognized as a valid command.
? FILE NAME ERR	A <FILENAME> given with a command cannot be translated to RAD50. (<FILENAME> is invalid)
?IND-COM ERR	The console detected an error in the format of an indirect command file. Possible errors are: 1) More than 80 characters in an indirect command line or 2) An indirect command line did not end with a CARRIAGE-RETURN and LINE FEED.

Command Generated Errors

?FILE NOT FOUND	A <FILENAME> given with a 'LOAD' or '@' command does not match any file on the currently loaded floppy disk. This error can also be generated by a 'HELP', 'BOOT' or an attempted WCS load if HELP FILE, BOOT FILE or WCS FILE is missing from Floppy.
?NO CPU RESPONSE	The console timed out waiting for a response from the CPU. (Retry, indicates pos-

	sible CPU-related hardware fault)
?CPU NOT IN CONSOLE WAIT LOOP,COMMAND ABORTED	A console command requiring assistance from the CPU was issued while the CPU was not in the console service loop. (HALT CPU, re-issue command)
?CPU CLOCK STOPPED,COMMAND ABORTED	A console command that requires the CPU clock to be running was issued with the clock stopped. (Clear step mode; re-issue command)
CANT DISABLE BOTH FLOPPY's, FUNCTION ABORTED	An attempt was made to disable both the remote and local floppy.

Micro-Routine Errors

The console uses various micro-code routines in the CPU's control store to perform console functions. The following errors are generated by micro-routine failures:

?MIC-ERR ON FUNCTION	A micro-error occurred in the CPU while servicing a console request. SBI error registers are dumped after this message is printed. (Action dependent upon error)
?INT-REG ERR	A micro-error occurred while attempting to reference a CPU internal (processor) register. An illegal address will cause this error.
?MICRO-ERROR, CODE=X	An unrecognized micro-error occurred. The code returned by the CPU is not in the range of recognized error codes. 'X' is the code returned by the CPU.
?MEM-MAN FAULT, CODE=XX	A virtual examine or deposit caused an error in the memory management micro-routine. 'XX' is a one byte error code returned by the routine, with the following bit assignments: Bit 0 = Length violation (bits numbered from right) Bit 1 = Fault was on a PTE reference Bit 2 = Write or modify intent Bit 3 = Access violation Bits 4 through 7 should be ignored

CPU Fault Generated Error Messages

?INT-STACK INVALID	The CPU halted because the interrupt stack was marked invalid.
?CPU DOUBLE-ERR HALT	A machine check occurred before a previous machine check had been handled, causing the CPU to execute a 'Double Error' Halt. (Examine ID Registers 30-3F (hex); contents will aid in locating cause of machine check).
?ILL I/E VECTOR	The CPU detected an illegal Interrupt/Exception vector.
?NO USR WCS	CPU detected an Interrupt/Exception vector to user WCS and no user WCS exists.
?CHM ERR	A change mode instruction was attempted from the interrupt stack.
INT PENDING	This is not actually an error, but indicates that an error was pending at the time that a console-requested halt was performed. (Continue CPU to clear interrupt).
?MICRO-MACHINE TIME OUT	Indicates that the VAX-11/780 micro-machine has failed to strobe interrupts within the max time period allowed.

Messages Generated by Floppy Errors

?FLOPPY ERROR,CODE=X	The console Floppy driver detected an error. Codes are as follows: (Codes always printed in HEX Radix). CODE 1-Floppy hardware error. (CRC, Parity, etc.) CODE 2-File not found. CODE 3-Floppy driver queue overfull. CODE 4-Console software requested an illegal sector number.
?FLOPPY NOT READY	The console floppy drive failed to become ready when booting. (Retry)
?NO BOOT ON FLOPPY	Console attempted to boot from a floppy that does not contain a valid boot block. (Change floppy disk)
?FLOPPY ERROR ON	A floppy error was detected while attempt-

BOOT	ing a console boot. (Retry)

Messages Related to Version Compatibility

?WARNING-WCS & FPLA VER MISMATCH	The microcode in WCS is not compatible with FPLA. This message is printed on each ISP START or CONTINUE, but no other action taken by console.
?FATAL-WCS & PCS VER MISMATCH	The microcode in PCS is not compatible with that in WCS. ISP START and CONTINUE are disabled by console.
?REMOTE ACCESS NOT SUPPORTED	Printed when console mode switch enters a 'REMOTE' position, and the remote support software routines are not included in the console.

Console Generated Errors

?TRAP-4, RESTARTING CONSOLE	The console took a time-out trap. Console will restart.
?UNEXPECTED TRAP MOUNT CONSOLE FLOPPY, THEN TYPE ↑C	Console trapped to an unused vector. Console reboots when ↑C typed.
?Q-BLKD	Console's terminal output queue is blocked. Console will reboot.

BOOTING THE VAX-11/780 SYSTEM

Initializing or booting the VAX-11/780 system can be viewed from two different perspectives. First, there are the actual steps that the system manager must perform in order to boot the system. And second, there are the actual events that occur within the system during the boot process.

To bootstrap a VAX-11/780 system, the system manager first invokes the console program. The remaining steps for a typical system boot are as follows.

1. Check that the console floppy is loaded into the floppy diskette drive.
2. Set the AUTO RESTART to the OFF position.
3. Turn the rotary key to the LOCAL position. When the power is turned on, the console floppy is booted and causes the system to prompt (>>>). If the power is already turned on, press CTRL/P to get the prompt. Type REBOOT to reboot the console.

VAX-11/780 Console Subsystem Action on Boot

When the system manager executes a boot command procedure on

the console floppy, the console subsystem performs the following steps to boot the system.

1. Halts the processor.
2. Unjams the SBI.
3. Initializes the processor.
4. Deposits the address of a SCB which is stored in ROM.
5. Loads general registers with the inputs required by the primary bootstrap program.

R0	Boot device type code
R1	Boot device adapters TR#
R2	UNIBUS address of boot device's CSR
R3	Boot device unit number
R5	Software boot control flags
FP	0 (signals ROM program that no machine check is expected)

The console subsystem then starts the program stored in ROM. The program will:

1. Perform a cursory CPU test and check for a valid memory configuration.
2. Locate a 64 KB block of contiguous memory that contains no uncorrectable errors.
3. Load base address + ↑X200 of 64 KB of good memory into the SP.
4. Set the cold start flag.

Next, the console subsystem loads the primary bootstrap at the address specified in the SP. Finally, control is transferred to the primary bootstrap (console enters program mode).

VAX-11/782 Boot Procedure

The VAX-11/782 primary processor boot command procedure is very similar to that of the VAX-11/780. The differences are as follows:

1. Only the MA780 shared memory is used. MS780 (local) memory is ignored.
2. The MA780 memory controller does not contain a boot ROM. The command procedure assumes that the first 64 KB of good memory starts at physical address 0, and places a 0 in the SP. The memory configuration registers are also initialized by the command procedure. The RPB (reboot parameter block) is located at physical address 0 and the primary bootstrap code, VMB.EXE, is loaded starting at physical address 200.

The attached processor is booted differently from a single processor VAX-11/780. The boot command procedure initializes the memory registers (same as the primary) and then starts executing a self-branch instruction in the RPB (located at physical address 0). After the primary is booted, a DCL command is executed that causes the primary to load the multiprocessing code. This command also modifies the self-branch instruction in the RPB to point to the attached processor's multiprocessing initialization code.

VAX-11/780 Console Subsystem Action on a Warm Start

The console subsystem will attempt to restart the CPU under the following conditions:

1. A power restoration with valid memory and the AUTO RESTART switch is in the ON position.
2. A Halt condition (HALT instruction executed or machine error halt condition) and the AUTO RESTART switch is in the ON position.

The console takes the following actions to restart the processor.

1. Loads the console program from the console floppy.
2. Loads the microcode from the console floppy.
3. Tests the AUTO RESTART switch: if OFF, it issues a console prompt (>>>); if ON, it continues to the next step.

The console program loads R12 (AP) with the Halt code which identifies the type of Halt. The PC and PSL are loaded at the time of the Halt into R10 and R11. The console then invokes the command file RESTAR.CMD on the console floppy. This program performs the following steps:

1. Ensures that the processor is halted and initialized, then deposits the ROM address of the SCB in the SCB base processor register.
2. Clears unused general registers and deposits a 3 in R1 (the boot device adapter's TR#).
3. Deposits a 0 in the FP (signals to ROM program that no machine check is expected).
4. Starts ROM program that searches memory for a valid RPB. When it is found, the restart routine is executed. Its address is located at the second longword of the RPB.

If the AUTO RESTART switch is set to AUTO RESTART ON, the console subsystem searches through physical memory for a valid RPB, shown as follows:

physical address of the RPB	0:
physical address of the restart routine	4:
checksum of ↑X1F longwords of restart routine	8:

Bit 0; ! warm start flag C:

Parameter Block, First Four Longwords

A valid RPB is defined as a block of four longwords, starting on a page boundary. The first longword points to itself. The second is a pointer to the address of the restart code and must not be zero. The third longword contains the checksum (sum, throwing away carries) of the 31 longwords pointed to by the second longword. The console subsystem starts at address zero and searches all available memory for a valid RPB. If it doesn't find one it attempts to reboot the system. If it does find an RPB, it examines bit 0 of the fourth longword of the RPB. If this bit is 1, it will halt. If this bit is 0, it sets it and then:

- Loads the stack pointer (SP) with the address of the RPB plus ↑X200.
- Loads the argument pointer (AP) with a value that indicates the cause of the restart. If the restart is occurring because of power restoration or reset switch activation, the AP is loaded with a 3. If it is because of a CPU Halt, it is loaded with one of the codes specified in Table 7-1.
- Starts execution of the restart routine, whose address is located at the second longword of the RPB.

VAX-11/782 Console Subsystem Action on a Warm Start

The action taken by the VAX-11/782 primary processor is similar to a single processor VAX-11/780. The key difference being the absence of an MA780 memory ROM to locate the RPB. Therefore, the command file RESTAR.CMD loads 200 into the SP (the address of the RPB plus ↑X200).

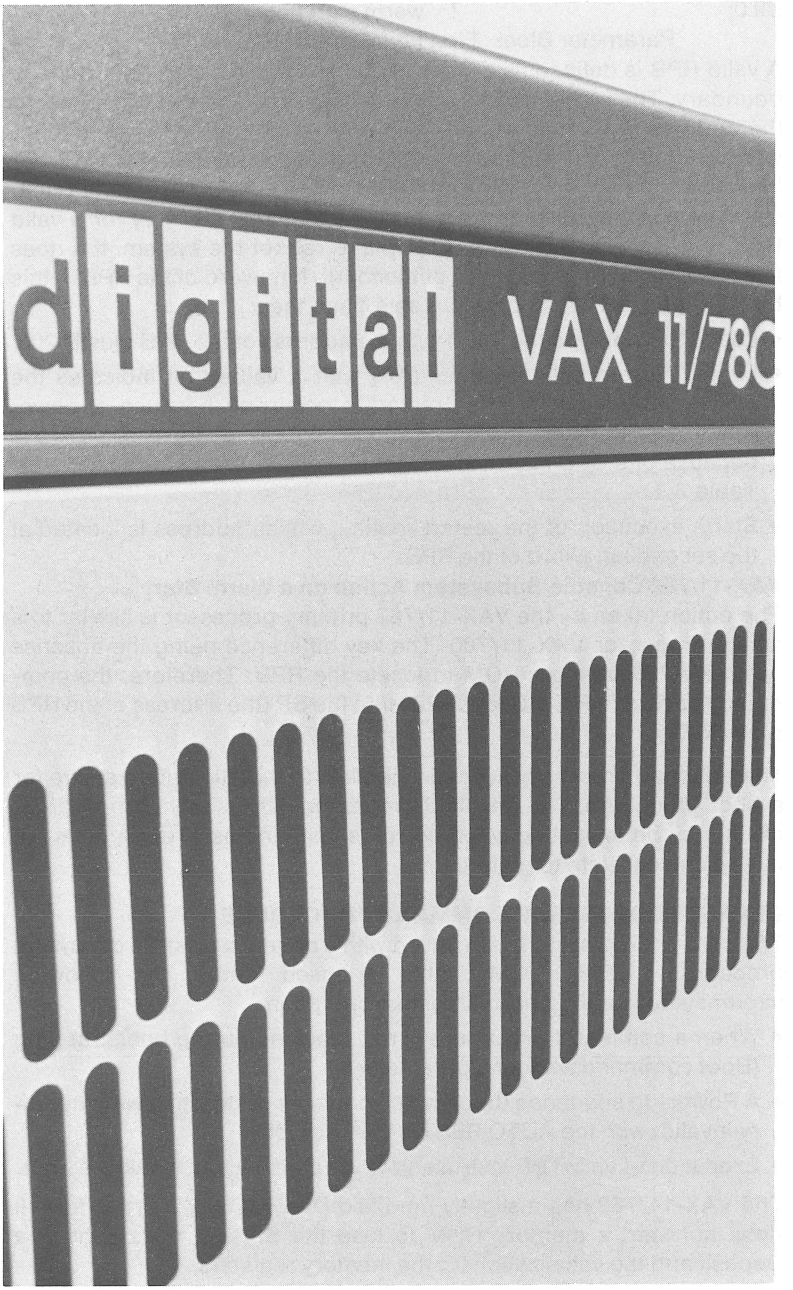
The attached processor action is identical to that described above for a boot procedure. The attached processor returns any current process it may be executing when power is lost, and as a result, does not have a current state to restore.

DEFAULT BOOTSTRAP COMMAND PROCEDURE

The VAX-11/780 system is installed with a default bootstrap command procedure, DEFBOO.CMD, that is used under the following circumstances during normal system operation:

- When a command procedure is not specified during bootstrapping (Boot command without a parameter).
- A Power-up sequence (boot switch or power restoration with memory invalid) with the AUTO RESTART switch ON.
- Execution of an MTPR instruction to the console that invokes a boot.

The VAX-11/782 has a slightly modified DEFBOO.CMD procedure. It does not start a memory ROM to load the SP, but does contain a Deposit and the initialization for the memory registers.


digital
VAX 11/78

CHAPTER 14

VAX-11/780 CENTRAL PROCESSOR

FEATURES	BENEFITS
32-bit microprogrammed processor	Provides the user with high-performance data throughput
Memory management hardware	Allows the user to directly address up to four billion bytes of virtual address space using a smaller physical memory
8 KB direct mapped memory cache	Significantly improves memory access time, increasing overall performance
PDP-11 compatibility mode	Gives the PDP-11 user an easy migration path to the VAX/VMS architecture
User control store (optional)	Increases throughput by allowing the user to create routines in microcode tailored to specific applications
Optional Floating Point Accelerator	Decreases instruction execution time of floating point arithmetic and some integer arithmetic operations

INTRODUCTION

The VAX-11/780 Central Processing Unit (CPU) is the hardware responsible for performing the logic and arithmetic operations requested of the computer system. The processor is a high-performance, microprogrammed computer that executes a large set of variable-length instructions in native mode, and non-privileged PDP-11 instructions in compatibility mode.

The CPU maintains 32-bit addressing and data capability, thereby allowing it direct access to four billion bytes of virtual address space (2^{32}). That is, the CPU references a location in terms of a 32-bit virtual address. This address is termed virtual because it is not the actual address in physical memory. The processor's memory management hardware translates a virtual address to a physical address under operating system control.

The processor provides 16 32-bit registers that can be used for temporary storage, as accumulators, index registers, and base registers. Four of these registers have special significance: the Program Counter, and three registers that are used to provide an extensive CALL facility. The processor offers a variety of addressing modes that use the general registers to identify instruction operand locations, including an indexed addressing mode that provides true post-indexing capability.

The native instruction set is highly bit efficient. It includes integral decimal, character string, and floating point instructions, as well as integer, logical, and bit field instructions. Instructions and data are variable length and can start at any arbitrary byte boundary or, in the case of bit fields, at any arbitrary bit in memory. Floating point instruction execution can be enhanced by an optional floating point accelerator.

The processor's instruction set is defined by the microcode loaded into the programmable read-only memory (control store).

The VAX-11/780 processor includes the following functional hardware components:

- 8 KB two-way set associative memory cache
- 8 byte prefetch instruction buffer
- 128 entry address translation buffer
- 24 KB writable diagnostic control store (WDCS)
- Time-of-year clock
- Programmable realtime clock
- Integral memory management
- Optional floating point accelerator (FPA)
- Optional KE780 Extended Floating Point Data Types
- Optional customer-writable control store (WCS)

This chapter is divided into three sections. The first section discusses processor hardware, functionality and example processor operation. The second section discusses the programming characteristics of the processing system from the user's point of view. And the last section looks at the processing system, but from an operating system viewpoint.

HARDWARE ELEMENTS

The VAX-11/780 CPU is a fast, high-performance, 32-bit microprogrammed computer. The CPU derives its speed and performance from the fact that it can handle several independent functions simultaneously.

The CPU can process both 32-bit data and addresses while maintaining the ability to manipulate:

- Bits (up to 32)
- Bytes
- Words
- Longwords
- Quadwords
- 32 bit floating point (single precision)
- 64 bit floating point (double precision)
- Packed decimal (up to 31 digits)
- Character strings (up to 64 KB)
- Queues

Control Store

The control store is a read-only memory containing 4 K 96-bit microwords plus 3 parity bits per microword. The control store contains the program that describes the operation and sequencing of the central processing unit. It also contains the native, compatibility, and floating point instruction sets. The control store contains a 96 bit buffer, enabling it to execute one microword while simultaneously fetching the next.

Data Paths

The data path subsystem consists of four independent and parallel sections used to process addresses and data specified by the instruction set. The arithmetic section is used to perform both arithmetic and logical operations on data and addresses. The exponent and sign section is used for fast exponent processing of floating point instructions. The data shift and rotate section packs and unpacks floating point and decimal string data. And finally, the address section calculates virtual addresses for the translation buffer.

8 KB Two-Way Set Associative Memory Cache

The memory cache is the primary cache system for all data coming from memory, including addresses, address translations, and instructions. The memory cache is an 8 KB, two-way set associative cache.

The memory cache also reduces the average time the processor waits to receive main memory data by reading eight bytes at a time from main memory, and transferring four bytes to the CPU data paths, or instruction buffer. Since the remaining four bytes are already available, the memory cache also provides pre-fetching. The cache memo-

ry system carries byte parity for both data and addresses for increased integrity. Cache locations are allocated when data is read from memory. When both of the possible locations for a particular datum are already filled, one of the previously cached data is randomly replaced.

Address Translation Buffer
The address translation buffer is a cache of likely-to-be-used physical address translations. It significantly reduces the amount of time spent by the CPU on the repetitive task of dynamic address translation. The cache contains 128 virtual-to-physical page address translations which are divided into equal sections: 64 system space page translations and 64 process space page translations. Each of these sections is two-way associative. There is byte parity on each entry for increased integrity.

8 Byte Prefetch Instruction Buffer
The 8 byte instruction buffer improves CPU performance by prefetching data in the instruction stream. The control logic continuously fetches data from memory to keep the 8 byte buffer full. It effectively eliminates the time spent by the CPU waiting for two memory cycles where bytes of the instruction stream cross 32-bit longword boundaries. In addition, the instruction buffer processes operand specifiers in advance of execution and subsequently routes them to the CPU.

24 KB Writable Diagnostic Control Store (WDCS)
The writable diagnostic control store consists of 2048 96-bit (24 KB) control words plus three parity bits per control word. These locations are used to contain basic instruction microcode, diagnostic microcode, and reserved space to accommodate future additions or improvements made by DIGITAL to the instruction set.

Processor Clocks
The VAX-11/780 processor contains a programmable realtime clock and a time-of-year clock. The interval or realtime clock was designed to permit the measurement of finely resolved variable intervals which are identified by interrupts (i.e., scheduling, diagnostics, etc.). The realtime clock is based upon a crystal oscillator with an accuracy of 0.01%, and a resolution of one μsec. The time-of-year clock is used by software to perform various timekeeping functions. Its major function is to provide the correct time to the system after power failure or other system interruptions.

Optional Floating Point Accelerator
The floating point accelerator is an optional high-speed processor

extension. When included in the processor, the floating point accelerator executes the addition, subtraction, multiplication, and division instructions that operate on single- and double-precision floating point operands, including the special EMOD and POLY instructions in both single- and double-precision formats. Additionally, the floating point accelerator enhances the performance of 32-bit integer multiply instructions.

The processor does not have to include the floating point accelerator to execute floating point operand instructions. The floating point accelerator can be added or removed without changing any existing software.

When the floating point accelerator is included in the processor, a floating point operand register-to-register add instruction takes as little as 800 nanoseconds to execute. A register-to-register multiply instruction takes as little as one μsec. The inner loop of the POLY instruction takes approximately one μsec per degree of polynomial.

Optional KE780 Extended Floating Point Data Types

For applications which require extended decimal digit precision, the optional G-floating (double precision) and H-floating (quadruple precision) data points are available to provide up to 33 decimal digit accuracy.

Optional 24 KB Customer Writable Control Store (WCS)

The user writable control store consists of 2048 96-bit (24 KB) control words plus three parity bits per control word (if the KE780 option is not present). These locations are optionally available to the customer for augmenting the speed and power of the basic machine with customized functions.

Figure 14-1 illustrates the central processing unit.

PROCESSOR OPERATION

For those interested in the hardware operations and interfaces of the VAX-11/780 CPU elements, the execution of a sample piece of code is described below. A FORTRAN IV DO LOOP is first expanded into its VAX-11 MACRO equivalent, and then into the VAX-11/780 machine specific implementation. For the purposes of this description, virtual to physical translation values, although valid, have been assumed.

Example: FORTRAN IV DO LOOP

```
            J = 0
            Do 100 I = 1, 10
100         J = J + N(I)
```

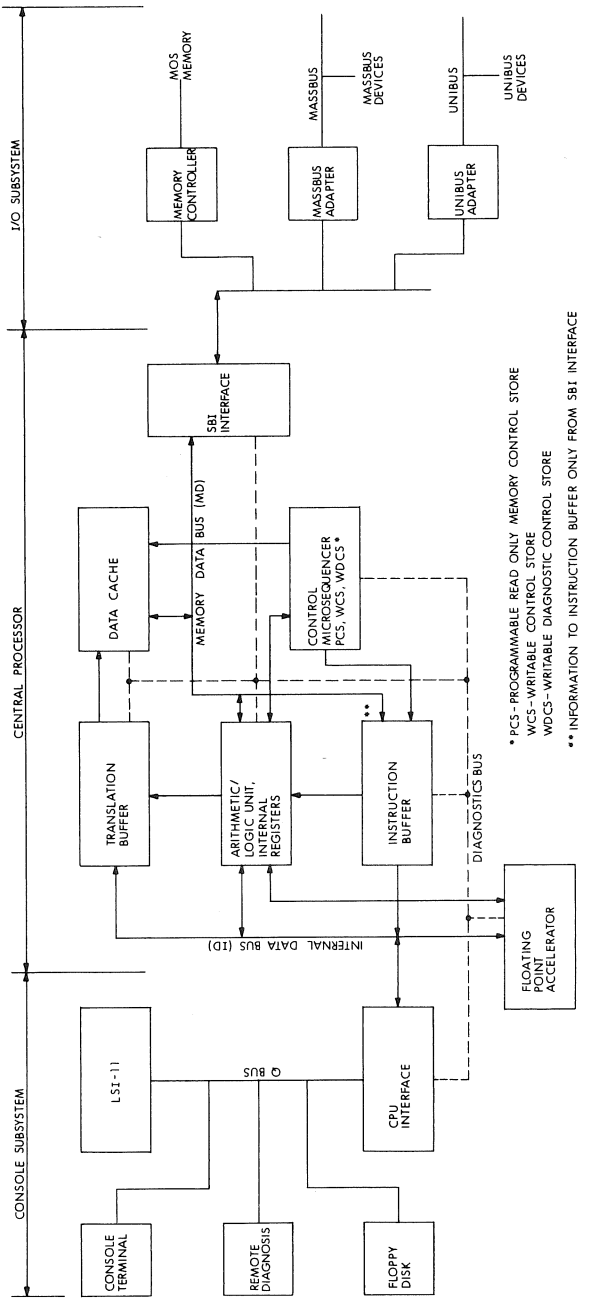


Figure 14-1 The Central Processing Unit

VAX MACRO EXPANSION

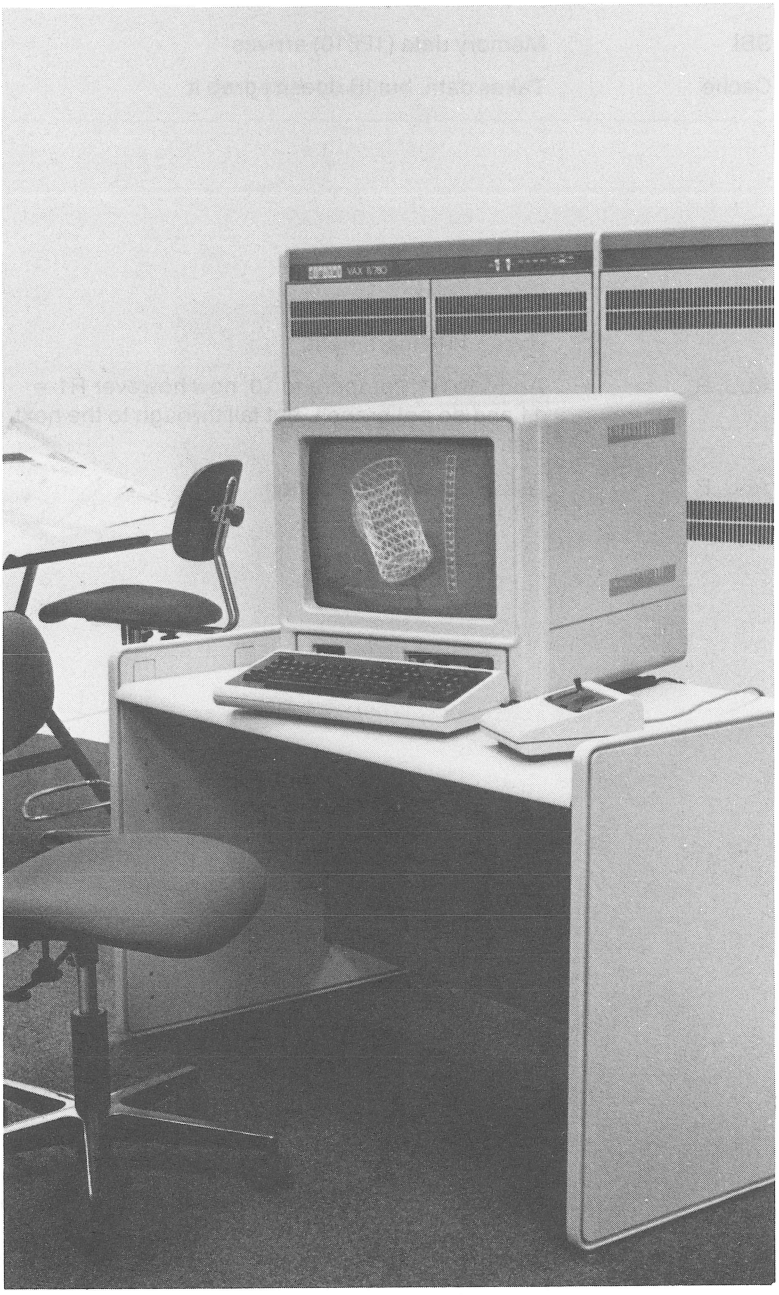
```
1000        CLRL      R0
1002        MOVL      #1, R1
1005
1$:         ADDL2     N<R1>, R0
100B        AOBLEQ    #10, R1, 1$
1FFC
N:          .BLKL     11
```

CENTRAL PROCESSOR IMPLEMENTATION

CPU Component	Operation
ALU, R	1000 → PC
TB	Translate virtual 1000 to physical 1F600
Cache	Does Cache presently contain address 1F600? (NO) therefore,
SBI	Fetch 1F600-1F607 from memory
Cache	Store address range 1F600-1F607 and corresponding contents in Data Cache
(IB)	Obtain instructions from physical addresses 1F600-1F603
ALU, R	Ask IB for instruction -- (CLRL)
ALU, R	Clear R0
(IB asks Cache for more	IB retrieves physical addresses 1F604-1F607 from Cache
ALU, R	Ask IB for next instruction -- (MOVL)
ALU, R	Ask IB for destination specifier -- (R1)
(IB asks Cache for more)	Cache asks SBI for 1F608
SBI	Asks memory for physical addresses 1F608-1F60F
ALU, R	Store 1 in R1

ALU, R	Ask IB for next instruction -- (ADDL2)
ALU, R	Calculate base address of N (virtual 1FFC)
ALU, R	Adds 4*R1 to address of N to yield virtual address 2000
TB	Look up address of N[1] : physical address A00
Cache	Searches for physical address A00, but finds it not there, therefore,
SBI	The SBI enters a wait mode because it is currently completing the fetch operation of physical addresses 1F608-1F60F
SBI	Finishes the prefetch operation of physical addresses 1F608-1F60F
IB	Grabs 1F608-1F60B
Cache	Gets 1F608-1F60F
SBI	Starts fetch of physical addresses A00-A07
Cache	Gets A00-A07
ALU, R	A00-A03
ALU, R	Asks IB for destination specifier (R6)
(IB asks for 1F60C)	Cache sends 1F60C-1F60F to IB
ALU, R	Add (A00-A03) (i.e., N[1]) to R0
ALU, R	Ask IB for instruction (AOBLEQ)
(IB asks for more data)	Cache asks SBI to get 1F610 from memory
ALU, R	Asks IB for next specifier (R1)
ALU, R	Add 1 to R1, compare to 10, if less than or equal to 10 then branch
ALU, R	Flush (clear) IB, load virtual 1005 into PC
IB	Fetch 1005 from cache (resumption of loop)
ALU, R	Ask IB for next instruction (ADDL2)
.	.
.	.

.	.
SBI	Memory data (1F610) arrives
Cache	Takes data, but IB doesn't grab it
.	.
.	.
.	.
.	.
.	.
	on the 11th increment,
ALU, R	Add 1 to R1, compare to 10, now however R1 = 11 and do not branch, but fall through to the next instruction
ALU, R	Ask IB for next instruction


VAX 8780

CHAPTER 15

SYNCHRONOUS BACKPLANE INTERCONNECT

FEATURES	BENEFITS
200 nanosecond cycle time	Allows throughput rate of up to 13.3 million bytes per second
Distributed arbitration	Signals need travel the length of the SBI only once, providing increased speed
16-level silo	Monitors SBI activity and contains a history of the 16 most recent cycles of bus activity
Maintenance registers	Help determine the cause of bus specific errors

INTRODUCTION

The Synchronous Backplane Interconnect (SBI) is the data path that links the central processor, the memory subsystem, and the hardware adapters provided for the UNIBUS and MASSBUS. The VAX-11/780 bus structure is illustrated in Figure 15-1.

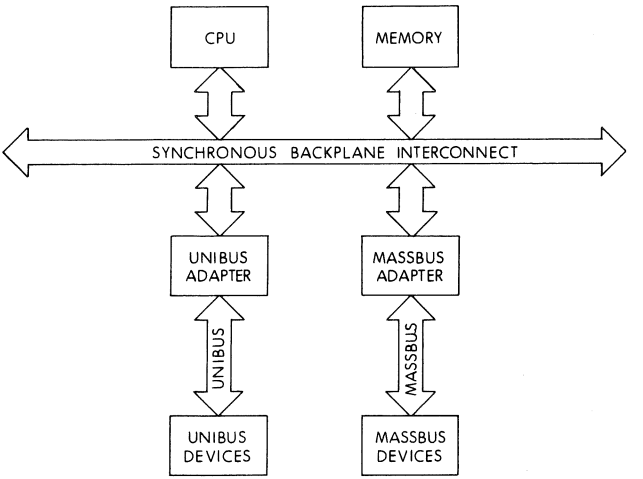


Figure 15-1 Basic Bus Configuration

When interfaced to the SBI, the central processor, memory subsystem, and I/O controllers are known as NEXUSs.

A NEXUS is a physical connection to the SBI and is capable of acting as any of the following:

> Commander — A NEXUS which transmits command and address information.
> Responder — A NEXUS which recognizes command and address information as directed to it and requiring a response.
> Transmitter — A NEXUS which drives the signal lines.
> Receiver — A NEXUS which samples and examines the signal lines.

A NEXUS also performs priority arbitration for its access to the SBI.

A NEXUS may perform more than one function, as illustrated in the two following examples.

When the CPU issues a read command it is a commander because it issues command/address information. At the same time it is a transmitter because it is driving the signal lines. When the device (responder) returns the requested data, the CPU is considered a receiver because it examines the signal lines.

In the case of a memory read exchange, memory is the responder because it recognizes and responds to command/address information. Also, because it examines the signal lines, it is a receiver. When memory returns the requested data by driving the signal lines, it is a transmitter.

All NEXUSs receive every SBI transfer. Logic in each NEXUS determines whether the NEXUS is the designated receiver for a particular transfer.

Data may be exchanged between the following system elements:

- The central processor and memory subsystem
- I/O controllers and memory subsystem
- Central processor and I/O controllers

The communication protocol allows the information path to be time-multiplexed in such a way that up to 32 data exchanges may be in progress simultaneously.

The SBI provides checked, parallel information transfer synchronous with a common system clock. In each clock period or cycle (duration of 200 nsec) interconnect arbitration, information transfer and transfer confirmation may occur. Utilizing the 200 nsec clock period, the SBI achieves a maximum information transfer rate of 13.3 million bytes per second.

SBI STRUCTURE

The SBI is comprised of 84 signal lines as illustrated in Figure 15-2. Its maximum physical length may not exceed 3 meters (9.8 ft). The lines of the SBI are divided into the following functional groups:

- Arbitration
- Information
- Response
- Interrupt
- Control

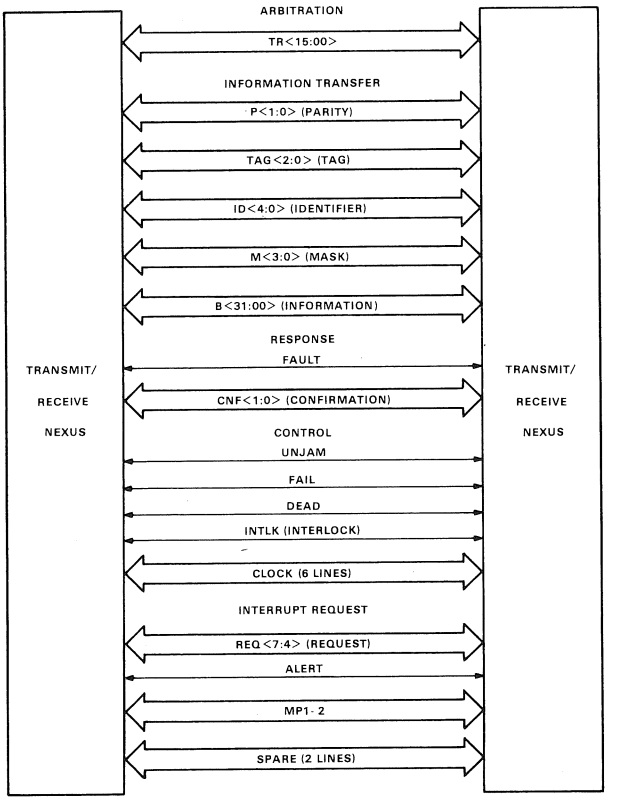


Figure 15-2 SBI Signal Description

Arbitration Lines

There are 16 bus arbitration lines. Each arbitration line, TR (transfer request) <15:01>, is assigned to one NEXUS, thereby establishing a fixed priority access to the information path (refer to Figure 15-3). Access priority increases from TR15 to TR00, where TR00 is reserved for use as a hold signal for the following reasons:

1. NEXUS requires two or three adjacent cycles for a Write type exchange.
2. NEXUS requires two adjacent cycles for an Extended Read exchange.
3. Central processors for Interrupt Summary Read exchanges.
4. TR00 is reserved for an SBI UNJAM operation.

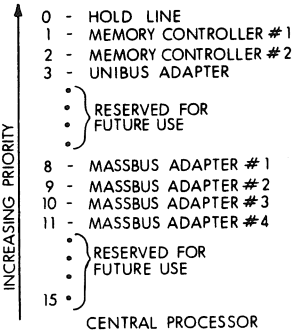


Figure 15-3 SBI Priority Access

To acquire control of the information path, a NEXUS asserts its assigned (transfer request) line at the beginning of a cycle.

At the end of a cycle, the NEXUS examines the state of all transfer request lines of higher priority. If no higher priority NEXUS is arbitrating for control of the SBI, the NEXUS will remove its transfer request and assert information path signals. The lowest priority NEXUS arbitrating for control of the SBI is the central processing unit. The CPU does not require a transfer request signal, because by default it will gain control of the SBI when no higher priority NEXUS is arbitrating.

Information Lines

The information transfer group exchanges command/addresses, data, and interrupt summary information. Each exchange consists of one

to three information transfers.

For write commands, the commander uses two or three successive SBI cycles. The number of successive cycles required depends on whether one or two data words are to be written in the exchange. In the first case, the commander transmits the command/address in the first cycle, and a data word in the second cycle. In the second case, the commander transmits the command/address in the first cycle, data word 1 in the second cycle, and data word 2 in the third cycle.

Read commands are also initiated with a command/address transmitted from the commander. Because data emanates from the responder, the requested data may be delayed by the characteristic access time of the responder. As in a write exchange, the read exchange will transmit data using one or two successive cycles depending on whether one or two data words were requested.

An interrupt summary exchange is in response to a device-generated interrupt to the CPU. The exchange is initiated with an interrupt summary read transfer from the CPU. The exchange is completed two cycles later with an interrupt summary response transfer containing the interrupt information.

The information Transfer Group is subdivided into the following five fields:

Field	Length in Bits
Parity check	2(P<1:0>)
Information Tag	3(TAG<2:0>)
Source/Destination Identity	5(ID<4:0>)
Mask	4(M<3:0>)
Information	32(B<31:00>)

PARITY FIELD

The parity field (P<1:0>) provides even parity for detecting single bit errors in the information transfer group. A transmitting NEXUS generates P0 as parity for the tag. indentifier, and mask fields and P1 as parity for the Information field. The parity field is illustrated in Figure 15-4.

P0 and P1 are generated in such a way that the sum of all logical one bits in the checked field, including the parity bit, is even.

When the SBI is idle, the information transfer path assumes an all-zero state. Therefore, the parity field should always carry an even parity.

TAG FIELD

The tag field (TAG<2:0>) is asserted by a transmitting NEXUS to

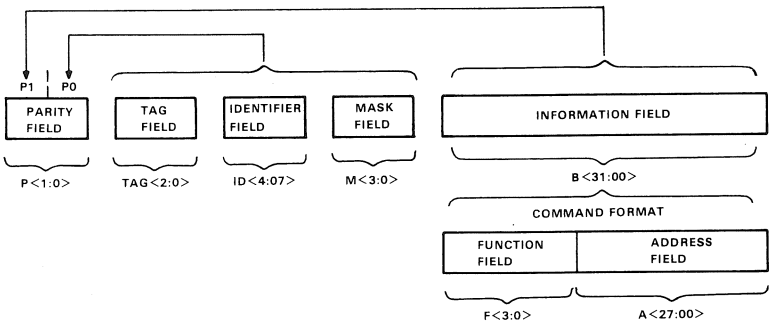


Figure 15-4 Party Field Configuration

indicate the information type being transmitted on the information lines. The tag field determines the interpretation of the ID and B fields. In addition, the tag field, in conjunction with the mask field, further defines special read and write data conditions.

Four tag fields and four reserved fields are defined as:

TAG<2:0>	B<31:00> contents
000	READ DATA
011	COMMAND ADDRESS
101	WRITE DATA
110	INTERRUPT SUMMARY READ

The remaining tag fields, 001, 010, 100, and 111, are reserved.

- Read Data Tag

A tag field content of 000 specifies that the information field B<31:00> contains data requested by a previous read type command. The retrieved data may be one of three types: read data, corrected read data, and read data substitute. The retrieved data type is identified by the mask field M<3:0>. Read data is the normal expected error-free data type, where M<3:0> = 0000. Corrected read data (CRD) is represented by M<3:0> = 0001, and read data substitute is represented by M<3:0> = 0010. The recipient of the read data is designated by ID<4:0>. The read data tag formats are illustrated in Figure 15-5.

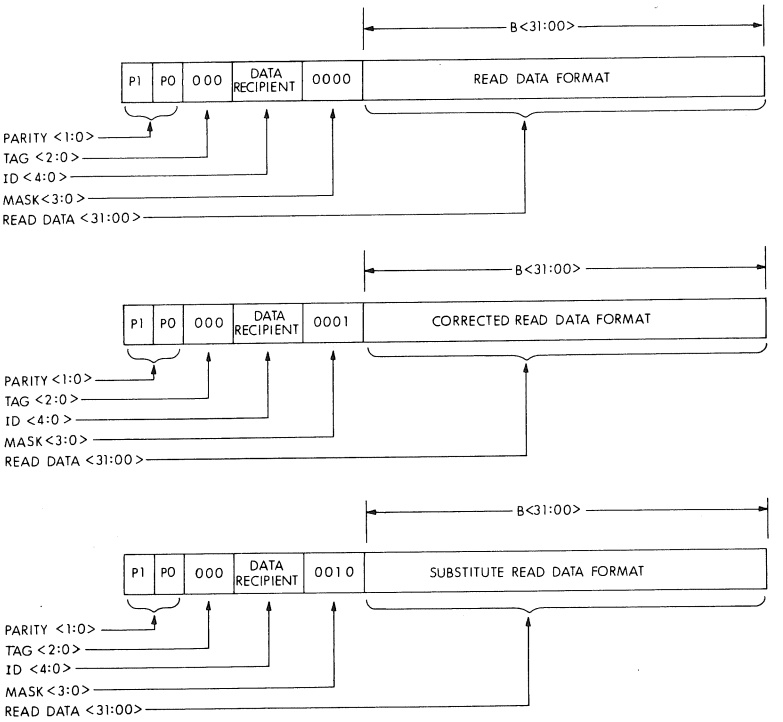


Figure 15-5 Read Data Tag Formats

- Command/Address Tag

A tag field content of 011 specifies that the data lines contain a command/address word, and that ID<4:0> is a unique code identifying the logical source (commander) of that command. As illustrated in Figure 15-6, B<31:00> is divided into a function field and an address field to specify the command and its associated address.

The ID field code represents the logical source of the data in a write command, and the address field specifies the address of where the data is to be written. For a read command, the ID field represents the logical destination of the data at the location specified in the address field.

The 28 bits of the address field define a 268,435,456 longword address space (1,073,741,824 bytes) which is divided into two sections. Addresses 0-7FFFFFF (hex) (A27=0) are reserved for primary memory. Addresses 8000000 (hex) - FFFFFFF (hex) (A27 = 1) are reserved for

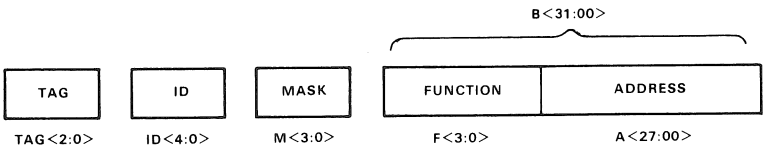


Figure 15-6 Command/Address Format

device control registers. Primary memory begins at address 0. The address space is dense and consists only of storage elements. Figure 15-7 illustrates the VAX-11/780 physical address space. Note that both physical and SBI addresses are provided.

The user has access to the physical address space via the 30-bit physical byte address. However, because NEXUS registers are accessible only as longword addresses, system hardware converts the physical byte address (30 bits) to the SBI longword address (28 bits). This translation is described in Figure 15-8.

The low order two bits of the physical to SBI translation are not lost, but are represented by the mask field adjoining the SBI command address format.

The control address space is sparse with address assignments based on device type. Each NEXUS is assigned a 2048, 32-bit longword address space for control and status. The addresses assigned are determined by the TR number as shown in Figure 15-9.

The command/address tag formats are illustrated in Figure 15-10.

- Write Data Tag

A tag field content of 101 specifies that B<31:00> contains the write data for the location specified in the address field of the previous write command. The write data will be asserted on B<31:00> in the SBI cycle immediately following the command/address cycle. The ID field transmitted is that of the commander. Figure 15-11 illustrates the write data tag format.

- Interrupt Summary Tag

A tag field content of 110 defines B<31:00> as the interrupt level mask for an interrupt summary read command. The level mask (B<07:04>) is used to indicate the interrupt level being serviced as the result of an interrupt request. In this case, the ID field identifies the

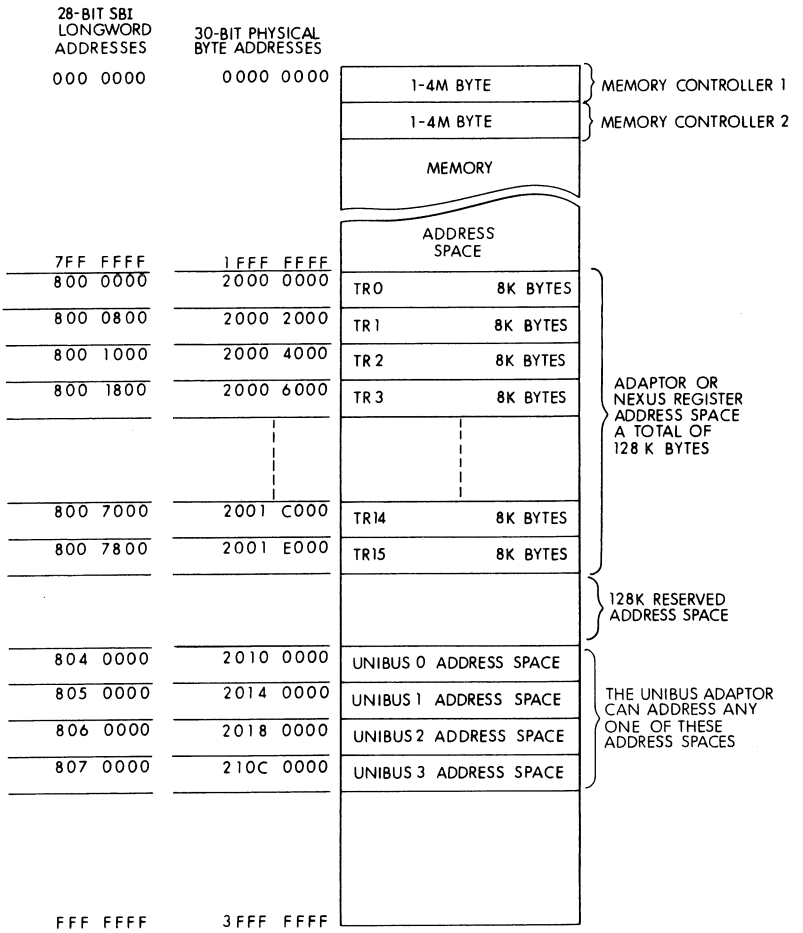


Figure 15-7 SBI Physical Address Space

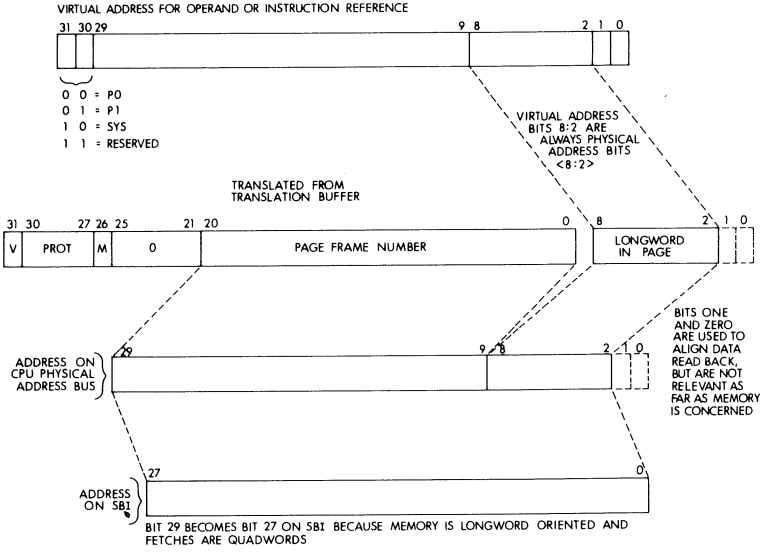


Figure 15-8 Physical to SBI Address Translation

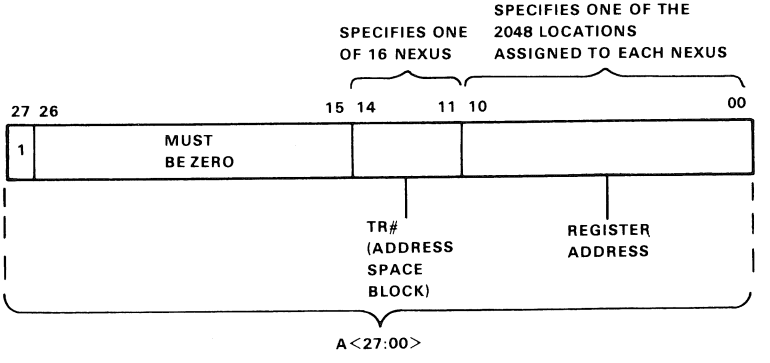


Figure 15-9 Control Address Space Assignment

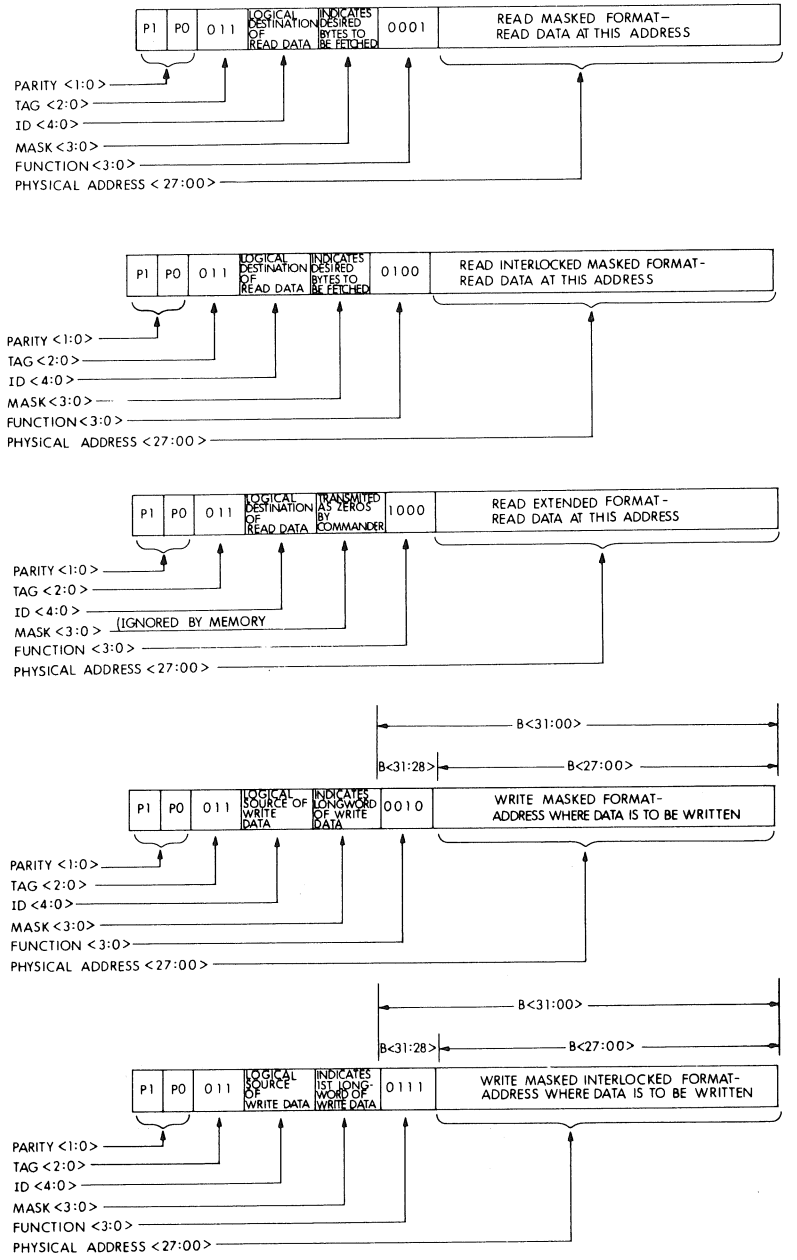


Figure 15-10 Command/Address Tag Formats

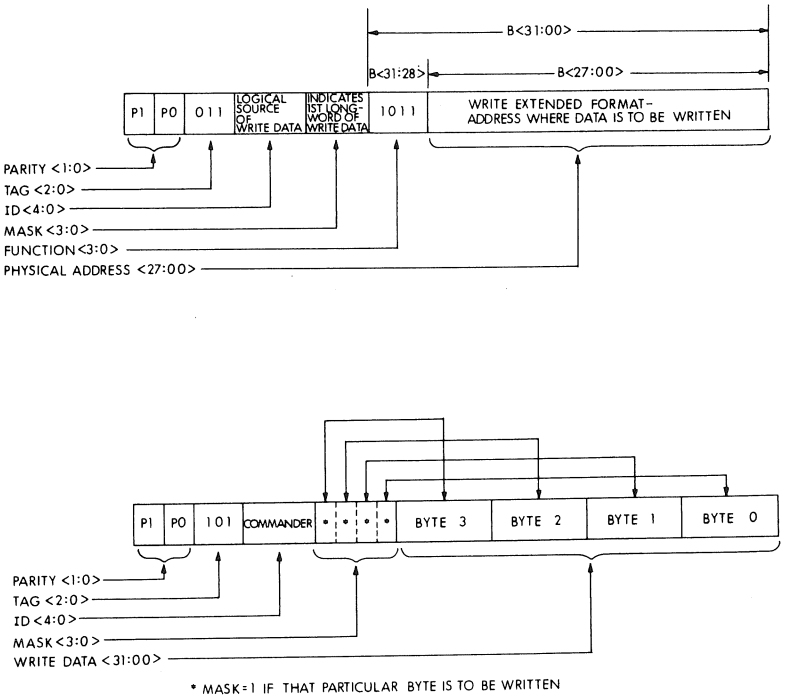


Figure 15-11 Write Data Tag Format

commander, which is the CPU. Although unused, M<3:0> is to be transmitted as zero.

The interrupt sequence consists of two exchanges:

The first exchange indicates the interrupt level being serviced. The interrupt level is determined as follows:

If the I/O controller asserts interrupt

If CPU strobes the interrupt, and if level 7 is the current level, interrupt code is called which performs the Interrupt Service Request.

The second exchange is the response, where the device requesting the interrupt identifies itself. From the identity of the device and the interrupt level, the starting address of the service routine can be determined.

Figure 15-12 illustrates the interrupt summary tag format.

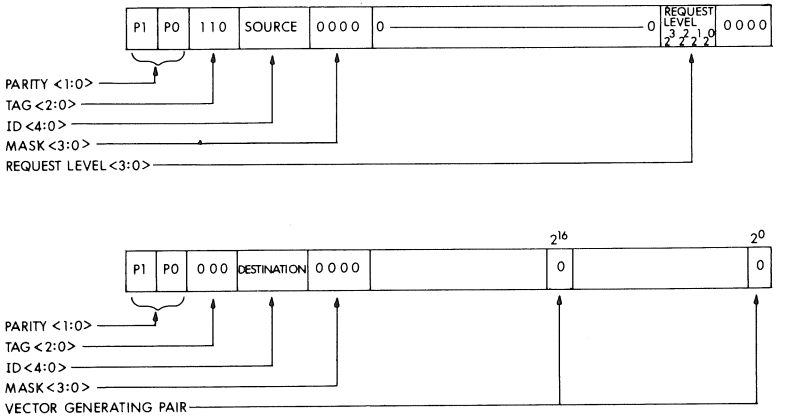


Figure 15-12 Interrupt Summary Tag Format

- Reserved Tags

Tag (<2:0>=111) is reserved for diagnostic purposes. Tag codes 001, 010, and 100 are unused and reserved for future definition.

SOURCE/DESTINATION IDENTITY FIELD

The ID field (ID<4:0>) contains a code which identifies the logical source or logical destination of the information contained in B<31:00>. ID codes are assigned only to commander and responder NEXUSs (which issue/recognize command/address information). Each NEXUS is assigned an ID code which corresponds to the TR line which it operates. For example, a NEXUS assigned TR05 would also be assigned ID code=5.

MASK FIELD

The mask field (M<3:0>) has two interpretations. In the primary interpretation, M<3:0> is encoded to specify operations on any or all bytes appearing on B<31:00>. The mask is used with the read masked, write masked, interlock read masked, interlock write masked, and extended write masked commands. As shown in Figure 15-13 each bit in the mask field corresponds to a particular byte of B<31:00>.

As previously mentioned, the secondary interpretation is used when Tag<2:0>=000 (read data). Figure 15-14 illustrates the read data mask field.

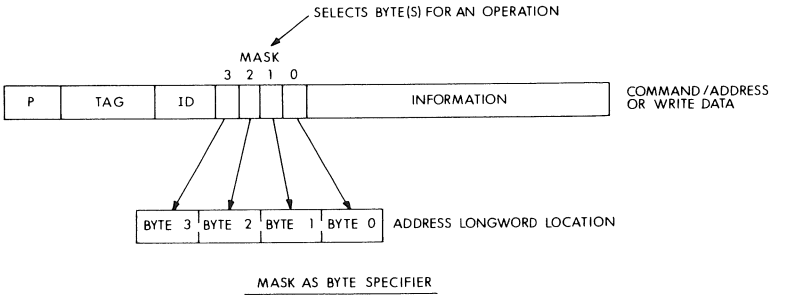


Figure 15-13 Mask Field Format

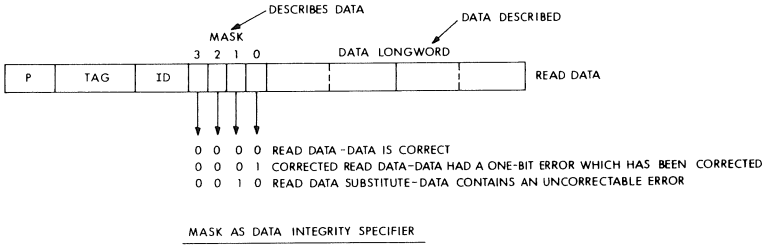


Figure 15-14 Read Data Mask Format

Response Lines

There are three response lines, broken down into two fields: confirmation CNF<1:0> and Fault (FAULT). CNF<1:0> informs the transmitter whether the information was correctly received, or if the receiver can process the command. FAULT is a cumulative error indication of protocol or information path malfunction, and is asserted with the same timing as the confirmation field. The CPU latches the fault signal, which in turn latches all the fault status registers and the SBI silo. The silo is a hardware mechanism used to record the last 16 SBI transactions. The silo aids in rapid error detection. The fault is then cleared by the software.

Either field is transmitted to the receiver two cycles after the associated information transfer. Confirmation is delayed to allow the information path signals to propagate, be checked, be decoded by all receivers, and to be generated by the responder. During each cycle, every NEXUS in the system receives, latches, and makes decisions on the information transfer signals. Except for multiple bit transmission

errors or NEXUS malfunction, one of the NEXUSs receiving the information path signals will recognize an address or ID code. This Nexus then asserts the appropriate response in CNF. The confirmation codes and their definitions are listed in Table 15-1.

Table 15-1 Confirmation Code Definitions

CNF Code	Definitions
00, No Response (N/R)	The unasserted state indicates no response to a commander's selection.
01, Acknowledge (ACK)	The positive acknowledgement to any transfer.
10, Busy (BSY)	The response to a command/address transfer, and indicates successful selection of a NEXUS which is presently unable to execute the command.
11, Error (ERR)	The response to a command/address transfer. Indicates selection of a NEXUS which cannot execute the command.

A BSY (10) or ERR (11) response to transfers other than command/address transfers will be considered as no response from the transmitter.

Interrupt Request Lines

The interrupt request group consists of four request lines (REQ <7:4>) and an alert (ALERT) line. A request line is assigned to each NEXUS that interrupts and represents its assigned CPU interrupt level. The lines are used by NEXUS to invoke a CPU to service a condition requiring processor intervention. The request lines are priority encoded in an ascending order of REQ4-REQ7. A requesting NEXUS asserts its request lines synchronously with the SBI clock to request an interrupt. Any of the REQ lines may be asserted simultaneously by more than one NEXUS, and any combination of REQ lines may be asserted by the collection of requesting NEXUSs.

The alert signal is asserted by NEXUSs which do not implement interrupt request lines. Its purpose it to indicate to the CPU a change in NEXUS status of power condition or operating environment. NEXUSs which implement the REQ lines report these changes by requesting an interrupt.

Control Lines

The control group functions synchronize system activities and provide specialized system communications. The group includes the system clock which provides the universal timebase for any NEXUS connected to the SBI. The group also provides initialization, power fail, and restart functions for the system. In addition, a path is provided for coordinating memories to assure access to shared structures.

The control lines are comprised of the following subgroups: clock functions, interlock line, dead signal function, fail function, and the UNJAM function.

CLOCK FUNCTIONS

Six control group lines are clock signals and are used as a universal time base for all NEXUSs connected to the SBI. All SBI clock signals are generated on the CPU clock module and provide a 200 nsec clock period.

INTERLOCK LINE

The interlock line will be discussed in the command code description section.

DEAD SIGNAL FUNCTION

The Dead signal indicates an impending power failure to the clock circuits on bus terminating networks. NEXUSs will not assert any SBI signal while Dead is asserted. Thus, NEXUSs prevent invalid data from being received while the bus is in an unstable state.

The assertion of the power supply DS LO to the clock circuits or terminating networks causes the assertion of Dead. Dead is asserted asynchronously to the SBI clock and occurs at least two μsec before the clock becomes inoperative. With power restart, the clock will be operational for at least two μsec before DC LO is negated. The negation of DC LO negates Dead.

FAIL FUNCTION

A NEXUS asserts the Fail (FAIL) function asynchronously to the SBI clock when the power supply AC LO signal is asserted to that NEXUS. The assertion of Fail inhibits the CPU from initiating a power up service routine. Fail is negated asynchronously to the SBI clock when all NEXUSs that are required for the power up operation have detected the negation of AC LO. The CPU samples the Fail line following the power down routine (assertion of FAIL) to determine if the power down routine should be initiated.

UNJAM FUNCTION

The UNJAM function restores (initializes) the system to a known, well-defined state. The UNJAM signal is asserted only by the console of the CPU, and is detected by all NEXUSs. The CPU asserts UNJAM only when a console key is selected. The duration of the UNJAM pulse is 16 SBI cycles and is negated at T0.

When the CPU intends to assert UNJAM it will assert TR00 for 16 SBI cycles. The CPU will continue to assert TR00 for the duration of UNJAM and for 16 SBI cycles after the negation of UNJAM. This use of TR00 insures that the SBI is inactive preceding, during, and after the UNJAM operation.

COMMAND CODE (Function 3:0) DESCRIPTION

Information bits B<31:00> carry most of the information on the SBI. Information appears on these lines in command/address format, data format, interrupt summary read format, or interrupt summary response format. In command/address format, information is grouped in three fields: M<3:0>, the byte mask: f<3:0>, the function code; and A<27:00>, a 28-bit physical address. Function codes are shown in Figure 15-15. Bit 27 of the SBI address field determines whether the longword address A <27:00> is located in memory or I/O space (refer to Chapter 17, Figure 2).

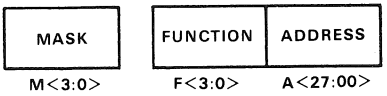
Read Mask Function (F=0001)

Once the commander has arbitrated for and gained control of the SBI, it asserts the information transfer lines at T0. The receiver latches open at T2 and closes at T3. Information in these latches is stable from T3 to the next T2.

The command/address format instructs the NEXUS selected by A<27:00> to retrieve the addressed data word, and transfer it to the logical destination specified in the ID field. The addressed NEXUS will respond to the command/address transfer with ACK (assuming the NEXUS can perform the command at this time) two SBI cycles after the assertion of command/address. Figure 15-16 illustrates the SBI read function.

Write Masked Function (F=0010)

The write masked function instructs the NEXUS selected by A<27:00> to modify the bytes specified by M<3:0> in the storage element addressed by A<27:00>, using data transmitted in the next succeeding cycle. Figure 15-17 illustrates a single SBI write transaction while Figure 15-18 illustrates two SBI write transactions from devices A and B.



MASK USE	FUNCTION CODE	FUNCTION DEFINITION
IGNORED	0000	RESERVED
USED	0001	READ MASKED
USED	0010	WRITE MASKED
IGNORED	0011	RESERVED
USED	0100	INTERLOCK READ MASKED
IGNORED	0101	RESERVED
IGNORED	0110	RESERVED
USED	0111	INTERLOCK WRITE MASKED
IGNORED	1000	EXTENDED READ
IGNORED	1001	RESERVED
IGNORED	1010	RESERVED
USED	1011	EXTENDED WRITE MASKED
IGNORED	1100	RESERVED
IGNORED	1101	RESERVED
IGNORED	1110	RESERVED
IGNORED	1111	RESERVED

Figure 15-15 SBI Command Codes

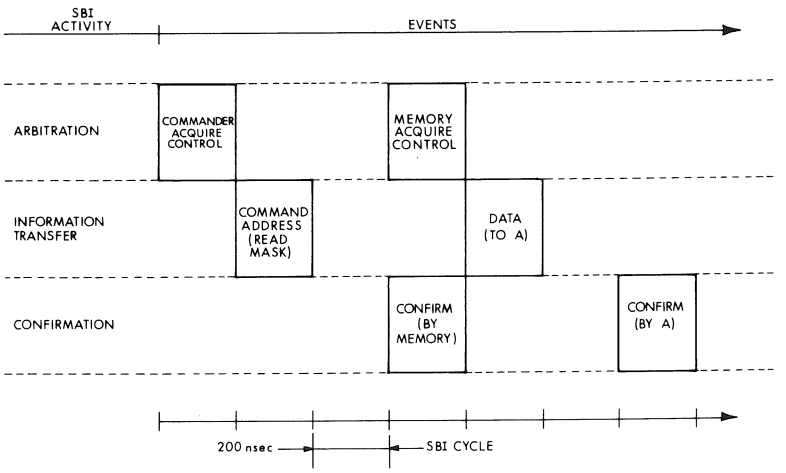


Figure 15-16 SBI Read Transaction

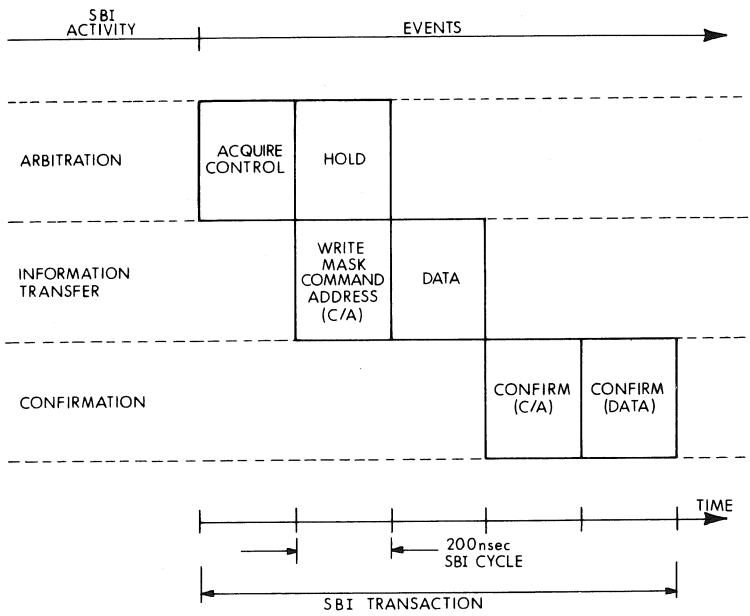


Figure 15-17 Single SBI Write Transaction

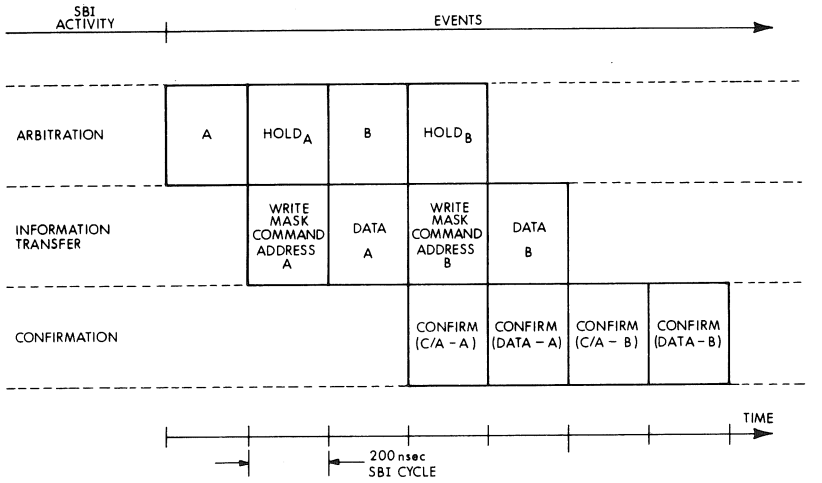


Figure 15-18 Two SBI Write Transactions

Interlock Read Masked (F=0100)
This command, used to insure exclusive access to a particular memory location, causes the NEXUS selected by A<27:00> to retrieve and transmit the addressed data as for Read Masked. In addition, this command causes the selected memory controller NEXUS to set an Interlock flip-flop. Only memory NEXUSs have the ability to assert Interlock. While this flip-flop is set, the NEXUS will assert the INTLK signal synchronously at time T0. Interlock is asserted during the same cycle as the confirmation signal. In the preceding cycle, the commander must assert Interlock. While the INTLK signal is asserted, the NEXUS will respond with BSY confirmation to Interlock read masked commands. The Interlock flip-flop is cleared on receipt of an Interlock write masked function. Interlock read masked and Interlock write masked are always paired by commanders utilizing them.

Interlock Write Masked (F=0111)
The Interlock write masked function instructs the NEXUS selected by A<27:00> to modify the bytes specified by M<3:00> in the storage element addressed, using data transmitted in the succeeding cycle with TAG=101. Additionally, the Interlock flip-flop is cleared.

Extended Read (F=1000)
The Extended Read function instructs the NEXUS selected by A<27:00> to retrieve the addressed 64-bit data and transmit it to the ID accompanying the command as in the read masked function. The responder transmits the data in two successive cycles with the low 32 bits (A00=0) preceding the high 32 bits (A00=1). Two data words are always transmitted. M<3:0> must be transmitted as 0000 by the commander. Figure 15-19 illustrates extended read transactions by two separate devices, A and B, reading memory via a single memory controller.

Figure 15-20 illustrates extended read transactions by two separate devices, A and B, reading memory via separate memory controllers, M1 an M2.

Extended Write Masked (F=1011)
The Extended Write Masked function instructs the NEXUS selected by A<27:00> that 64 bits of data are to be written. The receiver ignores A00 of the command/address transfer. A<27:00> indicates the low 32 bit word address. The write data is transmitted in two 32 bit words. The first word corresponds to A00=0 and the second word corresponds to A00=1. M<3:0> that accompanies the command address transmission indicates bytes to be written in the first write data word. M<3:0>

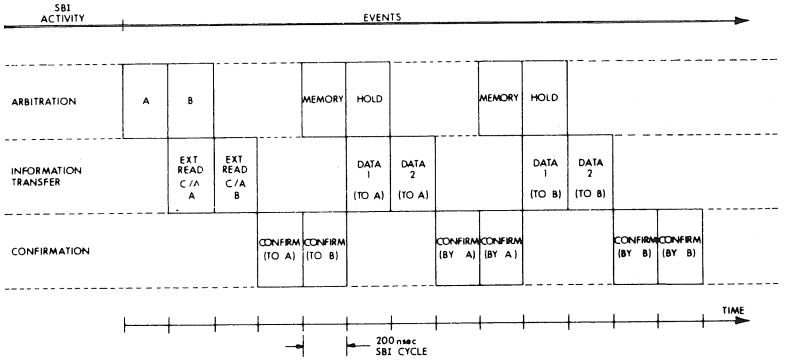


Figure 15-19 Extended Read Transactions Via Single Memory Controller

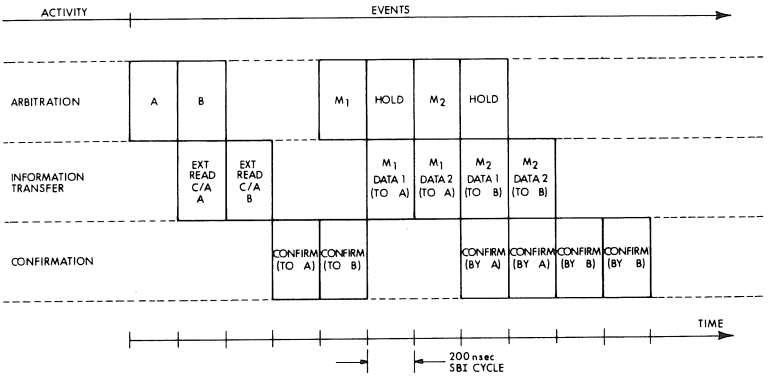


Figure 15-20 Extended Read Transactions Via Separate Memory Controllers

that accompanies the first write data word transmission indicates bytes to be written in the second write data word. The M<3:0> field of the second data word cycle is ignored by receivers but must be transmitted as zeros. The assertion of a particular mask bit signifies that the byte corresponding to that mask bit is to be modified. The NEXUS implmenting the Extended Write Masked function must implement all combinations of M<3:0>.

SYNCHRONOUS BACKPLANE INTERCONNECT THROUGHPUT

The following is a derivation of the aggregate throughput rate of the SBI:

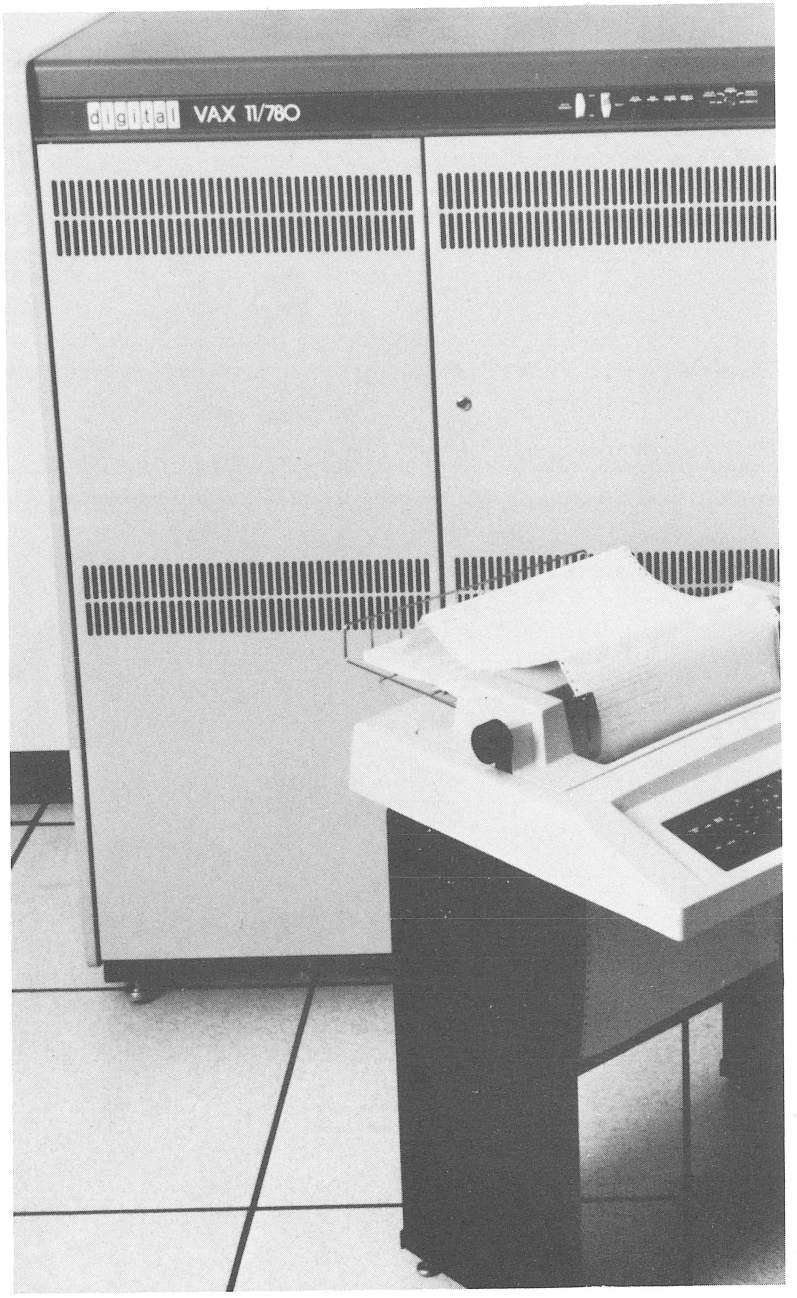
200 nanoseconds/cycle = 5 million cycles/second.

Each cycle can carry an address (memory request) or four bytes of data.

Thus, one cycle is used to request eight bytes of data (to be read or written), and two cycles are used to carry data (at four bytes/cycle).

5 million cycles/second x 4 bytes/cycle = 20 million bytes/second.

20 x 2/3 (1 of every 3 cycles is an address) = 13.3 million bytes/second.


digital VAX 11/780

CHAPTER 16

VAX-11/780 MAIN MEMORY SUBSYSTEM

FEATURES	BENEFITS
Expandable memory configuration	Allows the addition of 256 KB array cards up to a maximum of 4 MB per controller, or 8 MB (12 MB with two MA780s)
Error correcting memory controller	Enhances data availability and reliability by correcting all single bit errors and detecting all double bit errors within the memory system. Permits write operations to any combination of bytes within an aligned longword.
Interleaving (with two controllers)	Improves memory subsystem throughput on the bus
Optional battery backup	Maintains power to preserve data for 4 MB of memory for at least ten minutes.

INTRODUCTION

Main memory is a dynamic MOS (metal oxide semiconductor), random access memory designed to interface with the VAX-11/780 synchronous backplane interconnect.

The memory subsystem consists of a controller and one to sixteen array boards utilizing either 4 K or 16 K N-channel MOS IC storage elements. Each array board can contain 64 K or 256 KB of memory, giving the system a capacity of either 1 or 4 MB, depending upon size of storage chips used.

Memory is capable of random access read and write operations to a single 32-bit longword or extended 64-bit quadword. Memory is also capable of random access write to an arbitrary byte, series of contiguous bytes, or a series of noncontiguous bytes. The memory array board has been organized to optimize eight byte read/write access.

Memory features an error checking and correcting scheme (ECC) which can detect all double bit errors and detect and correct all single bit errors. The error detection and correction algorithm requires an entire quadword of data and thus during any type of read or write

operation an entire quadword of data is fetched from the array.

Eight ECC check bits are stored with each quadword and accessed with the data to determine its integrity. Therefore, a total of 72 bits are accessed at once.

The basic memory subsystem is illustrated in Figure 16-1.

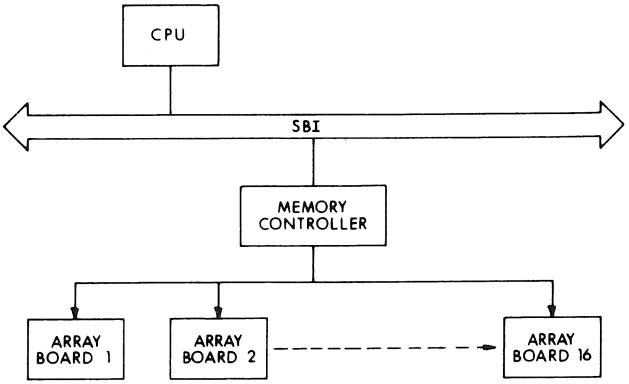


Figure 16-1 Main Memory Configuration

MEMORY CONTROLLER

The memory controller is the NEXUS interfacing main memory to the SBI. The controller examines the command and address lines of the SBI for each SBI cycle. To initiate and complete a memory write masked, interlock write masked, or extended write masked transaction, the controller must receive a recognizable command or address and data in two or three SBI cycles. However, to perform a read masked, extended read, or interlock read masked operation, the controller need only recognize an appropriate command/address. The controller provides the necessary timing and control to complete all memory transactions. The controller informs a commander, via a confirmation, of a successful write operation and contends and arbitrates for SBI bus control to transmit information during a read masked, extended read, or interlock read operation. However, before the controller will initiate a memory cycle operation, it checks for bus transmission parity errors and other fault conditions and reports these conditions to the commander, conforming to the SBI protocol. Data transfers to and from main memory are protected by ECC logic, i.e., main memory contains single bit error detection and correction and double bit error detection logic to improve system reliability.

Error reporting provides an early warning to protect the system from performance degradation. The system error logging feature requires tagging single bit errors and uncorrectable errors during memory read transmission from the memory subsystem. Also saved in the memory controller are the syndrome bits for the first memory read error and the error address for error logging and servicing. The memory controller retains this information until the first error is serviced. There are ten bits in register B that are used for ECC diagnostics only. In addition to its error detection capabilities, the controller provides the logic to buffer commands, addresses, and data, thus improving memory throughput.

A system may require more than one memory controller. If the system requires a two-controller interleaved memory configuration, the memory controllers must have consecutive TR selects. The interleave bit will be cleared on power up and must be set by writing to configuration register A in each controller. Each controller must have the same array size and be issued the same starting address. In the case of multiple memory controllers, each controller will assume a different starting address on power up. The proper starting address will be written into the configuration B register from the SBI bus.

A read-only memory that can be addressed on the SBI bus resides in the memory subsystem. The address, timing, and control logic to read the information from the ROM for booting the system is also contained in the subsystem.

The memory controller provides power up initialization logic and refresh control logic for the dynamic MOS memory devices. The dynamic MOS memory cell is a capacitor in which the stored charge represents a data bit. As the stored charge tends to diminish over a period of time, each cell requires a refresh cycle every 2 msec to retain the charge reliability.

BASIC MEMORY OPERATIONS

The memory subsystem operates synchronously with the SBI clock cycles, satisfying the system communication protocol. As discussed in Chapter 17, *Synchronous Backplane Interconnect*, the physical address space is divided into two equal areas, memory address space, and I/O address space. Figure 16-2 illustrates the physical address space.

The 28-bit (A<27:00>) SBI longword address field (refer to Figure 16-2) is capable of accessing up to 512 MB of main memory. The hardware, however, will currently support a maximum of 2 MB of main memory utilizing the 4K chip design and 8 MB utilizing the 16K chip design. Physical memory operations are performed when bit <27> of

Figure 16-2 is zero. I/O operations occur when bit <27> is one. The operation field identifies one of the following six transactions performed by the memory subsystem:

- Read Masked
- Extended Read
- Interlock Read Masked

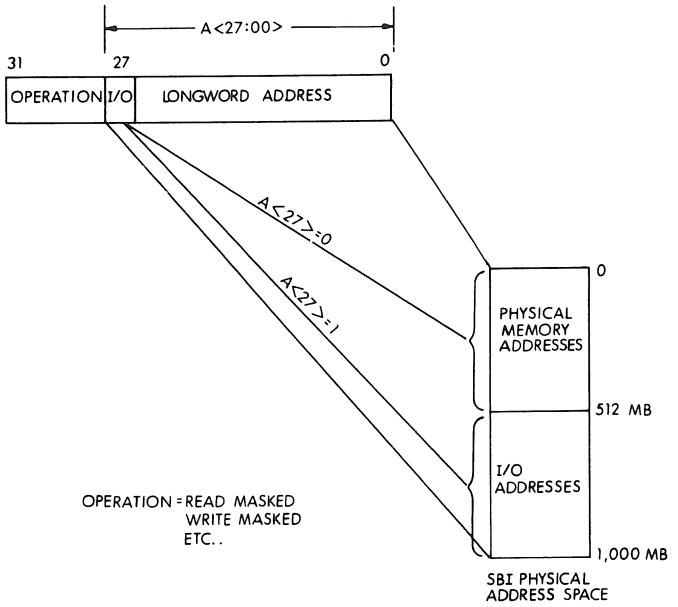


Figure 16-2 Physical Address Space

- Write Masked
- Extended Write Masked
- Interlock Write Masked

A write mask operation is executed to transfer one to four bytes of data to memory. A read mask operation, however, is only capable of transferring four bytes of data from memory.

An Extended Read is executed to transfer eight bytes of data (two longwords) from memory to a requesting NEXUS. An Extended Write Masked, on the other hand, provides a byte-selectable transfer of up to eight bytes to memory. Interlock Read Masked and Interlock Write Masked perform the same function as Read Masked and Write Masked but also provide process synchronization.

Read Cycle

The read cycle will fetch 32 bits of data from the addressed location in the memory subsystem, will check for a single or double bit error, will correct a single bit error if it exists, and will transmit the data word along with the proper tag and ID code of the commander that requested the data. In the event a single bit error occurs during the read operation, corrected data would be rewritten into the memory in a subsequent memory cycle. In the case of a double bit error, the exact data and check code read is rewritten to ensure that the double bit error recurs on subsequent reads. In either case, an indication of the error condition would be tagged and transmitted with the corrected data or uncorrectable bad data during the next available bus cycle to the requester. The sequence of events that initiates a read cycle in a memory subsystem is as follows:

Any commander on the SBI (central processor or I/O controller) that wants to initiate a read cycle in any one of the memory subsystems on the bus will arbitrate and gain control of the bus. Having gained control, the commander then transmits a command or address tag and identification code information on the bus. All subsystems on the bus monitor and decode the tag and command or address lines prior to initiating any action. If the decoded address corresponds to the memory subsystem and if no faults are detected, it would immediately (unless already busy) initiate a memory cycle. If the memory is presently executing a cycle, the command will be stored in the queue, if there is room in the buffer, until the present cycle is complete. Either way, the memory will notify the commander that the message has been received. The address under interrogation would be fetched from the memory, while the memory controller in the meantime would request, arbitrate, and gain control of the bus to transmit the data along with the commander's identification code. Read cycles with single bit errors require an extra bus cycle to correct the error, and therefore the controller would re-request the bus and transmit data after gaining control of the bus.

Extended Read

The extended read cycle is the same as the read cycle, except that it fetches 64 bits of data from the addressed location. Also, the data would be transmitted on the SBI in two successive bus cycles; the lower 32 bits are transmitted first and then the upper 32 bits. In the event a single bit error occurs, the start of data transmission would be delayed until the memory controller re-requests the bus and gains control of the bus.

Write Masked

The write masked function instructs the memory controller selected by the address (A <27:00>) to modify the bytes specified by (M<3:0>) in the storage element addressed, using data transmitted in the next succeeding cycle.

This is accomplished in the memory subsystem in two successive memory cycles, a read followed by a write cycle as the memory is organized as an 8K x 72, with an ECC over 64-bit width. During the read portion of the cycle, the 64-bit word is retrieved, the error code checked, and the appropriate bytes are modified in the upper or lower half of the word. New check bits are then encoded and the modified word is written into the memory. If a single bit error occurs during the read portion of the cycle it would be corrected. In the case of an uncorrectable error the bad data would be rewritten into the memory with the bad check code and the new data would not be used.

Up to four bytes in the upper or lower word can be modified in a write masked cycle.

Extended Write Masked

The extended write masked function instructs the memory controller selected by the address (A <27:00>) to first modify the bytes specified by (M<3:0>) in the low 32 bits of storage element addressed, using data transmitted in the next succeeding cycle. Then the controller is to modify the bytes of the high 32-bit storage element specified by the masks (M<3:0>) field found in the first data word cycle, using data transmitted in the next succeeding cycle. The mask field in the second data word transmission is ignored.

The implementation of this cycle is similar to write masked except in the following areas:

- One to eight bytes of an address can be modified during this operation in the upper and lower word.
- An extended write masked that specified modification to all eight bytes does not execute a read cycle first but unconditionally writes the new 64 bits and eight check bits to the designated address. This is described as a full write cycle.

INTERLOCK CYCLES

The interlock cycles are special memory cycles used for process synchronization. They consist of the interlock read cycle and the interlock write cycle. The memory controller treats the interlock cycles as a pair of cycles, with an interlock read masked always followed (an arbitrary number of cycles after) by an interlock write masked. Interlock read and write cycles are 32-bit operations. The interlock line on the SBI is

used to coordinate activity between memory controllers. An interlock timer of 512 bus cycles is started with the acceptance of an interlock write. If the interlock write is not found, after 512 bus cycles, the interlock line is cleared.

Interlock Read

The interlock read masked cycle is implemented in the same manner as the read masked cycle, with the following exception. The interlock read has a special function code which the memory controller decodes and also monitors the interlock line on the bus to verify any interlock activity elsewhere in the system. If the interlock line is not asserted, the memory controller addressed would acknowledge the cycle and set its interlock line on the SBI until a valid interlock write has been received.

In the case of a single bit error, the controller corrects the data and transmits it with the proper tag. If an uncorrectable error occurs, the read data substitute tag with the bad data would be transmitted and the memory would rewrite the bad data and bad ECC.

Every commander on the SBI capable of issuing an interlock command should also assert the interlock line on the bus for one cycle immediately following the interlock read mask command. This is to insure cooperation among memory controllers responding to interlock reads without ambiguity.

Interlock Write

The interlock write masked cycle is similar to the write masked cycle with the following exceptions:
The set state of the interlock line on the SBI would verify the integrity of the command prior to acknowledging the cycle and implementing it. The interlock flip-flop would be cleared and consequently the interlock line on the bus would be deasserted.

If the interlock line was not asserted, the write interlock command would not be executed and the interlock sequence fault would be set.

ERROR CHECKING AND CORRECTION (ECC)

The ECC scheme used in the memory subsystem is capable of detecting a single or double bit error. It is also capable of correcting all single bit errors. This is accomplished by storing eight check bits, along with the 64 data bits in each memory location. Each check bit is generated by parity-checking selected groups of data bits in the given data quadword. When parity is again checked during a read, an incorrect bit will be detected by the parity-checking logic and will develop a unique 8-bit syndrome which will identify the bit in error. Error correction logic

may thus correct the bit in error. There are 72 unique syndromes pointing to individual bits in the coded quadword.

MEMORY CONFIGURATION REGISTERS

There are three configuration registers in the memory controller to provide configuration-dependent information to the operating system and diagnostic software. These are addressable registers with read and write access.

Memory Configuration Register A

Figure 16-3 illustrates memory configuration register A.

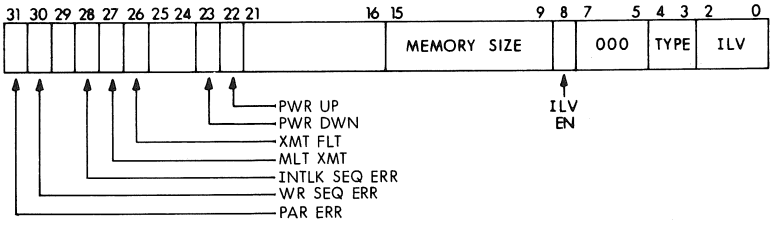


Figure 16-3 Memory Configuration Register A

Register A contains the following information:
Bit: <31:26>
Name: SBI Fault Status
Function:

Bit: <23:22>
Name: Power Up/Power Down Status
Function: These bits work in conjunction with the Alert line. If the memory is strapped to inhibit ROM decode, the assertion of AC LO will set the power down status, clear the power up status and activate the Alert line. The deassertion of the AC LO signal will set power up status, clear power down status and assert the Alert line. Writing a one to the active status bit will clear it and deassert Alert.

Bit: <15:9>
Name: Memory Size
Function: These bits contain the binary representation of the memory size in 64 KB increments, zero inclusive. For the 4K chip, bits <12:9> are used. For the 16K chip, bits <14:9> are used. These bits are read-only.

Bit: <4:3> **Name:** Memory Type
Function: These bits specify the memory type. This refers to the 4K MOS chip implementation or the 16K MOS chip implementation.

These bits are read-only.

Bit <4>	Bit <3>	Description
0	0	Error condition, no array cards plugged in
0	1	4K chip
1	0	16K chip
1	1	Error condition, both 4K and 16K chip array boards are being used.

Bit: <2:0> **Name:** Interleave
Function: These bits contain interleave information. If bit <0> is 0, the memory is not interleaved. If bit <0> is a 1, the system is interleaved. Bits <1> and <2> are not used at this time and should be 0. Bit <8> is the interleave write enable bit. When bit <8> is written to with a one, bit <0> will take on whatever state bit <0> in the written data is. Bit <8> will always read as a 0. If bit <8> is written to with a 0, interleave bit <0> will be unchanged. The interleave flip-flop receives its power from the +5V BAT supply so it retains its state during battery backup. On a cold start, this bit will come up 0.

Memory Configuration Register B

Figure 16-4 illustrates memory configuration register B.

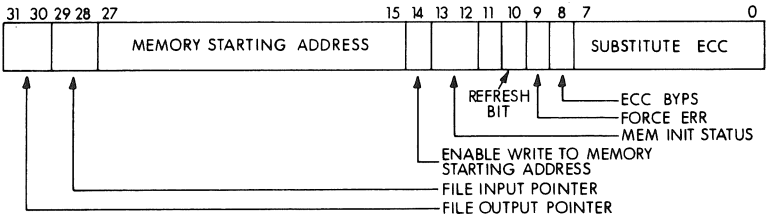


Figure 16-4 Memory Configuration Register B

Register B contains the following information:

Bit: <31:30>
Name: File Output Pointer (File Read Address Counter)
Function: These two bits point to the address that would be read from the command and data file and operated on by the timing and control logic for starting a new memory cycle at the appropriate time, depending on the state of the memory busy line. This information is also useful for diagnosing the file control logic problems in the file read path. Bit <31> is the most significant bit, bit <30> is the least significant bit.

Bit: <29:28>
Name: File Input Pointer (File Write Address Counter)
Function: The memory controller command buffer (File) is four addresses deep and the Write Address Counter state can be read via these two bits. These bits point to the next available file address into which the command address or data information will be written after accepting the command address and data from the SBI. These bits assist in diagnosing the file control logic problems in the file write path. Bit <29> is the most significant bit, bit <28> is the least significant bit.

Bit: <27:15>
Name: Memory Starting Address
Function: These bits indicate the starting address of the memory controller in 64 KB granularity or increments. These bits are writable and can be altered by the system after power up. During a cold start the memory controller would come up with a default starting address depending on the starting address jumpers in the memory backplane. A cold start is defined as a CPU power up from inactive battery backup and memory power supply. There are two starting address jumpers in the memory controller backplane, and in a four controller system the default starting address assignments are as follows for cold starts.

Controller No.	Starting Address Jumpers	Starting Address
	SA 01 SA 00	
1	OPEN OPEN	zero
2	OPEN GND	4 Megabyte
3	GND OPEN	8 Megabyte
4	GND GND	12 Megabyte

Also during battery backup the contents of the starting address bits are saved.

Bit: <14> **Name:** Write Enable to Memory Starting Address
Function: This bit must be at a one state during a write to register B in order to alter the state of the memory starting address. If bit <14> is a zero, writes to register B will leave the starting address unchanged.

Bit: <13:12>
Name: Memory Initialization Status
Function: These are read-only bits and contain the recovery mode information necessary to determine whether or not the memory has recovered from battery backup and therefore contains valid data.

Bit <12>	Bit <13>	Description
0	0	Initialization cycle in process. This

		means the memory is presently writing a known data pattern and check code throughout the storage area. A command issued to the array at this time will receive a busy response.
0	1	Invalid state
1	0	This state means the memory contains valid data. This state after a CPU Power Fail implies that all memory data was saved.
1	1	This state signifies that initialization is complete and that the power restoration was from a cold state. No data was preserved.

Bit: <10> **Name:** Refresh Indication
Function: This bit is used for diagnostic purposes only, and will verify the access time delay due to refresh collision.

Bit: <9> **Name:** Force ERR
Function: This bit is used in conjunction with bits <7:0>. When it is set, it will enable the ECC substitute bits to replace the actual check bits for the ECC computation when operating on an address with SBI bits 3 and 12 active. Writing a one sets this bit and writing a zero clears it.

Bit: <8> **Name:** ECC BYPS
Function: This bit is set to totally bypass (BYPS) the ECC check function. If this bit is set, the data that is read from the memory will be placed on the SBI exactly as it is found. Also, no CRD or RDS flag will accompany the data if it is in error but the error log will continue to operate normally (register C).

Writing a one sets this bit, writing a zero clears it. This bit is used for diagnostics only.

Bit: <7:0> **Name:** Substitute ECC Bits
Function: These bits can be substituted for the eight check bits read from the memory, providing that bit <9> in Register B is set to a one and the address read contains SBI bits 3 and 12 active. These bits are for diagnostics only and can be used to simulate any single bit or multiple bit error, thereby checking the entire ECC path. Writing a one sets the bits, writing a zero will clear them.

Memory Configuration Register C

Figure 16-5 illustrates memory configuration register C.

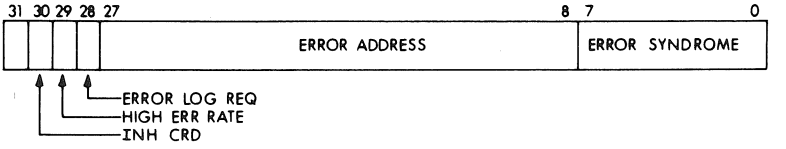


Figure Figure 16-5 Memory Configuration Register C

This register gathers all the ECC error information:

Bit: <30> **Name:** Inhibit CRD
Function: This bit is used to prevent constant CRD flags from being sent to the commander when working in sequential memory locations with single cell failures, thus preventing repeated error service invocation by the operating system. Writing a one to this bit prevents subsequent CRD flags from being transmitted to the commander until such a time as the commander writes a zero to bit <30>. However, in the event an uncorrectable error occurs in the memory it would be reported right away regardless of the state of this bit.

Bit: <29> **Name:** High Error Rate
Function: This bit flags the high error rate in the memory by setting this bit if an error occurs between the time the first error message was sent and the time the error service subroutine was invoked by the operating system. This bit can be cleared by writing a one.

Bit: <28> **Name:** Error Log Request Flag
Function: This bit is set when the first error occurs during the memory controller's response to an SBI read cycle. This would indicate to the error service subroutine whether the controller has logged an error during its operation or not. When this bit is set, any subsequent CRD reports to the bus commander will be inhibited. In a multiple memory controller system, this is needed in determining which controller sent the error message. This can be cleared by writing a one.

Bit: <27:8>
Name: Error Address
Function: The SBI longword address at which the first read error occurred during memory controller response to an SBI read command is saved in these bits. Subsequent error addresses, if they occur, are not saved until the first one is serviced.

The address field is described as follows:
(bit order is least to most)

Bit <8> indicates the word in error.
0 = lower word
1 = upper word
Bits <20:9> indicate the 4K chip address in error.
Bit <21> indicates the 4K chip array bank in error.
0 = lower 4K chip
1 = upper 4K chip
Bits <23:22> are unused for 4K chip.

Used in 16K chip for two necessary extra chip address bits. All chip address and bank address bits shift left two.
Bits <27:24> indicate the array card in error.

Bit: <7:0> **Name:** Error Syndrome
Function: These eight bits store the error syndrome of the first error word that was read from memory in response to an SBI read command. The syndrome will be saved until the error service routine has serviced the error. Subsequent error syndromes will not be saved but will be indicated by bit <29>.

MEMORY INTERLEAVING

The memory subsystem is capable of operating in the non-interleaving or two-way interleaved mode. Interleaving improves memory subsystem throughput on the bus.

In a single memory controller system the starting address is assigned by the ROM bootstrap. The size of the memory subsystem is encoded from the number of array cards plugged into the backplane. Array boards must be mounted contiguously in the backplane. If boards are misplugged, an indicator light indicates configuration error.

Interleaving can be used to increase the overall speed of the memory subsystem when there are two memory controllers with equal amounts of MOS memory on each. The effectiveness of interleaving is based on the principle that most memory operations are performed on consecutive memory locations. While one controller is fetching data, the other controller is available to decode an address for the next operation. On the VAX-11/780, the two memory controllers access alternate quadwords.

With an interleaved memory system, both controllers must have contiguous bus TR select levels (odd and even pairs), the same array size, the same starting address, and both controllers must have their interleaved bits set.

It is also possible to have two two-way interleaved memory systems, four controllers, by following the rules just listed and assigning the

second interleaved memory system a starting address that is one location above the final address of the first interleaved set.

Four memory controllers on one bus may require reassigning of bus TR select levels of the other SBI NEXUS.

ROM BOOTSTRAP

A 4 KB programmable read-only memory to boot the system resides in the memory controller and it uses a 1 K x 4, bipolar, high speed device. The memory is organized as a 1 K x 32 and is assigned 4 KB I/O address space. The ROM can be addressed via the SBI interface in the memory controller during system initialization. All the address, data and control logic for addressing the ROM bootstrap is in the memory controller. The ROM is packaged in such a way that ECOs can be easily handled by providing sockets in the PROM locations.

ROM access time = 5 bus cycles (with respect to the commander).

BATTERY BACKUP

A battery backup option is available that will maintain the contents of 4 MB of MOS memory for a minimum of 10 minutes, or less memory for longer than 10 minutes.


VAX 11/780

CHAPTER 17

VAX-11/780 UNIBUS SUBSYSTEM

FEATURES	BENEFITS
Direct memory access (DMA) data transfers	Eliminate processor intervention for high data throughput
Peer communication between UNIBUS devices	Allows direct data transfer between UNIBUS devices without CPU involvement
Total compatibility with the PDP-11 UNIBUS data path	Gives the PDP-11 user an easy migration path to the VAX-11/780 processor
UNIBUS device registers are addressed like memory locations	Simplifies I/O programming
Supports a wide range of standard DIGITAL peripherals	Offers the user flexibility in peripheral selection to meet specific requirements
15 buffered data paths	Increases data throughput by buffering data traffic to memory

INTRODUCTION

The UNIBUS subsystem is the hardware developer's primary interface to VAX-11/780. All devices other than high-speed disk drives, magnetic tape transports and devices which use the high performance DR780 interconnect are connected to the UNIBUS, an asynchronous bidirectional bus. The UNIBUS is connected to the SBI through the UNIBUS adapter. The UNIBUS adapter does priority arbitration among devices on the UNIBUS.

The UNIBUS adapter provides access from the processor to the UNIBUS peripheral device registers and to UNIBUS memory by translating UNIBUS addresses, data, and interrupt requests to their SBI equivalents, and vice versa. The UNIBUS adapter address translation map translates an 18-bit UNIBUS address to a 30-bit SBI address. The map provides direct access to system memory for nonprocessor request UNIBUS peripheral devices and permits scatter/gather disk transfers.

The UNIBUS adapter enables the processor to read and write the peripheral controller status registers. In the case of processor interrupt request devices, this constitutes the transfer.

This chapter is organized to provide the reader with an understanding of the UNIBUS and the VAX-11/780 UNIBUS adapter. The UNIBUS subsystem is comprised of the UNIBUS adapter logic, the UNIBUS, and associated peripheral devices. Figure 17-1 illustrates the UNIBUS subsystem configuration.

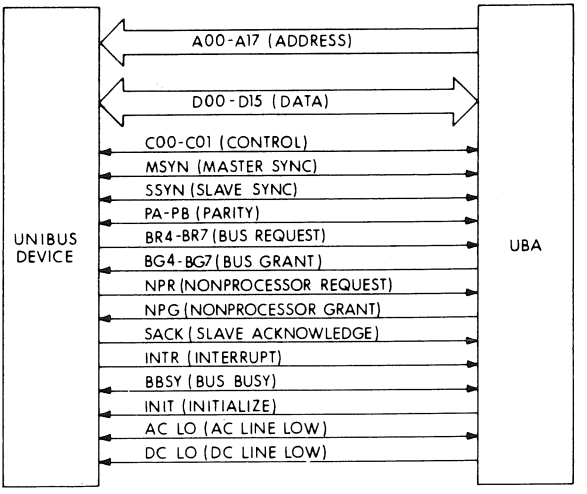


Figure 17-1 UNIBUS Configuration

UNIBUS SUMMARY

The UNIBUS, a high-speed communication path, links together I/O devices to the UNIBUS adapter. Device-related address, data, and control information are passed along the 56 lines of the UNIBUS. The UNIBUS adapter handles all communications between the UNIBUS and the SBI, and fields device-generated interrupts.

The following UNIBUS summary description takes into account the presence of the UBA, which performs the following UNIBUS functions:

- arbitration
- interrupt fielding
- power fail/restart
- initialization

For example, the UBA enables the system to accept device interrupts and transfer the requests from the UNIBUS to the SBI. However, UBA and SBI operations between the VAX-11/780 CPU and UNIBUS are transparent to the UNIBUS devices.

Communications and Control

A master/slave relationship defines all communications between devices on the UNIBUS. The device in control of the bus is considered the master; the device being addressed is the slave. Communication on the UNIBUS is interlocked, that is, each control signal issued by the master device must be acknowledged by a corresponding response from the slave to complete the transfer.

Bus Request Levels

Each device uses one of five priority levels for requesting bus control: Non-Processor Requests (NPR) and four Bus Requests (BR). The NPR is used when a device requests a direct access data transfer to memory or another device (i.e., a transfer not requiring processor intervention). Normally, NPR transfers are made between a mass storage device (e.g., disk drive) and memory. Two bus lines are associated with the NPR priority level. The device issues its request on the NPR line; the UBA responds by issuing a grant on the Non-Processor Grant (NPG) line.

A BR level is used when a device interrupts the VAX-11/780 CPU in order to request service. The device may require the CPU to initiate a transfer. Or it may need to inform the CPU that an error condition exists. Two lines are associated with each of four BR levels. The bus request is issued on a BR line (BR7-BR4); the bus grant is issued on the corresponding Bus Grant line (BG7-BG4).

Priority Structure and Chaining

When a device requests use of the bus, the handling of that request depends on the location of that device in a two-dimension device-priority structure. Priority is controlled by the priority arbitration logic of the CPU and the UBA.

The device-priority structure consists of five priority levels: NPR and BR7-4. Bus requests from devices can be made on any one of the request lines. The NPR has highest priority; BR7 is the next highest priority, and BR4 is the lowest. The priority arbitration logic is structured so that if two devices on different BR levels issue simultaneous requests, the priority arbitration logic grants the bus to the device with the highest priority. However, the lower priority device keeps its request up and will gain bus control when the higher-priority device finishes with the bus (providing that no other higher-priority device issues a BR).

Since there are only five priority levels, more than one device may be connected to a specific request level. If more than one device makes a request at the same level, the device closest (electrically) to the UBA

has highest priority. The grant for each BR level is connected to all devices on that level in a daisy-chain arrangement (chaining). When a corresponding BG is issued it goes to the device closest to the UNIBUS adapter. If that device did not make the request it permits the BG to pass to the next closest device. When the BG reaches the device making the request, that device captures the grant and prevents it from passing on to any subsequent device in the chain. Functionally, NPG chaining is similar to BG chaining.

Device Register Organization

The actual transfer of data and status information over the UNIBUS is accomplished between status, control, and data buffer registers located within the peripheral devices and their control units. All device registers are assigned addresses similar to memory addresses. These registers can therefore be accessed by word type memory reference instructions (i.e., MOVW, BITW, etc.).

Control and status functions are assigned to the individual bits within the corresponding addressable registers. Since the register content can be controlled, setting and clearing register bits can control service operations. Internal device status may be loaded into the appropriate register and retrieved when a program instruction addresses that register. Depending on the function, register bits may be read/write, read only, or write only. The number of addressable registers in a device (and control unit) varies depending on the device's function.

UNIBUS Line Definitions

The UNIBUS consists of 56 signal lines which may be divided into three function groups: bus control, data transfer, and miscellaneous signals. The 13 lines of the bus control group comprise those signals required to gain bus control through an NPR/BR or for a priority arbitration to select the next bus master while the current bus master is still in control of the bus. The 40 bidirectional lines of the data transfer group are those signals required during data transfers to or from a slave device. The miscellaneous group are the initialization and power fail signals required on the UNIBUS. Table 17-1 describes the bus signals within each group.

Table 17-1 UNIBUS Signal Descriptions

SIGNAL LINE	DESCRIPTION
Data Transfer Group	
Address Lines	These lines are used by the master device to se-

(A<17:0>)	lect the slave (actually a unique memory or device register address). A <17:1> specifies a unique 16-bit word; SA00 specifies a byte within the word.
Data Lines (D<15:0>)	These lines transfer information between master and slave.
Control (C1,C0)	These signals are coded by the master device to control the slave in one of the four possible data transfer operations specified below. Note that the transfer direction is always designated with respect to the master device.

Data Transfer Designation Description

C1	C0	
0	0	Data in (DATI): a data word or byte transferred into the master from the slave.
0	1	Data in Pause (DATIP): similar to DATI except that it is always followed by a DATO or DATOB to the same location. The master keeps control of the UNIBUS during the entire DATIP-DATO sequence.
1	0	Data Out (DATO): a data word is transferred out of the master to the slave.
1	1	Data Out Byte (DATOB): identical to DATO except that a byte is transferred instead of a full word. Address bits A00 determine which byte will be written. A00=0, low byte (D07-00) is written. A00=1, high byte (D15-08) is written.

Parity A-B (PA,PB)	These signals transfer UNIBUS device parity information. PA is currently unused and not asserted. PB, when true, indicates a device parity error.
Master Synchronization (MSYN)	MSYN is asserted by the master to indicate to the slave that valid address and control information (and data on a DATO or DATOB) are present on the UNIBUS.
Slave Synchronization	SSYN is asserted by the slave. On a DATO it indicates that the slave has latched the write data. On

(SSYN)	a DATI or DATIP it indicates that the slave has asserted read data on the UNIBUS.
Interrupt (INTR)	This signal is asserted by an interrupting device, after it becomes bus master, to inform the UBA that an interrupt is to be performed, and that the interrupt vector is present on the data (D) lines. INTR is negated upon receipt of the assertion of SSYN by the UBA at the end of the transaction. INTR may be asserted only by a device which obtained bus mastership under the authority of a BG signal.

Priority Arbitration Group

Bus Request (BR7-BR4)	These signals are used by peripheral devices to request control of the bus for an interrupt operation.
Bus Grant (BG7-BG4)	These signals form the UBA's response to a bus request. Only one of the four will be asserted at any time.
Nonprocessor Request (NPR)	This is a bus request from a device for a transfer not requiring CPU intervention (i.e., direct memory access).
Nonprocessor Grant (NPG)	This is the grant in response to an NPR.
Select Acknowledge (SACK)	SACK is asserted by a bus-requesting device after having received a grant. Bus control passes to this device when the current bus master completes its operation.
Bus Busy (BBSY)	BBSY indicates that the data lines of the UNIBUS are in use and is asserted by the UNIBUS master.

Initialization Group

Initialize (INIT)	This signal is asserted by the UBA when DC LO is asserted on the UNIBUS, and it stays asserted for ten msec following the negation of DC LO. It is used to initialize UNIBUS peripherals.
AC Line Low (AC LO)	This is a signal which indicates that a power failure is about to occur on the UNIBUS. The assertion of this signal initiates the UNIBUS power fail sequence of the UBA and can cause an

interrupt to the VAX-11/780 CPU. It may also be used by peripheral devices to terminate operations in preparation for power loss.

DC Line Low (DC LO) — This signal is available from each system power supply and remains clear as long as all DC voltages are within the specified limits. If an out-of-voltage condition occurs, DC LO is asserted.

THE UNIBUS ADAPTER

The UNIBUS adapter provides the interface between the asynchronous UNIBUS and the Synchronous Backplane Interconnect in the VAX-11/780. The UNIBUS adapter provides the following functions:

- Access to UNIBUS address space (i.e., UNIBUS device registers) from the SBI
- Mapping of UNIBUS addresses to SBI addresses for UNIBUS DMA transfers to SBI memory
- Data transfer paths for UNIBUS device access to random SBI memory addresses and high-speed transfers for UNIBUS devices that transfer to consecutive increasing memory addresses
- UNIBUS interrupt fielding
- UNIBUS priority arbitration
- UNIBUS power fail sequencing

The UNIBUS Subsystem is illustrated in Figure 17-2.

VAX-11/780 hardware will support a UNIBUS adapter in one of four physical address spaces. The UNIBUS adapter maintains two independent address spaces within the Synchronous Backplane Interconnect I/O address space. The first area of addressable space is within the area reserved for all NEXUSs (i.e., UBA, MBA, memory controller) internal registers. Each NEXUS (UBA) register address space occupies 8 KB (16 pages of 512 bytes/page). This address space contains all control and status registers of the UBA, registers required for UNIBUS interrupt fielding, and registers required for mapping UNIBUS device transfers to the SBI address space. The second address space is the UNIBUS address space associated with the UBA. The UNIBUS address space occupies a total of 256 KB (512 pages of 512 bytes/page). Figure 17-3 illustrates the SBI I/O address space.

SBI ACCESS TO UNIBUS ADDRESS SPACE

The UNIBUS Address Space (248 KB of memory space and 8 KB of device register space) is accessible as part of the SBI I/O Address

Space. The UBA translates SBI command/addresses to UNIBUS command/addresses, thereby giving the software the ability to read and write UNIBUS device registers using word type memory reference instructions (MOVW, BITW, etc.).

Device Registers are assigned I/O addresses within the UNIBUS Address Space spanning 760000_8—777777_8. In VAX-11/780 physical byte address terms, the device registers occupy address space $201XE000_{16}$—$201XFFFF_{16}$. The hexadecimal digit 3,7,B or F_{16} is substituted in place of the X value within the physical address, depending upon which one of four UNIBUS address spaces the UBA is configured for. Table 17-2 illustrates the UNIBUS device register address structure.

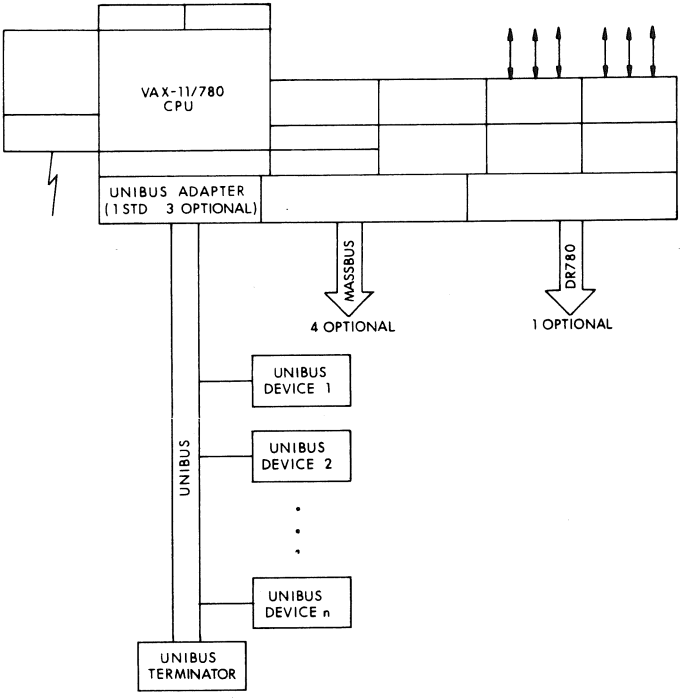


Figure 17-2 UNIBUS Subsystem

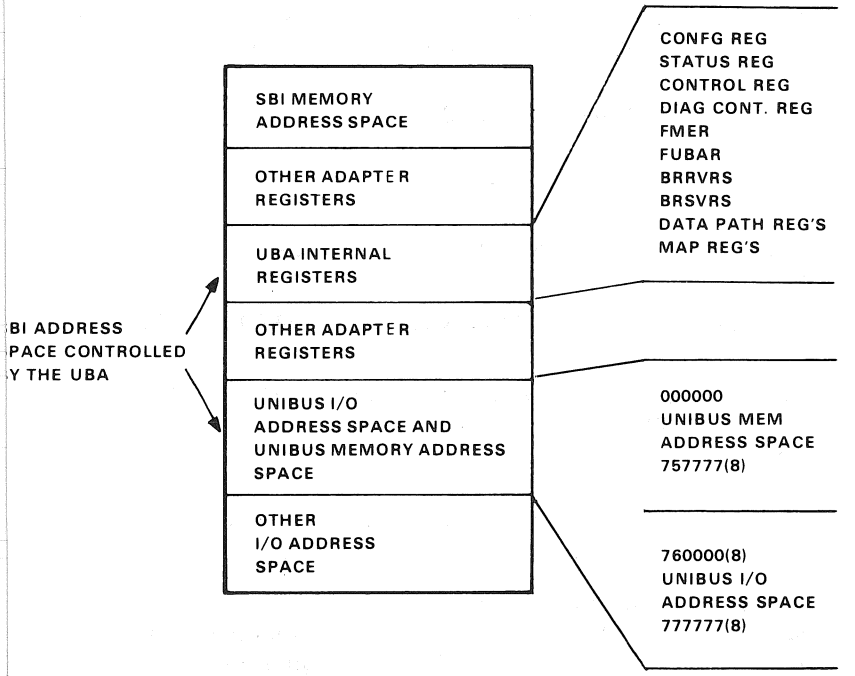


Figure 17-3 SBI I/O Address Space

Table 17-2 UNIBUS Device Address Space

UNIBUS I/O ADDRESS SPACE	UNIBUS ADDRESS (OCTAL)	PHYSICAL BYTE LOCATIONS (HEX)
UNIBUS 0 Address Space	760000-777777	2013E000-2013FFFF
UNIBUS 1 Address Space	760000-777777	2017E000-2017FFFF
UNIBUS 2 Address Space	760000-777777	201BE000-201BFFFF
UNIBUS 3 Address Space	760000-777777	201FE000-201FFFFF

Table 17-3 illustrates the translation of SBI to UNIBUS transfer operations involved in accessing the UNIBUS address space.

Table 17-3 CPU-Initiated Transfer

SBI FUNCTION	TRANSFER DIRECTION	UNIBUS FUNCTION
Read-masked (word or byte)	device to UBA	DATI
Write-masked (word or byte)	UBA to device	DATO or DATOB
Interlock Read-masked then Interlocked Write-masked	device to UBA then UBA to device	DATIP then DATO or DATOB

During such transfers, the UNIBUS adapter becomes the highest priority UNIBUS Non-Processor request (NPR) device.

Address and Function Translation

Figure 17-4 shows the SBI command/address format for accessing the UNIBUS address space for UBAs 0 through 3. Each SBI address (longword address) covers two 16-bit UNIBUS addresses (word addresses). In addition to the SBI address being decoded, the SBI function and byte mask is decoded to determine the word or byte to be accessed. The SBI to UNIBUS address and command translation is shown below.

Table 17-4 illustrates the translation from SBI Mask and Function fields to UNIBUS Control and Address fields.

Only the function byte mask combinations shown will be valid. All other function byte mask combinations addressed to the UNIBUS address space will be given an ERR confirmation. The UNIBUS address space will respond only to word or byte SBI references. Note that extended transfers cannot be made to either the UNIBUS address space or the UNIBUS adapter Registers.

The translation from SBI Mask and Function to UNIBUS control and byte address is handled by the UNIBUS control and byte address encoder illustrated in Figure 17-4.

When the VAX-11 software initiates a data transfer, reading from or writing to a UNIBUS device register, the UBA will recognize the ad-

dress as being an address within the UNIBUS address space and will pass the lower 16 SBI address bits through to the UNIBUS as UNIBUS address bits UA <17:2>. The UNIBUS adapter generates UNIBUS address bits UA <1:0> and control bits C <1:0> by decoding the SBI mask and function bits. Table 17-5 shows the relationship of the UNIBUS space controlled by UBA #0 to the SBI address space.

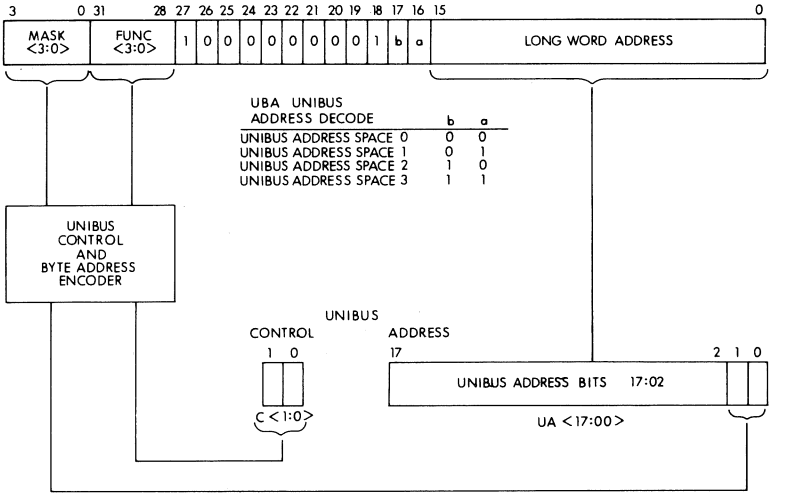


Figure 17-4 SBI to UNIBUS Control Address Translation

SBI to UNIBUS Transfer Failures

If, during a read sequence to the UNIBUS address space, data is received from the UNIBUS device with UNIBUS PB asserted (UNIBUS Device Parity Error) then the data will be sent to the SBI as a Read Data Substitute.

If, for some reason, an access is made to the UNIBUS address space and the transfer is not completed on the UNIBUS (i.e., nonexistent device), the following will occur:

1. An all-zeroes word will be sent as a read data for a read transfer.
2. The UNIBUS address bits <17:2> will be stored in the Failed UNIBUS Address Register(FUBAR).
3. The bit indicating the cause of failure (UBA Select Time Out or SSYN Time Out) will be set in the UBA Status Register. Note that in the case of a Write Transfer to the UNIBUS, the error bit is set at

least 13 μs after the command was issued and acknowledged by the UBA. It will therefore not be immediately known to the software. If the software has set the SUFFIE (SBI to UNIBUS Error Interrupt Enable), the setting of UB select Time Out or SSYN Time Out will initiate an adapter interrupt request (13 μs for SSYN timeout, 50 μs for select Time Out).

This method gives the VAX software an opportunity to exit gracefully from a transfer failure rather than being trapped out of a program due to a Read Data timeout. The method is also consistent for read and write failures.

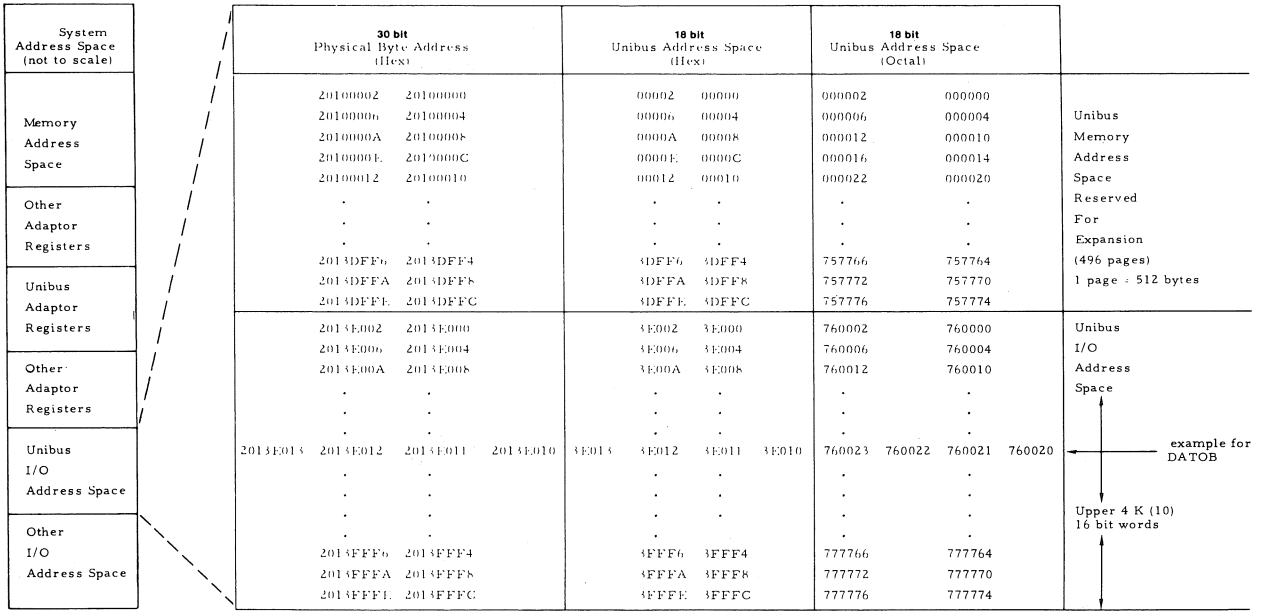
SBI		Unibus	
Function <3:0>	Mask 3 2 1 0	Control C<1:0>	Address UA<1:0>
Read Mask	0 0 0 1	DATI	0 0
	0 0 1 1	DATI	0 0
	0 0 1 0	DATI	0 0
	0 1 0 0	DATI	1 0
	1 1 0 0	DATI	1 0
	1 0 0 0	DATI	1 0
Write Mask	0 0 0 1	DATOB	0 0
	0 0 1 0	DATOB	0 1
	0 1 0 0	DATOB	1 0
	1 0 0 0	DATOB	1 1
	0 0 1 1	DATO	0 0
	1 1 0 0	DATO	1 0
Interlock Read Mask (Sets Interlock Flip Flop for DATIP-DATO Sequence)	0 0 0 1	DATIP	0 0
	0 0 1 0	DATIP	0 0
	0 1 0 0	DATIP	1 0
	1 0 0 0	DATIP	1 0
	0 0 1 1	DATIP	0 0
	1 1 0 0	DATIP	1 0
Interlock Write Mask	0 0 0 1	DATOB	0 0
	0 0 1 0	DATOB	0 1
	0 1 0 0	DATOB	1 0
	1 0 0 0	DATOB	1 1
	0 0 1 1	DATO	0 0
	1 1 0 0	DATO	1 0

Table 17-4 SBI Function-Mask Translation To UNIBUS Control-Address

UNIBUS ACCESS TO THE SBI ADDRESS SPACE

UNIBUS initiated transfers to UNIBUS memory addresses are mapped by the UBA to SBI addresses on a page-by-page basis, allowing UNIBUS data transfers to discontiguous pages of SBI memory. The SBI uses a 30-bit addressing scheme and a 32-bit wide data path, while the UNIBUS uses an 18-bit addressing scheme and a 16-bit data path.

Table 17-5 UNIBUS and SBI Address Space



Note: These addresses refer to UBAO.

The SBI is synchronous, supporting a maximum of 16 NEXUSs while UNIBUS functions are asynchronous, supporting a large number of devices.

The UNIBUS adapter accepts one of two forms of input from the UNIBUS:

- Hardware-generated interrupts
- Direct memory access transfers

Terminal input, for example, is an interrupt-driven process in which the DZ-11 (terminal interface) initiates an interrupt sequence. The interrupt service routine for the terminal driver will accept and process the data resulting from the terminal input. This process is therefore classified as a non-direct memory transfer.

In contrast, once initiated by the software, an RK06 disk will transfer its data directly to or from SBI memory via the UBA without processor intervention. The RK06, therefore, is a direct memory access (DMA) device. The direct memory access transfer may be further divided into two groups:

- Random access—access of noncontiguous addresses
- Sequential access—access of sequentially increasing addresses

The UNIBUS adapter can channel data through any one of 16 data paths for UNIBUS devices performing DMA transfers. The UBA provides a direct data path to allow UNIBUS transfers to random SBI addresses. Each UNIBUS transfer through the direct data path is mapped directly to an SBI transfer, thereby allowing only one word of information to be transferred during an SBI cycle. The UBA provides 15 buffered data paths (BDP), each of which allows a sequential access device on the UNIBUS (a device that transfers to consecutive increasing addresses) access to the SBI in a more efficient manner than that offered by the direct data path. Each of the BDPs stores data for the UNIBUS, so that four UNIBUS transfers are performed for each SBI transfer, making more efficient use of the SBI and memory. Using the BDPs, the UBA can support high-speed DMA block transfer devices such as the RK06 disk subsystem and the DMC-11. The Buffered Data Paths also allow a UNIBUS device to operate on random longword aligned 32-bit data.

UNIBUS to SBI Address Translation

The UNIBUS adapter provides for direct memory access transfers to main memory via the memory controllers connected to the Synchronous Backplane Interconnect. The UNIBUS adapter translates UNIBUS memory addresses to SBI addresses through a UNIBUS to SBI address translation map. The UNIBUS adapter physically contains

496 (decimal) hardware map registers utilized in mapping UNIBUS memory page addresses to SBI page addresses (longwords). Each map register is assigned an SBI longword address. The map register contains the SBI page address and the data path required to transfer data between the UNIBUS and the SBI.

Each UNIBUS address is mapped to an SBI address in three sections:

1. SBI page address (one page equals 512 bytes).
2. Longword within an SBI page (one longword equals four bytes).
3. Word or byte within a longword.

NOTE

To avoid confusion between UNIBUS and SBI address bits, UNIBUS address bits will be shown as UA <bit num> and SBI address bits will be shown as SA <bit num>.

As illustrated in Figure 17-5, the UNIBUS to SBI page map translates UNIBUS memory page addresses to any SBI page address. The map allows the transfer of data to discontiguous pages of SBI memory. The map translates the nine UNIBUS page address bits (UA<17:9>) to the 21 SBI page address bits (SA<27:7>).

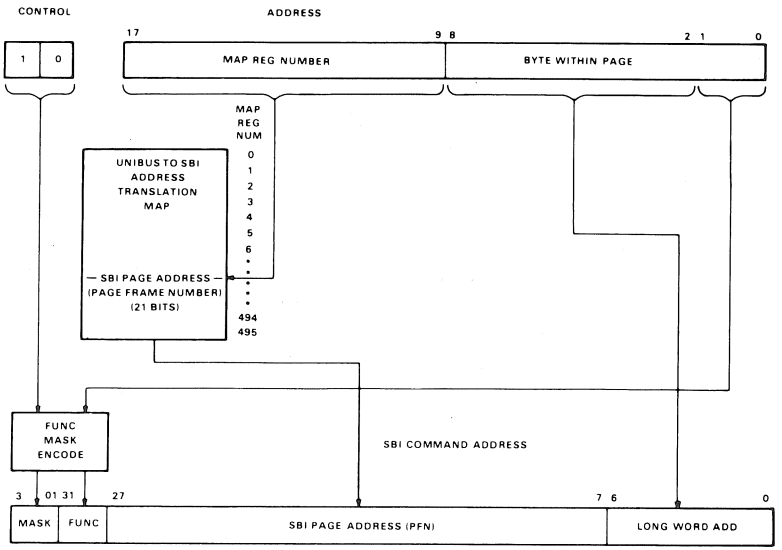


Figure 17-5 UNIBUS to SBI Address Translation

There are 496 map registers provided to map the entire UNIBUS memory address space at once, thereby reducing the problem of register allocation. Each map register corresponds to the UNIBUS page which is to be mapped. The map registers are available to the VAX-11 software as part of the SBI I/O address space. These registers are discussed in detail in the section titled SBI ADDRESSABLE UNIBUS ADAPTER REGISTERS.

UNIBUS address bits UA <08:02> determine the longword within a page and are seen by the SBI as address bits SA <06:00>. These seven bits are concatenated with the mapped page address to form the 28-bit SBI address.

The two low order UNIBUS address bits (UA<01:00>) and the two control bits (C<1:0>) determine the SBI function and byte mask (F<3:0>, M<3:0>).

The mask field points to either one or two bytes within the longword address. The function field selects either read or write and the associated qualifier. The mask and the function fields are illustrated in the following table. Table 17-6 illustrates the translation from the UNIBUS control and byte address fields to the SBI function and mask fields.

UNIBUS ADAPTER DATA TRANSFER PATHS

Data is transferred between the UNIBUS and the SBI through one of the 16 data paths of the UNIBUS adapter:

1. The direct data path (DDP) translates each UNIBUS data transaction (DATI, DATIP, DATO, DATOB) directly to an SBI function for each UNIBUS word (or byte) transfer, thereby transferring data between SBI memory and a UNIBUS device in 16-bit quantities.
2. The Buffered Data Paths allow fast, sequential access UNIBUS devices to access the SBI in a more efficient manner than is offered by the Direct Data Path. Each buffered data path (BDP1-15) accumulates data and transfers the data as words or bytes to or from the UNIBUS device. The BDPs perform quadword transfers (64 bits) to SBI memory addresses. The BDPs will respond to UNIBUS DATI, DATO, and DATOB functions but will not respond to the DATIP function.
3. The Buffered Data Paths also allow a UNIBUS device to operate on random 32-bit longword-aligned data.

The data path to be used by a particular device is assigned by the software when setting up the map registers. The data paths are numbered from DP0 to DP15. DP0 is the direct data path (DDP) and DP1 through DP15 are the buffered data paths, BDP1 through BDP15

respectively. One or more transferring UNIBUS devices can be assigned to DP0. No more than one transferring UNIBUS device, however, can be assigned to any one of the BDPs at any time. If, during a DMA transfer, the UNIBUS address points to an invalid map register or a map register that has a parity error within the high order 16 bits, the UNIBUS transfer will be aborted (SSYN Timeout in the UNIBUS device), and the bit indicating the problem will be set in the UBA status register (IVMR or MRPF). Note that for this implementation, the low order 16 bits of the map register are accessed only when an SBI transfer is required, and only at that time is parity checked on the low 16 bits of the map register.

Direct Data Path (DDP)

The Direct Data Path (DP0) translates each UNIBUS data transfer function (DATI, DATO, DATOB) to a unique SBI function (Read Mask, Write Mask). The DDP can transfer words or bytes directly between the UNIBUS and SBI memory. In addition, the DDP allows a UNIBUS device to interlock its operation with the system by translating a DATIP-DATO/DATOB UNIBUS sequence to an Interlock Read Mask Interlock Write Mask SBI sequence, thereby setting and clearing the memory interlock.

Each UNIBUS word (or byte) transfer is translated by the UNIBUS adapter to an SBI transfer. The UNIBUS transfer does not complete until the SBI transfer has been completed. The SBI address, function and byte mask are mapped directly from the UNIBUS address and control lines, and the state of an internal interlock flip flop in the case of a DATIP-DATO sequence.

Use of the Direct Data Path

- The Direct Data Path can be assigned to more than one transferring UNIBUS device.
- The DDP must be used by any device wanting to execute an interlock sequence (DATIP-DATO/DATOB) to the SBI.
- The Direct Data Path must be used by devices not transferring to consecutive increasing addresses or devices that mix read and write functions.
- The maximum throughput via the DDP is approximately 425 K words per second for write operations and 316 K words per second for read operations. These rates will decrease as other SBI activity increases.
- The DDP is the simplest data path, as far as programming goes, since the map registers are the only UNIBUS adapter registers required to be accessed when initiating a UNIBUS device transfer.

Table 17-6 UNIBUS Field To SBI Field Translation

UNIBUS CONTROL C<1:0>	UNIBUS BYTE ADDRESS A<1:0>	SBI FUNCTION FUNC<3:0>	SBI MASK M<3:0> 3210
DATI	0 0	READ MASK	0 0 1 1
	1 0		1 1 0 0
DATO	0 0	WRITE MASK	0 0 1 1
	1 0		1 1 0 0
DATOB	0 0	WRITE MASK	0 0 0 1
	0 1		0 0 1 0
	1 0		0 1 0 0
	1 1		1 0 0 0
DATIP	0 0	INTERLOCK READ MASK	0 0 1 1
	1 0		1 1 0 0
followed by		INTERLOCK WRITE MASK	
DATO	0 0		0 0 1 1
	1 0		1 1 0 0
OR			
DATOB	0 0	INTERLOCK WRITE MASK	0 0 0 1
	0 1		0 0 1 0
	1 0		0 1 0 0
	1 1		1 0 0 0

Table 17-7 illustrates the translation of UNIBUS to SBI data transfer operations.

Table 17-7 UNIBUS-Initiated Transfer Via the Direct Data Path

UNIBUS FUNCTION	TRANSFER DIRECTION	SBI FUNCTION
DATI	UBA to device	Read-masked (16 bits)
DATO or DATOB (byte)	device to UBA	Write-masked (8 or 16 bits)
DATIP then DATO or DATOB	UBA to device then device to UBA	Interlock Read-masked then Interlock Write-masked

Buffered Data Path (BDP)

There are 15 Buffered Data Paths, DP1-DP15. The Buffered Data Paths are provided for the following reasons:

1. To be used by fast DMA block transfer devices such as the RK07. The BDPs allow UNIBUS devices to make more efficient use of the SBI and memory and therefore improve system performance. The use of BDPs improves the effective UNIBUS bandwidth. The maximum throughput via the BDP is 695 K words per second for both read and write operations. This rate will decrease as other SBI activity increases.
2. To enable word-aligned block transfer devices to begin and end on an odd byte of SBI memory. (Byte offset operation will be discussed under Byte Offset Data Transfers).
3. To allow a UNIBUS device to operate on random longword-aligned 32-bit data from SBI memory so that all 32 bits of the longword are read or written at the same time.

The software assigns a UNIBUS Transfer to a Buffered Data Path when it sets up the map registers corresponding to the transfer.

The software must assure that no more than one active transfer is assigned to a particular BDP at any time.

A UNIBUS device transfer using the Buffered Data Path must have the following properties:

1. It must be a block transfer. (A block is greater than or equal to one byte). BDP maintenance (purge) will be initiated by the software following each block transfer. The purge operation is a software-initiated function of the UBA that clears the BDPs of any remaining bytes of data. These bytes will be transferred to SBI Memory for UNIBUS to Memory Write operations or cleared for UNIBUS to Memory Read Operations.
2. All transfers within a block must be to consecutive increasing addresses.
3. All transfers within a block must be of the same function type, Memory Read (DATI) or Memory Write (DATO or DATOB). The DATIP UNIBUS function will not be recognized by the BDP. A SSYN Timeout will result in a device attempting a DATIP to a BDP.

Each BDP contains eight bytes of DATA buffering, forming a quadword-aligned memory image. DATA is transferred between the UNIBUS and a BDP as words or bytes. Data is transferred between the BDP and SBI memory as quadwords or between the BDP and an SBI I/O register as longwords.

The Buffered Data Paths are transparent to the UNIBUS device. The

device will perform its transfer as if transferring directly to memory. The operation of the BDPs is described in the following section.

UNIBUS Data Transfers to Memory

As a UNIBUS device transfers data to memory (DATO,DATOB) via a BDP, the BDP will store the data and complete the UNIBUS cycle. The Buffered Data Paths are implemented so that a quadword image is formed in the BDP before an SBI cycle is initiated. When the UNIBUS device addresses the last byte or word of a physical quadword, the UBA will complete the data cycle and the BDP will perform an extended write operation, thereby transferring the stored bytes of data. The SBI transfer will be completed before recognizing additional UNIBUS transfers. The BDP will set its Buffer Not Empty (BNE) bit whenever a UNIBUS Write to the BDP is performed, and clear the BNE bit each time the SBI transfer is executed. The BNE bit indicates whether or not valid data is contained in the BDP. Figure 17-6 illustrates a Buffered Data Path transfer. In this illustration, a Buffered Data Path transfer of four 16-bit data words to the Buffered Data Path takes place. The fourth data transfer initiates the extended write transfer of all 64 bits to memory.

The BDP stores the UNIBUS address of data contained in the BDP. The BDP stores the UNIBUS address of the current transfer in order to transfer the remaining bytes to memory at the end of a block transfer. This is the purge function that will be discussed in a later section.

The BDP also stores the type of function and the state of each byte of the data buffer (buffer state). The buffer state is transmitted as the SBI mask bits during the BDP to SBI write cycle so that only the correct bytes will be written into memory.

UNIBUS Data Transfers From Memory

As a UNIBUS device performs Memory Read operations (DATI) via a BDP, the BDP tests the state of its data buffers. If the buffers do not contain data for the UNIBUS transfer, the BDP will initiate an Extend Read operation to memory. The BDP will then transfer data for the current cycle to the UNIBUS, thereby completing the UNIBUS cycle, store the remaining bytes in its buffers, and set the BNE bit. If the data for the current UNIBUS cycle is available in the data buffers, then the BDP will pass the data to the UNIBUS and complete the cycle. The BDP will prefetch the next quadword of data (Extended Read Transfer) after each UNIBUS access to the last word of a quadword-aligned group. The Buffer Not Empty (BNE) bit is cleared by the BDP before the prefetch and set when the Read Data returns, thereby indicating the state of the BDP. Figure 17-7 illustrates the Buffered Data Path transfer from memory to the UNIBUS.

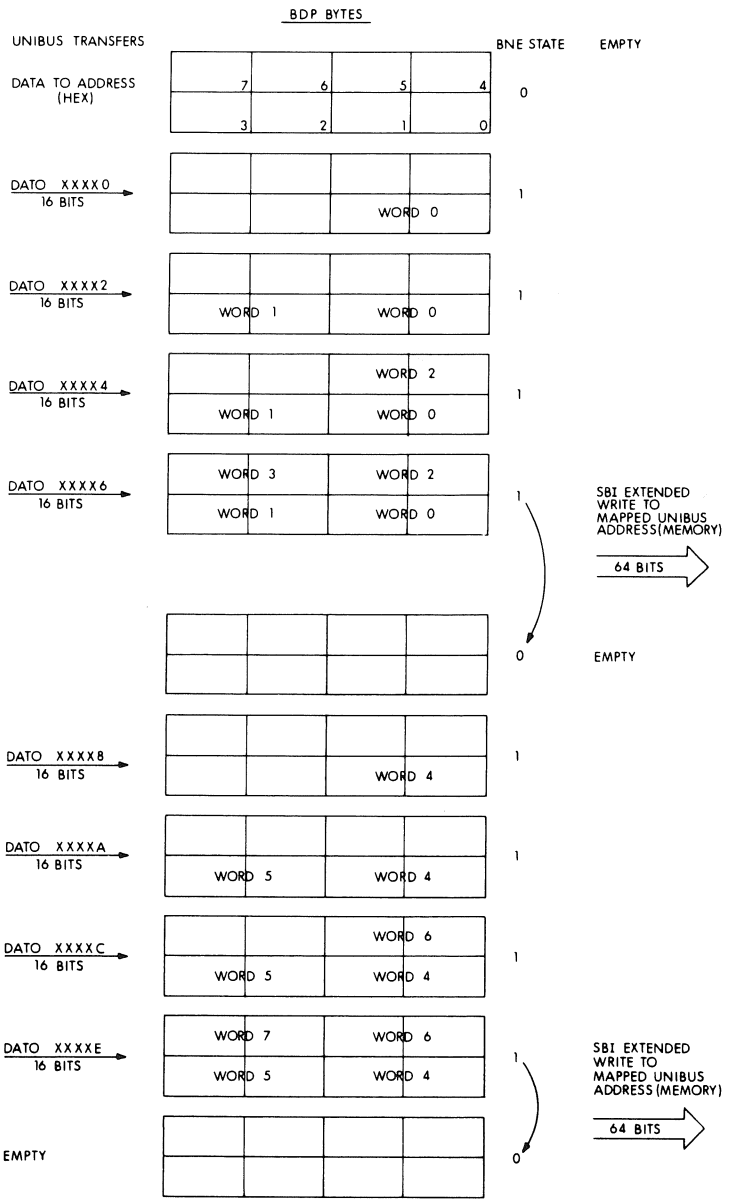


Figure 17-6 UNIBUS Transfer to Memory

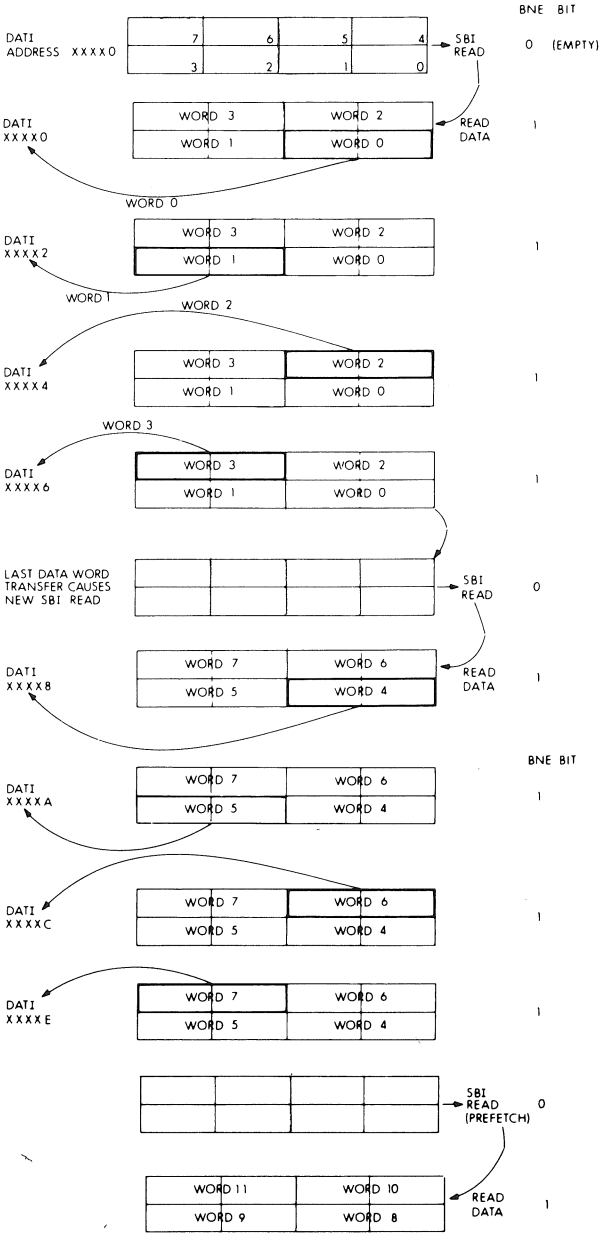


Figure 17-7 UNIBUS Transfers From Memory

Programming Note:
Since the prefetch allows the possibility of the UBA crossing a page boundary into nonexistent memory, resulting in a 100 μsec timeout, it is recommended that the software allocate an additional map register following a block. This map register must be invalidated. When the prefetch crosses this page boundary to the invalid map register, the prefetch will be aborted immediately, thereby eliminating the 100 μsec timeout. The UBA does not record any UBA or SBI errors that may occur during the prefetch operations since this is an anticipatory function based on the next probable address. If an error does occur then the prefetch will be aborted and the BDP will not be filled with data. If the UNIBUS device accesses the same BDP again, then the BDP to SBI read will be initiated and any errors that occur will be logged at this time.

Byte Offset Data Transfers
The BDPs enable word-aligned UNIBUS devices (devices beginning transfers on word boundaries and transferring an integral number of words) to begin and end a block transfer at an ODD byte of SBI memory. To use this feature, the software will set the Byte Offset bit of the map registers involved during the devices transfer.

When the Byte Offset bit is set for a transfer using the BDPs, the BDP will, in effect, increase the SBI memory address by one byte. The data will apppear on the UNIBUS in the byte or word indicated by the UNIBUS Address. The data will appear on the SBI shifted to the left (increased) by one byte. The UNIBUS adapter will distribute the data, and adjust the SBI address and byte mask so that the data will get to or come from the correct memory location. This operation is transparent to the UNIBUS device.

Figure 17-8 shows the relative position of data being transferred between a UNIBUS device and SBI memory. Figure 17-8 top shows the relative positions without Byte Offset and Figure 17-8 bottom shows the position with Byte Offset.

Purge Operation
The purge operation is a software-initiated function of the UBA in which the Buffered Data Paths are purged of data and initialized. The Buffered Data Path used by a UNIBUS device must be purged at the completion of the device's transfer. The software initiates the purge by writing a one to the BNE bit of the data path register (DPR) corresponding to the Buffered Data Path to be purged. The UBA will perform the following, depending on the transfer function that was being performed by the BDP:

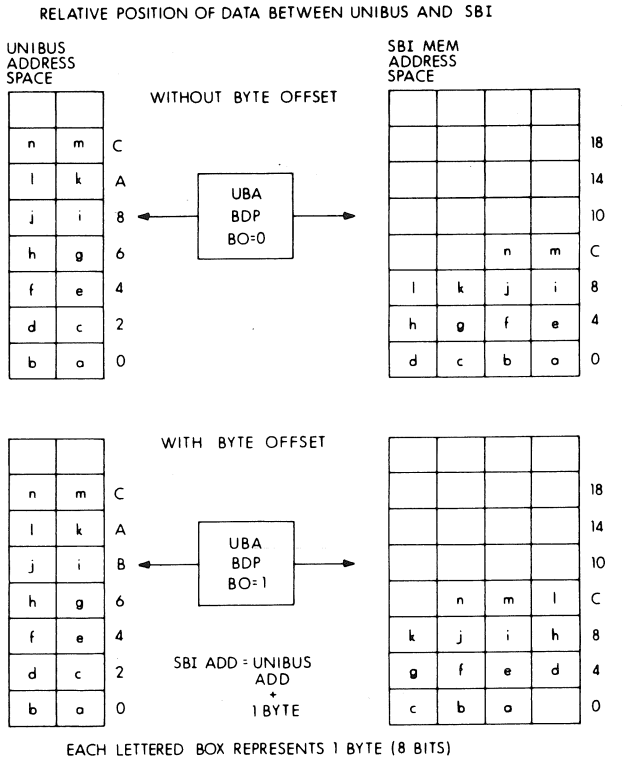


Figure 17-8 Relative Position of Data Between UNIBUS and SBI

1. Writes to memory. If there are any remaining bytes of data in the BDP, this data will be transferred to memory. The UBA will then clear the BNE bit, function bit and buffer state bits and leave the BDP in its initialized state. If an error occurs during this transfer, the Buffer Transfer Error bit of the data path register will be set, indicating that the data was not successfully transferred to memory. Software must clear this bit before the BDP can be used again.

   If there were no data remaining in the Buffered Data Path, then the buffer is left in its initialized state.

2. Reads from memory. The UBA will initialize the BDP by clearing the BNE bit of the DPR.

Longword-Aligned 32-Bit Random Access Mode

The UNIBUS adapter can be used in a mode so that a UNIBUS device

can operate on random longword-aligned 32-bit quantities without requiring purge operations. This mode is selected by setting the longword access enable (LWAE) bit 26 of the map register corresponding to the UNIBUS transfer. A Buffered Data Path must be selected for this operation.

In this mode, a UNIBUS device must first operate on the low order word of the longword and then the high order word. An operation is considered to be a read from memory (DATI) or a write to memory (DATO) or a read/write (DATI/DATO). The UNIBUS DATIP function code is not valid for transfers using Buffered Data Paths, and any device performing the DATIP through a Buffered Data Path will receive an SSYN timeout (NXM).

The Buffered Data Path will not perform the prefetch operation when this mode is enabled, thereby allowing for random access of longword-aligned 32-bit quantities. This mode eliminates the need for the purge operation at the completion of the transfer, providing the UNIBUS device operates on both words of the longword and operates on them in order (i.e., low word, then high word).

Maximum throughput in this mode is approximately 1.7 Mbyte/sec as illustrated in Figure 17-9.

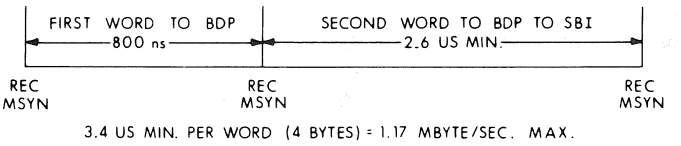


Figure 17-9 Random Access Mode Throughput

The operation of the UBA for the longword-aligned 32-bit access mode is determined by the function (DATI, DATO/DATOB) and address (A1, A0) received from the UNIBUS and the state of the buffer not empty (BNE) bit of the data path register, corresponding to the Buffered Data Path being used for this operation, within the UBA. (BNE SET = buffer not empty, BNE CLEAR = buffer empty).

The following statements summarize the operation of the UBA for the longword-aligned 32-bit random access mode of operation.

DATI Functions

1. SBI reads will occur when a DATI operation is received and the Buffered Data Path is empty (BNE = 0).
2. The BNE bit will be set in response to a successful SBI read

generated by a DATI operation to the low order word (A1 = 0). Longword data from memory is stored in the Buffered Data Path.

3. The BNE bit will be cleared by a DATI operation to the high order word (A1 = 1).
4. If the BNE bit is set, data from the Buffered Data Path will be returned to the UNIBUS device.

DATO/DATOB Functions

1. The BNE bit will be set by a DATO or DATOB operation. The data from the UNIBUS device will be stored in the Buffered Data Path and the byte mask bit is set within the data path register to indicate the bytes or words that have been written by the UNIBUS device.
2. SBI writes will occur when a DATO operation occurs to the high order word or when a DATOB operation occurs to the high order byte. The bytes or words that were written (i.e., those for which the byte mask bits are set) are written into main memory.
3. The BNE bit will be cleared after a SBI write operation.

The UBA operations per UNIBUS access, as a function of BNE and received UNIBUS function and address for this mode of operation are:

Present BNE State	Function	A1,A0	UBA Operations	Next BNE State
0	DATI	0 X	SBI READ,RETURN LOW WORD, STORE DATA	1
0	DATI	1 X	SBI READ, RETURN HIGH WORD	0
1	DATI	0 X	RETURN LOW WORD	1
1	DATI	1 X	RETURN HIGH WORD	0
0	DATO	0 X	STORE LOW WORD	1
0	DATO	1 X	STORE HIGH WORD,SBI WRITE	0
1	DATO	0 X	STORE LOW WORD	1
1	DATO	1 X	STORE HIGH WORD,SBI WRITE	0
0	DATOB	0 0	STORE BYTE 0	1
0	DATOB	0 1	STORE BYTE 1	1
0	DATOB	1 0	STORE BYTE 2	1
0	DATOB	1 1	STORE BYTE 3,SBI WRITE	0
1	DATOB	0 0	STORE BYTE 0	1
1	DATOB	0 1	STORE BYTE 1	1
1	DATOB	1 0	STORE BYTE 2	1

1	DATOB	1	1	STORE BYTE 3,SBI WRITE	0
0	DATIP	X	X	UBA DOES NOT RESPOND (NXM TO UNIBUS DEVICE) NO CHANGE	0
1	DATIP	X	X	UBA DOES NOT RESPOND (NXM TO UNIBUS DEVICE) NO CHANGE	1

To enable this mode of operation, Bit 26 of the map register has been changed to the Longword Access Enable (LWAE) bit. This bit when set and when a buffered data path is selected, will enable the longword-aligned 32-bit random access mode. It is a read/write bit and is cleared on initialization.

Programming the UBA for longword-aligned random access mode requires loading the map registers with the following data:

BIT<31>	MRV	Map register valid, must be set.
BIT<30:27>		Must be zero.
BIT<26>	LWAE	Longword access enable, must be set. Ignored during Direct Data Path transfers.
BIT<25>	BO	Byte offset, must be zero.
BIT<24:21>	DPDB	Data path designator bits, must use a buffered data path, BDP1-BDP15. LWAE bit is ignored when DPDB = 0 (Direct Data Path).
BIT<20:0>	PFN	Page frame number, SBI page address.

The allowed UNIBUS sequences for this mode of operation are:

		A1,A0	
1.	DATI	0 0	SBI READ—Low word is returned to UNI BUS device. Both words are stored in BDP. BNE bit is set.
	DATI	1 0	High word from BDP is returned to UNIBUS device. BNE is cleared.
2.	DATO	0 0	Low word is written to BDP. BNE bit is set.
	DATO	1 0	SBI WRITE—High word is written to BDP—then low word and high word are transferred to memory. BNE bit is cleared.

3. DATOB	0	0	Byte 0 is written to BDP, BNE is set.
DATOB	0	1	Byte 1 is written to BDP, BNE is set.
DATOB	1	0	Byte 2 is written to BDP, BNE is set.
DATOB	1	1	Byte 3 is written to BDP, SBI WRITE.
4. DATI	0	0	SBI READ-Low word is returned to UNIBUS device. Both words are stored in BDP.
DATO	0	0	Low word of BDP is written by UNIBUS device.
DATI	1	0	High word from BDP is returned to UNIBUS device.
DATO	1	0	SBI WRITE-High word of BDP is written by UNIBUS device and modified longword is returned to the memory.
DATO	1	0	SBI WRITE-High word of BDP is written by UNIBUS device and modified longword is returned to the memory.

Additional BDP Software Information

1. For purge operations in which data is transferred to memory, the SBI transfer takes about 2 μs. The UBA will not respond to Data Path Register Read during this period (Busy Confirmation), thus preventing a race condition when testing for the BNE bit to be cleared.
2. The Buffer Transfer Error bit (BTE) of the data path registers indicates that an error occurred during an operation involving a buffered data path. Once this bit is set, UNIBUS transfers using the BDP will be aborted until the bit is cleared by the software. The purge operation does not clear the BTE bit.
3. Any purge operations initiated by the software to BDPs for which the purge or initialization is not required are treated by the UBA as a NO-OP.
4. A purge operation to Data Path Register 0 (Direct Data Path) is treated by the UBA as a NO-OP.

INTERRUPTS

SBI interrupts can be generated from two sources within the UNIBUS subsystem: either from a UNIBUS device or from the UNIBUS adapter.

Interrupts from the UNIBUS can occur at any one of the four request levels, as determined by the UNIBUS BR lines. Interrupts from the UNIBUS adapter will occur at one assigned request level. This level is assigned by backplane jumper.

The UNIBUS adapter contains one request sublevel. The UBA will therefore require four of the 64 possible SBI interrupt vectors (1 for each of the 4 required levels). The four vectors will each "point" to a UBA Service Routine corresponding to an interrupt request level. Each UBA service routine must read and test the BR Receive Vector Register corresponding to the level of interrupt:

BRRVR 7 for Req Level 7
BRRVR 6 for Req Level 6
BRRVR 5 for Req Level 5
BRRVR 4 for Req Level 4

From the contents of the BRRVRs, the UBA service routine will determine whether the interrupt was generated from within the UBA Status Register, from the UNIBUS device,or from both. The UBA service routine can then service the interrupt as determined by testing the contents of the BRRVR.

Bit <31>	Bits <15:0>	
0	0	No service required.
0	V	UNIBUS service as indicated by vector V received from the UNIBUS device (UNIBUS device Interrupt Service Routine).
1	0	UNIBUS adapter service required. Read configuration register and status register to determine the service required.
1	V	UNIBUS and UNIBUS adapter service required. 1. Save the vector V (received from the UNIBUS device). 2. Read UBA configuration register and status register. 3. Perform UBA service as indicated by configuration and status register. 4. Index into UNIBUS device service routine with vector V.

V is the vector field of the BRRVR received from the UNIBUS device. Zero is the null vector indicating that a vector was not received from the UNIBUS device.
Software Note: The zero vector resulting from an SBI Interrupt Summary Read must be reserved and interpreted as a Passive Release Condition.

Interrupts from the UNIBUS

The UBA will translate the UNIBUS BR interrupts to SBI request inter-

rupts, providing the Interrupt Fielder Switch (IFS) bit and the BR Interrupt Enable (BRIE) bit of the UBA control register are set. The assertion of the SBI request lines will initiate an SBI interrupt transaction vectoring to the UBA interrupt service routine. This routine will then read the BR Receive Vector Register (BRRVR) corresponding to the level of the interrupt. On receiving the read BRRVR command, the UBA will test that the following conditions are true:

1. The UNIBUS BR line corresponding to the BRRVR number is asserted.
2. The BRRVR does not contain an already valid vector.
3. UNIBUS AC LO is not asserted.

If all of the three conditions are met, then the UBA will issue the UNIBUS Bus Grant and complete the UNIBUS interrupt transaction. The BRRVR is loaded with the interrupt vector by the successful completion of the interrupt transaction. The device vector received during the transaction will be sent as the read data to the BRRVR Read Command. If a UBA interrupt is active then the vector will be sent as a negative quantity (bit 31 sent as a one).
The BRRVR is cleared by the successful completion of the SBI Read Data Cycle, otherwise the vector is saved and the BRRVR remains full.
If conditions 1, 2, 3 are not met then the contents of the BRRVR (either the stored vector, from a previously failing SBI read data cycle, or zero) will be sent as read data. If a UBA interrupt is active then bit 31 will be sent as a one.
The following sequence is performed for UNIBUS device interrupts:

1. A bus request line is asserted by the UNIBUS device.
2. The UNIBUS adapter asserts the SBI request line, corresponding to the UNIBUS BR line, to initiate the interrupt transaction in the CPU.
3. When the interrupt summary read corresponding to the above request level is seen by the UNIBUS adapter, the UNIBUS adapter asserts the request sublevel assigned to the UBA.
4. The CPU will then transfer control to the UNIBUS adapter interrupt service routine.
5. The UNIBUS interrupt service routine will execute a read to the BR receive vector register corresponding to the level of interrupt.
6. The UNIBUS adapter will issue the UNIBUS Bus Grant corresponding to the level of the interrupt being serviced providing the following conditions are met: adapter interrupt is not pending; BR line corresponding to the BRRVR is asserted; the BRRVR does not contain a previous vector.
7. The UNIBUS interrupt transaction is completed, the vector is

loaded into the corresponding BRRVR, and the vector is given to the UNIBUS Interrupt Service Routine as a Read DATA.

8. The BRRVR will be cleared when the ACK Confirmation is received for the Read DATA.
9. The UNIBUS interrupt service routine will then dispatch to the UNIBUS device service routine (or service the UBA) as indicated by the received interrupt vector.

NOTE

The UNIBUS adapter interrupt service routine (UBASR) is the routine that will interface the CPU interrupt process to the individual UNIBUS device service routines. This routine will provide the additional level of dispatch required for UNIBUS-initiated interrupts.

Failure to Complete the UNIBUS Interrupt Transaction

If for some reason, the UNIBUS initiated an interrupt transaction and then fails to complete (i.e., passive release), the interrupt vector will not be loaded into the interrupt vector register. The following mechanism will allow the UNIBUS interrupt service routine to gracefully exit. The idle state of the BRRVR is zero. If, when reading the interrupt vector register, the UNIBUS interrupt service routine receives the zero vector, it will log an error (if desired) and return from the service routine.

Once successfully loaded, the BRRVR will maintain the interrupt vector until an ACK confirmation to the BRRVR Read Data has been received, or an adapter Init sequence is initiated. If the ACK confirmation is not received for the Read Data then the BRRVR full bit will not be cleared, and subsequent reads to that BRRVR will result in the stored vector being returned for the Read Data until ACK is received for the Read Data.

Interrupts from the UNIBUS adapter to the SBI

When the UNIBUS adapter interrupt enable bit is set, and a condition warranting an interrupt occurs in the UNIBUS adapter, the following sequence occurs:

1. The UNIBUS adapter asserts its assigned request line.
2. When the Interrupt Summary Read, corresponding to the above request level, is seen by the UNIBUS adapter, the request sublevel assigned to the UNIBUS adapter is sent to the CPU as an Interrupt Summary Response.
3. With this information, request level and request sublevel, the CPU can dispatch to the UNIBUS adapter service routine, which will

then read the BR Receiver Vector Register corresponding to the level of interrupt. The BRRVR will contain a negative value (bit 31 set).

4. The UBA service routine will detect the negative value and branch to a routine that will read the Configuration Register and Status Register to determine the service required.

The request line will remain asserted until all conditions (bits of the UNIBUS adapter status register) have been cleared by the software.

UNIBUS ADAPTER (NEXUS) REGISTER SPACE

Each NEXUS register address space occupies 16 pages (512 bytes/page) of Synchronous Backplane Interconnect I/O address space. The address location of the UNIBUS adapter is determined by the transfer request priority number assigned to the adapter. The transfer request number is determined by electrical jumpers on the NEXUS backplane and may vary from one system configuration to the next.

Table 17-8 illustrates the physical base address and SBI base address for a NEXUS assigned to any one of the SBI transfer request numbers.

Table 17-8 Transfer Number Address Assignments

SBI Transfer Request Number	Address Base Physical (Hex)
1	20002000
2	20004000
3	20006000
4	20008000
5	2000A000
6	2000C000
7	2000E000
8	20010000
9	20012000
10	20014000
11	20016000
12	20018000
13	2001A000
14	2001C000
15	2001E000

Table 17-9 lists each of the UNIBUS adapter registers and its associated physical address offset.

Table 17-9 UNIBUS Adapter Register Address Offset

UNIBUS Adapter Register	Byte Offset (Physical Hex)
Configuration Register	000
UNIBUS Adapter Control Register	004
UNIBUS Adapter Status Register	008
Diagnostic Control Register	00C
Failed Map Entry Register	010
Failed UNIBUS Address Register	014
Failed Map Entry Register	018
Failed UNIBUS Address Register	01C
Buffer Selection Verification Register 0	020
Buffer Selection Verification Register 1	024
Buffer Selection Verification Register 2	028
Buffer Selection Verification Register 3	02C
Buffer Receive Vector Register 4	030
Buffer Receive Vector Register 5	034
Buffer Receive Vector Register 6	038
Buffer Receive Vector Register 7	03C
Data Path Register 0	040
Data Path Register 1	044
.	.
.	.
.	.
Data Path Register 14	078
Data Path Register 15	07C
Reserved	080
.	.
.	.
.	.
Reserved	7EC
Map Register 0	800
Map Register 1	804
.	.
.	.
.	.
Map Register 494	EB8
Map Register 495	EBC
Reserved	EC0
.	.
.	.
.	.
Reserved	EFC

The offset within the UNIBUS adapter address space is shown for each of the UNIBUS adapter registers with respect to the physical address. As described in Table 17-9, the addresses of all other UNIBUS adapter registers are relative to the configuration register address by an offset. The base address of the configuration register is the physical base address described in Table 17-8. Therefore, the byte offset for the configuration register in Table 17-9 is 000.

SBI ADDRESSABLE UNIBUS ADAPTER REGISTERS

The UNIBUS adapter registers occupy eight pages of the SBI I/O address space. These registers fall into four categories: map registers, data path registers, interrupt vector registers and control and status registers. The UNIBUS adapter registers are all 32-bit registers and can only be written as longwords. These registers will, however, respond to byte or word read commands. These registers will also respond to the Interlock Read/Interlock Write sequence but will not affect the interlock of the SBI. The following sections discuss the function and content of each of the UNIBUS adapter registers.

Configuration Register (CNFGR)

The configuration register contains the SBI fault bits, the UNIBUS adapter and UNIBUS environment status bits, and the UNIBUS adapter code. This register is required to interface with the SBI. Figure 17-10 illustrates the configuration register.

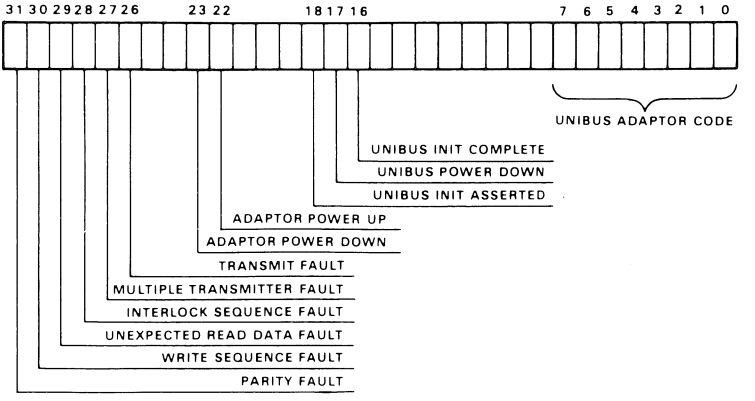


Figure 17-10 Configuration Register Bit Configuration

The contents of the Configuration Register are as follows:

Bit: 31:27 **Name:** SBI faults
Function: These bits are set when the UNIBUS adapter detects specific fault conditions on the SBI. These bits cannot be set once FAULT has been asserted. The negation of FAULT and the disappearance of the fault conditions clear the bits. The setting of any of the bits <31:26> will cause the UNIBUS adapter to assert the FAULT signal on the SBI.

Bit: 31 **Name:** Parity Fault (PAR FLT)
Function: PAR FLT is set when the UNIBUS adapter detects an SBI parity error.

Bit: 30 **Name:** Write Sequence Fault (WSQ FLT)
Function: WSQ FLT is set when the UNIBUS adapter receives a Write Masked, Extended Write Masked, or Interlock Write Masked command which is not immediately followed by the expected write data.

Bit: 29 **Name:** Unexpected Read Data Fault (URD FLT)
Function: URD FLT is set when the UNIBUS adapter receives data for which a Read Masked, Extended Read, or Interlock Read Masked command has not been issued.

Bit: 28 **Name:** Interlock Sequence Fault (ISQ FLT)
Function: ISQ FLT is set when an Interlock Write Masked command or a UNIBUS address space is received by the UNIBUS adapter without a previous Interlock Read Masked command.

Bit: 27 **Name:** Multiple Transmitter Fault (MXT FLT)
Function: MXT FLT is set when the UNIBUS adapter is transmitting on the SBI and the IB bits transmitted by the UNIBUS adapter do not match those latched from the SBI. The lack of correspondence indicates a multiple transmitter condition.

Bit: 26 **Name:** Transmit Fault (XMT FLT)
Function: XMT FLT is set if the UNIBUS adapter was the transmitter during a detected fault condition. When the software subsequently reads the configuration and status registers of each of the NEXUSs on the SBI in order to identify the source of the fault, the UNIBUS adapter will be identified as that source if bit <26> is set.

Bit: 25:24 **Name:** Reserved and zero
Function: Bits <23,22,18,17,16> are UNIBUS Subsystem Environmental Status Bits. If any of these bits are set and the Configuration Interrupt Enable bit (CNFIE) of the control register is also set, then the UNIBUS adapter will initiate an SBI interrupt request at the level assigned to the UNIBUS adapter.

Bit: 23 **Name:** Adapter Power Down (AD PDN)
Function: This bit is set when the UNIBUS adapter power supply asserts AC LO. It is cleared by writing a one to the bit location or when the Adapter Power Up bit is set.

Bit: 22 **Name:** Adapter Power Up (AD PUP)
Function: This bit is set by the negation of power supply AC LO. It is cleared by writing a one to the bit location or by setting the Adapter Power Down bit.

Bit: 21:19 **Name:** Reserved and zero
Function:

Bit: 18 **Name:** UNIBUS INIT Asserted (UB INIT)
Function: The assertion of UNIBUS Init will set this bit. It is cleared by the setting of the UNIBUS Initialization Complete bit (UBIC) or by the writing of a one to this bit location.

Bit: 17 **Name:** UNIBUS Power Down (UB PDN)
Function: This bit is set when UNIBUS AC LO is asserted. It indicates that the UNIBUS has initiated a power down sequence. The setting of the UNIBUS initialization complete bit or writing a one to this location will clear UB PDN.

Bit: 16 **Name:** UNIBUS Initialization Complete (UBIC)
Function: This bit is set by a successful completion of a power up sequence on the UNIBUS. It is the last of the status bits to be set during a UNIBUS adapter initialization sequence, and it can be interpreted to mean that the UNIBUS adapter and the UNIBUS are ready. The assertion of UNIBUS AC LO or UNIBUS INIT, or the writing of a one to this bit location will clear UBIC.

Bit: 15:8 **Name:** Reserved
Function:

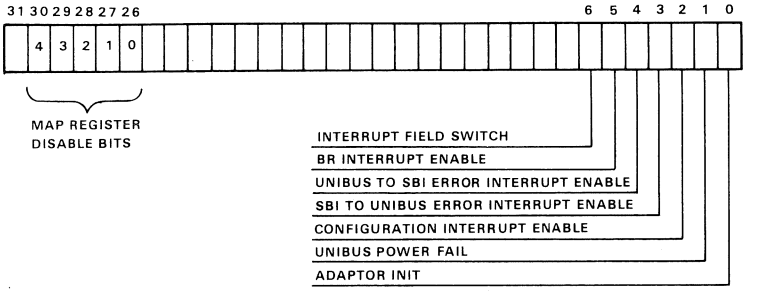
Bit: 7:0 **Name:** Adapter Code
Function: These bits define the code assigned to the UNIBUS adapter. Table 17-10 shows the bit assignment.

Table 17-10 Adapter Code Bit Assignment

BIT NUMBER	7	6	5	4	3	2	1	0
UNIBUS ADDRESS SPACE	0	0	1	0	1	0		
0							0	0
1							0	1
2							1	0
3							1	1

Adapter code bits 1 and 0 are determined by backplane jumpers and indicate the starting address of the UNIBUS address space associated with the UNIBUS adapter, as shown in Table 17-11.

Table 17-11 Selectable UNIBUS Starting Addresses



Note that the lowest two bits of the Configuration Register (Vb and Va) correspond to SBI address bits 16 and 17.

Control Register (UACR)

The UNIBUS Adapter Control Register enables the software to control operations both on the UNIBUS adapter and on the UNIBUS. All bits except for the Adapter INIT bit are set by writing a 1 and cleared by writing a 0 to the bit location. The Adapter INIT bit is set by writing a one to the bit location and is self clearing. Figure 17-11 shows the Control Register bit configuration.

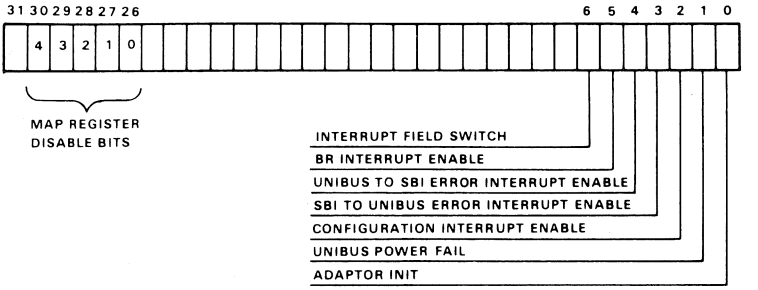


Figure 17-11 Control Register Bit Configuration

The contents of the control register are as follows:

Bit: 31 **Name:** Reserved and zero
Function:

Bit: 30:26 **Name:** Map Register Disable <4:0> (MRD)
Function: This field of five read/write bits disables map registers in groups of 16, according to the binary value contained in the field. The MRD bits prevent double addressing if UNIBUS memory is used. This field is loaded with a binary value equal to the number of 8 Kbyte units of memory attached to the UNIBUS, as shown in Table 17-12.

Table 17-12 Map Register Disable Bit Functions

MRD <4:0>	AMOUNT OF UNIBUS MEMORY (WORDS)	MAP REGISTERS DISABLED
00000	0 K	NONE
00001	4 K	0 TO 15 (10)
00010	8 K	0 TO 31 (10)
00011	12 K	0 TO 47 (10)
•	•	•
•	•	•
•	•	•
11110	120 K	0 TO 480 (10)
11111	124 K	0 TO 495 (10)

DMA transfers to addresses controlled by disabled map registers are not recognized by the UNIBUS adapter. No error bits are set and no transfers are initiated. However, SBI access to disabled map registers is permitted. The MRD field is initialized as zero, with all map registers enabled.

Bit: 25:7 **Name:** Reserved and zero
Function:

Bit: 6 **Name:** Interrupt Field Switch (IFS)
Function: This bit determines whether interrupts from a UNIBUS device on the UNIBUS outside of the UNIBUS adapter will be fielded by the VAX-11 CPU or passed to the UNIBUS inside of the UNIBUS adapter. If the bit is set (1), then the interrupt will be passed to the SBI, if the BR Interrupt Enable bit of the control register is set. If the bit is cleared (0), then the interrupt will be passed to the UNIBUS inside of the UNIBUS adapter, where it is in effect ignored.

The power up state of the IFS bit is 0. The bit is also cleared by the adapter unit and SBI dead signals. This bit and BRIE must be set by the software to receive UNIBUS device interrupts.

Bit: 5 **Name:** Bus Request Interrupt Enable (BRIE)
Function: When this bit is set it allows the UNIBUS adapter to pass interrupts from the UNIBUS to the VAX-11 CPU. The power up state of the BRIE bit is 0. The bit is also cleared by the Adapter INIT, SBI UNJAM, and SBI Dead signals. This bit and IFS must be set by the software to receive UNIBUS device interrupts.

Bit: 4 **Name:** UNIBUS to SBI Error Field Interrupt Enable (USEFIE)
Function: The USEFIE bit enables an interrupt request to the VAX-11 CPU whenever any of the following Status Register bits is set on a DMA transfer.

RDTO (Read Data Time Out)
RDS (Read Data Substitute)
CXTER (Command Transmit Error)
CXTO (Command Transmit Time Out)
DPPE (Data Path Parity Error)
IVMR (Invalid Map Register)
MRPF (Map Register Parity Failure)

The power up state of the USEFIE (UNIBUS Error Field Interrupt Enable) bit is zero. SBI UNJAM and Adapter Init will clear the bit.

Bit: 3 **Name:** SBI To UNIBUS Error Field Interrupt Enable (SUEFIE)
Function: If this bit is set, the UNIBUS adapter will generate interrupt requests to the VAX-11 CPU when one of the two bits in the SBI to UNIBUS data transfer error field is set.

UBSTO (UNIBUS Select Time Out)
UBSSYNTO (UNIBUS Slave Sync Time Out)

The power up state of this bit is zero. SBI UNJAM, SBI Dead, and Adapter INIT will clear the bit.

Bit: 2 **Name:** Configuration Interrupt Enable (CNFIE)
Function: If this bit is set, the UNIBUS adapter will initiate an interrupt request to the VAX-11 CPU whenever any one of the environmental status bits of the configuration register is set.

AD PDN (Adapter Power Down)
AD PUP (Adapter Power Up)
UB INIT (UNIBUS INIT Asserted)
UB PDN (UNIBUS Power Down)
UBIC (UNIBUS Initialization Complete)

The power up state of this bit is set (1). The bit is cleared by Adapter INIT, SBI UNJAM, and SBI Dead.

Bit: 1 **Name:** UNIBUS Power Fail (UPF)
Function: When this bit is set, it initiates a power fail sequence on the UNIBUS, asserting AC LO, DC LO, and INIT. The software uses this bit to initialize the UNIBUS. The UNIBUS will remain powered down as long as UPF is set. Thus the software can initialize the UNIBUS by setting and then clearing the UPF bit.

Bit: 0 **Name:** Adapter INIT (ADINIT)
Function: When this bit is set it will completely initialize the UNIBUS adapter and the UNIBUS. The map registers, the data path registers, the status register, and the control register will be cleared. The UNIBUS adapter will initialize all of its control logic, and will generate a power fail sequence on the UNIBUS. The adapter initialization sequence takes only 500 μs to complete, while the UNIBUS power fail sequence requires 25 ms.

Only the configuration register and the diagnostic control register can be read during the adapter initialization sequence. And only the configuration register, the diagnostic control register, and the control register can be written during the adapter initialization sequence.

Once the sequence has been completed, all UNIBUS adapter registers can be accessed. However, the UNIBUS cannot be accessed until the UNIBUS initialization sequence has been completed as well. The software can test for this condition by reading the UBIC bit of the configuration register, or by setting the configuration interrupt enable bit of the control register and looking for the interrupt generated by the setting of the UBIC bit. Note that the assertion of UNIBUS INIT (UBINIT) can also initiate an interrupt . The Adapter INIT bit can be set by writing a one to the bit location, and it is self-clearing.

Status Register (USAR)
The UNIBUS Adapter Status Register contains program status and error information. Bits <27:24> are read only bits which are set and cleared by operations within the UNIBUS adapter. Bits <10:0> can be read and cleared by writing a one to the appropriate bit location. Specific conditions which occur on the UNIBUS adapter will set these bits. Writing a zero has no effect on any of the bits. Figure 17-12 shows the Status Register bit configuration.

The contents of the status register are:

Bit: 31:28 **Name:** Reserved and zero
Function:

Bit: 27:24 **Name:** BR Receive Vector Register Full
Function: These bits indicate the state of the SBI addressable BR receive vector registers. Each bit is set when the interrupt vector is

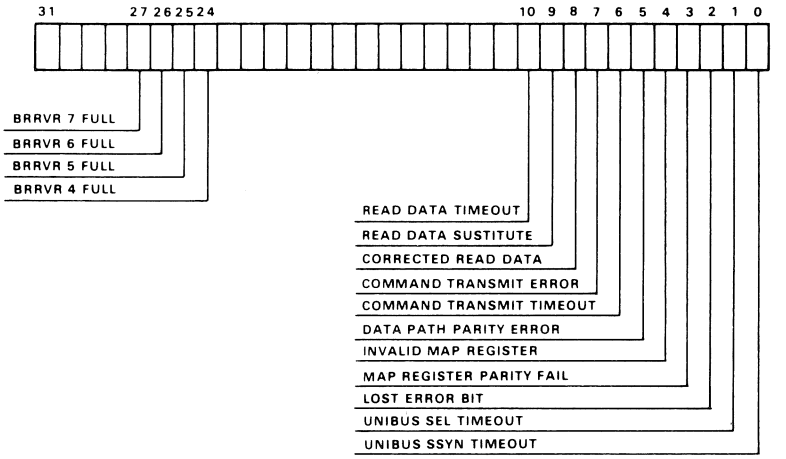


Figure 17-12 Status Register Bit Configuration

loaded into the corresponding BRRVR during a UNIBUS interrupt transaction, providing that the SBI processor is fielding UNIBUS device interrupts.

Each bit is cleared by the successful completion of a read data transmission following a read BRRVR command. The software will see these bits set only after a read data failure has occurred during the execution of the read BRRVR command, and the UNIBUS interrupt vector has been saved by the UNIBUS adapter.

Bit 27=BRRVR 7 Full

Bit 26=BRRVR 6 Full

Bit 25=BRRVR 5 Full

Bit 24=BRRVR 4 Full

Bit: 23:11 **Name:** Reserved and zero
Function: The remaining bits identify specific data transfer errors. They are read and write-one-to clear bits.

Bit: 10 **Name:** Read Data Time Out (RDTO)
Function: The UNIBUS adapter sets the Read Data Time Out bit when the following conditions are met: When a UNIBUS device has initiated a DMA read transfer, when the UNIBUS adapter has successfully transmitted a read command on the SBI, and the SBI memory has not returned the requested data within 100 μs, and when the UNIBUS device has not timed out. Note that the normal UNIBUS timeout is 10-

20 μs, and that after 10-20 μs, the UNIBUS device will set its nonexistent memory bit. Thus, the Read Data Time Out bit will be set on the UNIBUS adapter status register only if the UNIBUS device timeout function is inoperative, or takes more than 100 μs.

Bit: 9 **Name:** Read Data Substitute (RDS)
Function: This bit is set if a read data substitute is received in response to a UNIBUS to SBI read command (DMA read transfer). No data will be sent to the UNIBUS device, and when the device timeout occurs, the nonexistent memory bit will be set within the UNIBUS device.

Bit: 8 **Name:** Corrected Read Data (CRD)
Function: The UNIBUS adapter sets this bit when it receives corrected read data in response to an SBI read command during a DMA read transfer. The setting of this bit has no effect on the completion of the UNIBUS transfer.

Bit: 7 **Name:** Command Transmit Error (CXTER)
Function: The UNIBUS adapter sets this bit when it receives an error confirmation in response to an SBI command transmission during a DMA transfer.

Bit: 6 **Name:** Command Transmit Timeout (CXTO)
Function: This bit is set when a command transmission times out during a UNIBUS to SBI data transfer or during a BDP to SBI write or purge.

Note that the normal UNIBUS timeout is 10 μs, which will result in the UNIBUS device setting its nonexistent memory bit and will also abort the UBA to SBI transfer. The CXTO bit will therefore only be set for a UNIBUS to SBI transfer if the device's timeout mechanism is inoperative. The UBA will, however, attempt to perform a BDP to SBI write or purge operation for the full 100 μs timeout period if busy or no response confirmation is received, thereby setting the CXTO bit. The bit is not set for a prefetch operation since the prefetch works by anticipated addresses (i.e., the next address) and any errors resulting from the prefetch are considered to be invalid.

Bit: 5 **Name:** Data Path Parity Error (DPPE)
Function: This bit is set when a parity error occurs in the Buffered Data Path during either a UNIBUS to BDP DATI, a BDP to SBI write, or a purge.

Note that during a purge operation the address to be mapped is also obtained from the BDP and it is possible for a parity error to occur when fetching the address from the BDP. This parity error will also set the DPPE bit and abort the SBI transfer that would have taken place.

Also note that any condition that sets the DPPE bit will also set the buffer transfer error bit in the DPR of the Buffer Data Path in which the error occurred, thereby aborting any SBI transfers in progress and any future UNIBUS transfers through that BDP until the buffer transfer error is cleared.

Bit: 4 **Name:** Invalid Map Register (IVMR)
Function: The UNIBUS adapter sets this bit during a DMA transfer or purge operation when the UNIBUS address points to a map register which has not been validated by the software, or when the DMA transfer crosses an SBI page boundary for which the map register has not been validated.

Bit: 3 **Name:** Map Register Parity Failure (MRPF)
Function: This bit is set with the occurrence of a map register parity failure when a UNIBUS address is being mapped to an SBI address on a DMA transfer operation or a purge operation.

Seven of the bits just listed (RDTO, RDS, CXTER, CXTO, DPPE, IVMR, and MRPF) form an error-locking field. If any one of these bits is set, the field is locked until the bit indicating the error is cleared. The Failed Map Entry Register (FMER) is also locked and unlocked with this field. And the setting of any of these bits will cause the UNIBUS adapter to initiate an interrupt request if the interrupt enable bit for the UNIBUS to SBI data transfer error field (USEFIE) in the control register is set.

Bit: 2 **Name:** Lost Error Bit (LEB)
Function: The UNIBUS adapter sets this bit if the locking error field is locked and another error within this field occurs. The lost error bit does not initiate an interrupt request.

Bit: 1 **Name:** UNIBUS Select Time Out (UBSTO)
Function: The UNIBUS adapter sets this bit if it cannot gain access to the UNIBUS within 50 μs in the execution of a software initiated transfer (SBI to UNIBUS transfer). When UBSTO is set it indicates that the UNIBUS adapter has issued NPR on the UNIBUS but has not become bus master. This condition indicates the presence of a hardware problem on the UNIBUS. The UNIBUS may be inoperative, or one device may be holding it for extended periods. Note that if the UNIBUS does become inoperative, it may be possible to clear the problem with the assertion of UNJAM on the SBI, by setting and clearing of the UNIBUS power fail bit (control register bit 1) or by setting Adapter INIT (control register bit 0).

Bit: 0 **Name:** UNIBUS Slave Sync Time Out (UBSSYNTO)
Function: This bit is set when an SBI to UNIBUS transfer (software-initiated transfer) times out during the data transfer cycle on the UNIBUS. The timeout occurs after 12.8 μs. UBSSYNTO indicates a

transfer failure resulting when a nonexistent memory or device on the UNIBUS is addressed.

The two bits just discussed, UBSTO and UBSSYNTO, form an SBI to UNIBUS transfer error-locking field. They are set by the occurrence of the conditions mentioned and cleared by writing a (1) to the bit location. The setting of either bit will cause the UNIBUS adapter to make an interrupt request on the SBI if the SBI to UNIBUS error interrupt enable bit (SUEFIE) in the control register is set. The setting of either UBSTO or UBSSYNTO will lock the failed UNIBUS address register (FUBAR), thus storing the high 16 bits of the UNIBUS address identified with the failure. The FUBAR will remain locked until the UBSTO and UBSSYNTO bits are cleared.

Diagnostic Control Register (DCR)

The Diagnostic Control Register provides control and status bits which aid in the testing and diagnosis of the UNIBUS adapter. The bits of this register, when set, will defeat certain vital functions of the UNIBUS adapter. The DCR is therefore not intended for use during normal system operation. Figure 17-13 shows the bit configuration of the DCR.

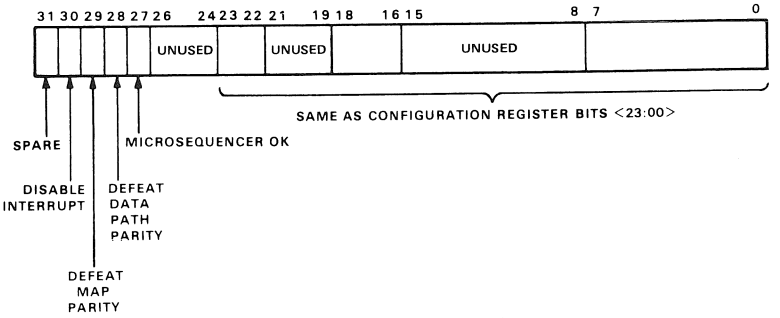


Figure 17-13 Diagnostic Control Register Bit Configuration

Bit: 31 **Name:** Spare
Function: This read/write bit has no effect on any UNIBUS adapter operation. It can be set by writing a one and cleared by writing a zero to the bit location. SBI Dead, Adapter INIT, and a power up sequence on the UNIBUS adapter will clear this bit.

Bit: 30 **Name:** Disable Interrupt (DINTR)
Function: When it is set, this bit will prevent the UNIBUS adapter

from recognizing interrupts on the UNIBUS. It is useful in testing the response of the UNIBUS adapter to the passive release condition during a UNIBUS interrupt transaction. This bit is set by writing a one and cleared by writing a zero to the bit location. SBI Dead, Adapter INIT, and the power up sequence on the UNIBUS adapter will also clear DINTR.

Bit: 29 **Name:** Defeat Map Parity (DMP)
Function: When it is set, this read/write bit will inhibit the parity bits of the map registers from entering the map register parity checkers. The map register parity generator checkers generate and check parity on eight bit quantities. Each parity field (eight data bits and one parity bit) is implemented so that the total number of ones in the field is odd.

For example, if bits <7:0> of a map register equal zero, then the parity bit equals one. However, if the DMP bit is set, then the parity bit is disabled and the parity checkers will see all zeros. This results in a map register parity failure. Then, if the DMP bit is set, the parity checkers will see correct parity. Note, however, that if bits <7:0> of the map register contain an odd number of ones, the generated parity bit will be zero. The state of the DMP bit will therefore have no effect on the parity result in this case.

When the integrity of the parity generator checkers is to be tested, the map register must contain data so that at least one of the bytes contains an even number of ones. The DMP bit, when set, will disable the parity bit, and the map register parity failure can be detected during a DMA transfer. SBI Dead, Adapter INIT, and the power up sequence on the UNIBUS adapter will clear this bit.

Bit: 28 **Name:** Defeat Data Path Parity (DDPP)
Function: The DDPP bit functions in the same way as the DMP bit. When it is set, the DDPP bit will inhibit the parity bits of the data path RAM from entering the parity checkers. The data path parity generator checkers generate and check parity on eight bit data units. Each parity field (eight data bits and one parity bit) is implemented so that the total number of ones in the field is odd. When the integrity of the parity generator checkers is to be tested through use of the DDPP bit, the total number of ones in at least one of the bytes of data must be even. With the parity bit disabled by the DDPP bit, a data path parity failure will result during a DMA transfer via that buffered data path. SBI Dead, Adapter INIT, and the power up sequence on the UNIBUS adapter will clear the DDPP bit.

Bit: 27 **Name:** Microsequencer OK (MIC OK)
Function: The MIC OK bit is a read-only bit which indicates that the UNIBUS adapter microsequencer is in the idle state. The mi-

crosequencer will enter the idle state after it has completed the initialization sequence or once it has completed a UNIBUS adapter function.

The MIC OK bit can be used by diagnostics to determine whether or not the microsequencer has completed a successful power up sequence and whether or not it is caught up in any loops. Note that SBI dead, UNIBUS adapter power supply DC LO, and Adapter INIT force the microsequencer into the initialization routine. Once the routine has been completed and the microsequencer has entered the idle state, MIC OK will be true (1).

Bit: 26:24 **Name:** Reserved and zero
Function:

Bit: 23:0 **Name:**
Function: Same as bits <23:0> of the Configuration Register

Failed Map Entry Register (FMER)
The Failed Map Entry Register contains the map register number used for either a DMA transfer or a purge operation which has resulted in the setting of one of the following error bits of the status register: IVMR, MRPF, DPPE, CXTO, CXTER, RDS, RDTO. This register is locked and unlocked with the UNIBUS to SBI data transfer error field of the status register. The contents of the FMER are valid only when the register is loaded. The FMER is a read-only register. Attempts to write to the FMER will result in an SBI error confirmation. No signals or events will clear the register.

The software can read the FMER to obtain the map register number associated with the failure. It can then read the contents of the failing map register to determine the number of the data path which failed.

Figure 17-14 shows the bit configuration for the Failed Map Entry Register.

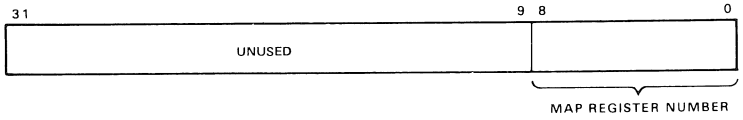


Figure 17-14 Failed Map Entry Register Bit Configuration

Bit: 31:9 **Name:** Reserved and zero
Function:

Bit: 8:0 **Name:** Map Register Number (MRN)
Function: These bits contain the number of the map register which

was in use at the time of a failure. Bits <8:0> correspond to bits <17:9> of the UNIBUS address.

Failed UNIBUS Address Registers (FUBAR)
The FUBAR contains the upper 16 bits of the UNIBUS address translated from an SBI address during a previous software-initiated data transfer. The occurrence of either of two errors indicated in the status register will lock the FUBAR: UNIBUS Select Time Out (UBSTO) and UNIBUS Slave Sync Time Out (UBSSYNTO). When the error bit is cleared, the register will be unlocked.

The FUBAR is a read-only register. Attempting to write to the register will result in an error confirmation. No signals or conditions will clear the register. Figure 17-15 shows the bit configuration of the FUBAR. The contents of the FUBAR are listed below.

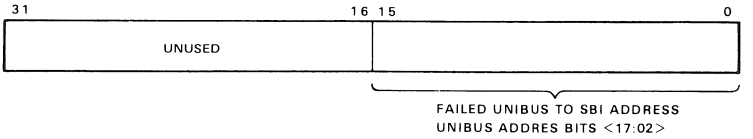


Figure 17-15 Failed UNIBUS Address Register Bit Configuration

Bit: 31:16 **Name:** Reserved and zero
Function:

Bit: 15:0 **Name:** Failed UNIBUS to SBI Address
Function: These bits correspond to UNIBUS Address bits <17:2>.

Buffer Selection Verification Registers 0-3 (BRSVR)
These four read/write do-nothing registers are provided in order to give the diagnostic software a means of accessing and testing the integrity of the data path RAM. Four spare locations in the data path RAM have been assigned to these registers. Writing and reading the BRSVRs has no effect on the behavior of the UNIBUS adapter. The BRSVR bit configuration is shown in Figure 17-16.

The contents of the BRSVRs are listed below.
Bit: 31:16 **Name:**
Function: Always zero

Bit: 15:0 **Name:**
Function: Read/write bits

BR Receive Vector Registers 4-7 (BRRVR)
The UNIBUS adapter contains four BR receive vector registers:

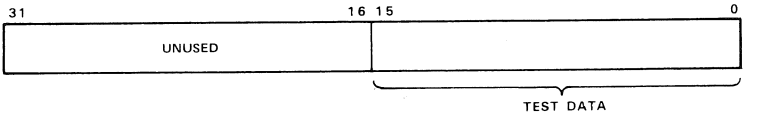


Figure 17-16 Buffer Selection Verification Register Bit Configuration

BRRVR 7, BRRVR 6, BRRVR 5, and BRRVR 4. Each BRRVR corresponds to a UNIBUS interrupt bus request level: 7, 6, 5, 4. Each BRRVR is a read-only register and will contain the interrupt vector of a UNIBUS device interrupting at the corresponding BR level. Each BRRVR is read by the software as a part of the UNIBUS adapter interrupt service routine. Note that the UNIBUS adapter interrupt service routine is the routine to which the VAX-11 CPU will transfer control once it has determined that the UNIBUS adapter has transmitted an interrupt request on the SBI.

If the IFS and BRIE bits on the control register are set, so that UNIBUS interrupt requests are passed to the SBI, then the VAX-11 CPU responds with an Interrupt Summary Read command. The UBA sends its request sublevel as an Interrupt Summary Response. The software then invokes the UBA interrupt service routine, initiating a read transfer to the appropriate BRRVR. The UNIBUS adapter will assert the contents of the BRRVR on the SBI as read data if the corresponding BRRVR Full bit in the status register is set. If the BRRVR Full bit is not set, the Read BRRVR command causes the UNIBUS adapter to fetch the interrupt vector from the interrupting UNIBUS device. The interrupt vector is loaded into the BRRVR only at the successful completion of a UNIBUS interrupt transaction. The UNIBUS adapter will then send the contents of the BRRVR to the SBI as read data. The BRRVR used is cleared only when the UNIBUS adapter receives an ACK confirmation for the read data. Following this exchange, the UNIBUS adapter interrupt service routine will use the contents of the BRRVR to branch to the appropriate UNIBUS device service routine.

Four types of failure conditions can occur when the software is reading a BRRVR and the VAX-11 CPU is servicing a UNIBUS device interrupt:

1. If the software attempts to read a BRRVR for which a BR interrupt line is not asserted, and BRRVR is not full, the zero vector (all zeroes data) will be sent as read data.
2. If the BR line asserted by the interrupting UNIBUS device is released during the interrupt summary read transfer, and the vector is not received from the device (passive release), then the zero vector will be sent as read data.

3. If the vector has been received from the interrupting device, but an ACK confirmation is not received following the read data transmission, then the BRRVR will not be cleared, and the BRRVR Full bit will remain set. Subsequent read commands to the full BRRVR will cause the UNIBUS adapter to send the stored vector, but the BRRVR will remain full until the UNIBUS adapter receives an ACK confirmation for the read data. Note that the BRRVR Full bits always reflect the state of the BRRVRs.
4. If the IFS bit in the control register is cleared and the software reads a BRRVR, then the zero vector will be sent as read data.

The contents of the BRRVR are also used by the software to determine whether or not the UNIBUS adapter itself has an interrupt pending. Bit 31 of the BRRVR is the Adapter Interrupt Request Indicator. Although the bit is present in all four BRRVRs, it will be active only in the BRRVR corresponding to the interrupt request level that has been assigned to the UNIBUS adapter. If bit 31 is set when the software reads the BRRVR, then an adapter interrupt request is pending.

Figure 17-17 shows the BR Receive Vector Register bit configuration.

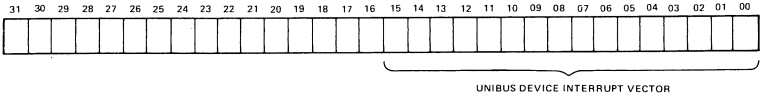


Figure 17-17 BR Receive Vector Register Bit Configuration

The contents of the four BRRVRs are as follows:

Bit: 31 **Name:** Adapter Interrupt Request Indicator
Function: 0=No UBA interrupt pending.
1=UBA interrupt pending.

Bit: 30:16 **Name:** Reserved and zero
Function:

Bit: 15:0 **Name:** Device Interrupt Vector Field
Function: These bits contain the device interrupt vector loaded by the UNIBUS adapter during a UNIBUS interrupt transaction.

Data Path Registers 0-15 (DPR)
The UNIBUS adapter contains 16 data path registers (DPR 0 to DPR 15), each of which corresponds to one of the 16 data paths. DPR 0, corresponding to the direct data path, is always 0.

Figure 17-18 shows the Data Path Register bit configuration.

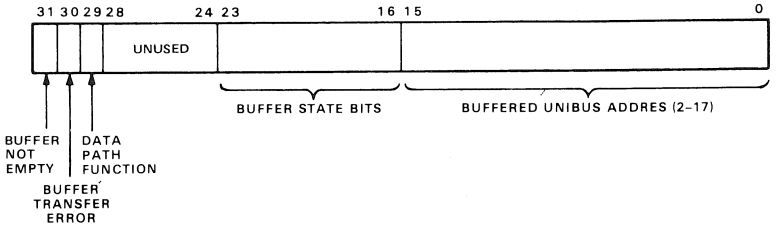


Figure 17-18 Data Path Register Bit Configuration

The DPR bit functions are as follows:

- Buffer Not Empty and the Purge Operation

Bit: 31 **Name:** Buffer Not Empty (BNE)
Function: Each DPR contains a data path status bit called Buffer Not Empty. This bit is read/write one to clear bit.

0 = Buffer empty
1 = Buffer not empty

If this bit is set (1), the BDP contains valid data. If clear, then the BDP does not contain valid data. The UNIBUS adapter uses the bit to determine the proper action for DMA transfers via the BDP. If bit <31> is set as a DATI transfer begins, the data in the BDP will be asserted on the UNIBUS. If bit <31> is clear on a DATI, the UNIBUS adapter will initiate a read transfer to SBI memory, load the read data into the BDP, thereby setting bit <31>, and gate the addressed data to the UNIBUS.

For a DMA write transfer via the associated BDP, the BNE bit is set each time UNIBUS data is loaded into the BDP. The bit is then cleared when the contents of the BDP are transferred to SBI memory.

The software will write a one to this bit to initiate the purge operation. The purge operation is required at the completion of a UNIBUS device block transfer and is performed in the following way:

1. Write transfers to memory. If any bytes of data remain in the corresponding BDP (BNE is set), the UNIBUS adapter will transfer this data to memory. The UNIBUS adapter will then initialize the BDP and clear the BNE bit. If no data remains to be transferred (BNE is cleared), the purge operation will be treated as a no-op (it is a legal do-nothing function).
2. Read transfers to memory. If any bytes of data remain in the BDP, the UNIBUS adapter will initialize the BDP by clearing the BNE bit. If no data remains, the purge will be treated as a no-op.

In addition, the following considerations apply to the purge operation:

- For purge operations in which data are transferred to memory, the SBI transfer takes about 2 μs. The UBA will not respond to Data Path Register read transfers during this period (busy confirmation), thereby preventing a race condition when testing for BNE bit.
- A purge operation to Data Path Register Zero (Direct Data Path) is treated by the UBA as a NO-OP.

Bit: 30 **Name:** Buffer Transfer Error (BTE)
Function: This is a read-write one to clear bit. The UNIBUS adapter sets the BTE bit if a failure occurs during a DMA write transfer, or a purge, or for a data path parity failure during a DMA read transfer via the associated BDP. If bit <30> is set, any additional DMA transfers via the BDP will be aborted until the bit is cleared by the software. Note that if a parity error on the UNIBUS occurs during a DMA read, the UNIBUS signal PB will be asserted, giving the UNIBUS device the opportunity to abort its own DMA transfer. The purge operation does not clear the BTE bit.

Bit: 29:0 **Name:**
Function: Read-only bits.

Bit: 29 **Name:** Data Path Function (DPF)
Function: This bit indicates the function of the DMA transfer using this data path.

0 = DMA Read
1 = DMA Write

Bit: 28:24 **Name:**
Function: Unused

Bit: 23:16 **Name:** Buffer State (BS)
Function: These eight bits indicate the state of each of the eight byte buffers of the associated BDP during a DMA write transfer. They are included in the Data Path Register for diagnostic purposes only. The UNIBUS adapter generates the SBI mask bits from the BS bits during a DMA write transfer or purge operation. The bits are set as each byte is written from the UNIBUS. The bits are cleared during the SBI write operation.

0 = Empty
1 = Full

Bit: 15:0 **Name:** Buffered UNIBUS Address (BUBA)
Function: This portion of each DPR contains the upper 16 bits of the UNIBUS address, UA <17:2>, asserted during the DMA transfer using the associated BDP. If the transfer through the associated BDP is in the byte offset mode, and the last UNIBUS transfer has spilled over into the next quadword, then these bits contain UA <17:2>. This is the

UNIBUS address from which the SBI address will be mapped should a purge operation occur before the next UNIBUS transfer.

Map Registers 0-495(10)
The UNIBUS adapter contains 496 map registers, one for each UNIBUS memory page address (a page = 512 bytes).

REG	Offset
MR0	800
MR1	804
MR2	808
MR3	80C
.	.
.	.
.	.
MR494	FB8
MR495	FBC

When a DMA transfer begins, the upper nine address bits asserted by the UNIBUS device select one of the map registers which the software has set up. The map register in turn tests for the validity of the current UNIBUS transfer, steers the transfer through one of the 16 data paths, determines whether or not the transfer will take place in the byte offset mode if a BDP has been selected, and maps the UNIBUS page address to an SBI page address.

The map registers are numbered sequentially from 0 through 495_{10}. There is a one to one correspondence between each map register and UNIBUS memory page address (i.e., MR0 corresponds to UNIBUS memory page 0, MR1 to UNIBUS Memory Page 1.....MR495 to UNIBUS memory page 495). Each map register contains the information required to effect the data transfer of the UNIBUS device addressing that page:

1. The fact that the software has loaded the map register (map register valid).
2. The number of the data path to be used by the transfer and, if a BDP is used, whether it is in byte offset mode.
3. The SBI page to which the transfer will be mapped.

Since the map register is implemented with a bipolar RAM, the contents of the map registers will be checked by parity. If, during a

UNIBUS transfer, the parity test fails, the map register parity fail bit of the UNIBUS adapter status register will be set and the UNIBUS transfer will be aborted.

Figure 17-19 illustrates the map register bit configuration.

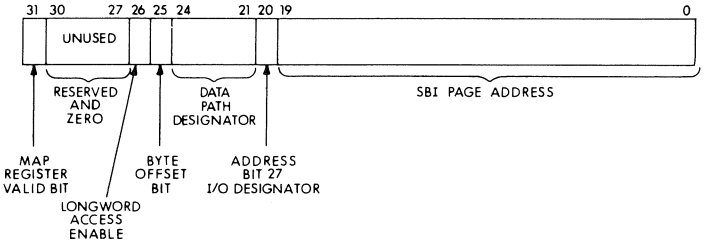


Figure 17-19 Map Register Bit Configuration

NOTE

For brevity, for the map register description, "this UNIBUS page" refers to "the UNIBUS memory page corresponding to this map register."

The contents of a map register are as follows:

Bit: 31 **Name:** Map Register Valid (MRV)
Function: 0 = not valid—initialized state
1 = valid

The MRV is set by the software to indicate that the contents of the map register are valid. The MRV is tested each time that "this UNIBUS page" is accessed. If the bit is set (1), the transfer continues. If the bit is not set, the UNIBUS transfer is aborted (nonexistent memory error in the UNIBUS device) and the invalid map register bit is set in the UNIBUS adapter status register.

The MRV can be set and cleared by the software.

Bit: 30:27 **Name:**
Function: Reserved read/write bits

Bit: 26 **Name:** Longword Access Enable (LWAE)
Function: This is a read/write bit. If set, and the map register selects a BDP, then the longword-aligned 32-bit random access mode is enabled for the BDP. The longword-aligned 32-bit random access mode has been discussed above. This bit has no effect if the Direct Data Path is selected by the map register. This bit is cleared on initialization.

Bit: 25 **Name:** Byte Offset Bit (BO)

Function: This is a read/write bit. If set, and "this UNIBUS page" is using one of the BDPs, and the transfer is to an SBI memory address, then the UNIBUS adapter will perform a byte offset operation on the current UNIBUS data transfer. The software can interpret this operation as increasing the physical SBI memory address, mapped from the UNIBUS address, by one byte. This allows word-aligned UNIBUS devices to transfer to odd byte memory addresses.

UNIBUS transfers via the DDP or to SBI I/O addresses will ignore the Byte Offset bit.

This bit is cleared on initialization.

Bit: 24:21 **Name:** Data Path Designator Bits (DPDB)
Function:

0000 =	Direct Data Path (DDP)
0001 =	Buffered Data Path 1
.	.
.	.
1111 =	Buffered Data Path 15

The DPDBs are read/write bits that are set and cleared by the software to designate the data path that "this UNIBUS page" will be using.

The software can assign more than one UNIBUS transfer to the DDP.

The software must assure that no more than one active UNIBUS transfer is assigned to any BDP.

The DPDBs are cleared on initialization.

Bit: 20:0 **Name:** SBI Page Address (SPA <27:7>, also known as Page Frame Number, PFN)
Function: The SPA bits contain the SBI page address to which "this UNIBUS page" will be mapped. These bits perform the UNIBUS to SBI page address translation. When an SBI transfer is initiated the contents of SPA<27:7> are concatenated with UNIBUS address bits UA<8:2> to form the 28-bit SBI address.

POWER FAIL AND INITIALIZATION

The UNIBUS adapter controls the UNIBUS power fail, power up, and initialization sequences of the UNIBUS. This section explains the behavior of the UNIBUS subsystem for each of the following:

1. System Power Up
2. System Power Down
3. UNIBUS Power Down

4. Programmed Power Down
5. SBI UNJAM

System Power Up
The UNIBUS remains in a powered down state as long as the UNIBUS adapter is in a powered down state. During System Power Up, the UNIBUS adapter will initiate the UNIBUS power up sequence, provided the UNIBUS has power. Once the power up sequence has been completed, the UNIBUS Initialization Complete bit of the UNIBUS adapter status register is set and an interrupt request is initiated to the CPU. If the UNIBUS power was not on at the time that the system powered up, the power up sequence will not continue until the UNIBUS power has been turned on. The power up sequence will completely initialize all registers and functions of the UNIBUS adapter. The deassertion of power supply AC LO will set the adapter power up bit in the Configuration Register and initiate an interrupt request.

SBI Power Fail
The UNIBUS adapter will initiate a UNIBUS power fail sequence whenever an SBI power failure is detected (SBI Dead asserted). The UNIBUS will remain powered down as long as SBI Dead is asserted. The UBA will initiate the UNIBUS power up sequence when SBI Dead is released.

UNIBUS Power Fail
A power loss on the UNIBUS will initiate a UNIBUS power fail sequence. The UNIBUS power down bit of the status register will be set and the UNIBUS adapter will initiate an interrupt request (providing the CNFIE bit is set). The UNIBUS will remain in a powered down state until UNIBUS power has been restored, at which time a UNIBUS power up sequence is initiated. The UNIBUS initialization complete bit of the status register will be set on a successful power up sequence and the UBPDN bit will be cleared. The UNIBUS power fail lines will not affect the state of the SBI power fail lines.

Programmed UNIBUS Power Fail
The software can induce a power fail sequence on the UNIBUS by first setting and then clearing the UNIBUS power fail bit of the control register. The UNIBUS adapter will initiate a power fail sequence when the UPF is set. Once it has been initiated, the power fail sequence will continue to completion independent of the state of the UPF. On completion of the power down sequence, the UNIBUS adapter will initiate a power up sequence if or when the UPF is cleared, provided power is normal for both the UNIBUS and UNIBUS adapter.

Setting the AD INIT bit will also initiate a power fail and initialization sequence on the UNIBUS as well as completely initialize all registers and functions of the UBA.

SBI UNJAM

The assertion of SBI UNJAM will initiate the UNIBUS power fail and initialization sequence. It will also clear all interrupt enable bits of the UBA control register. It will initialize the UBA SBI logic so that the UBA is available for an SBI Command.

EXAMPLE

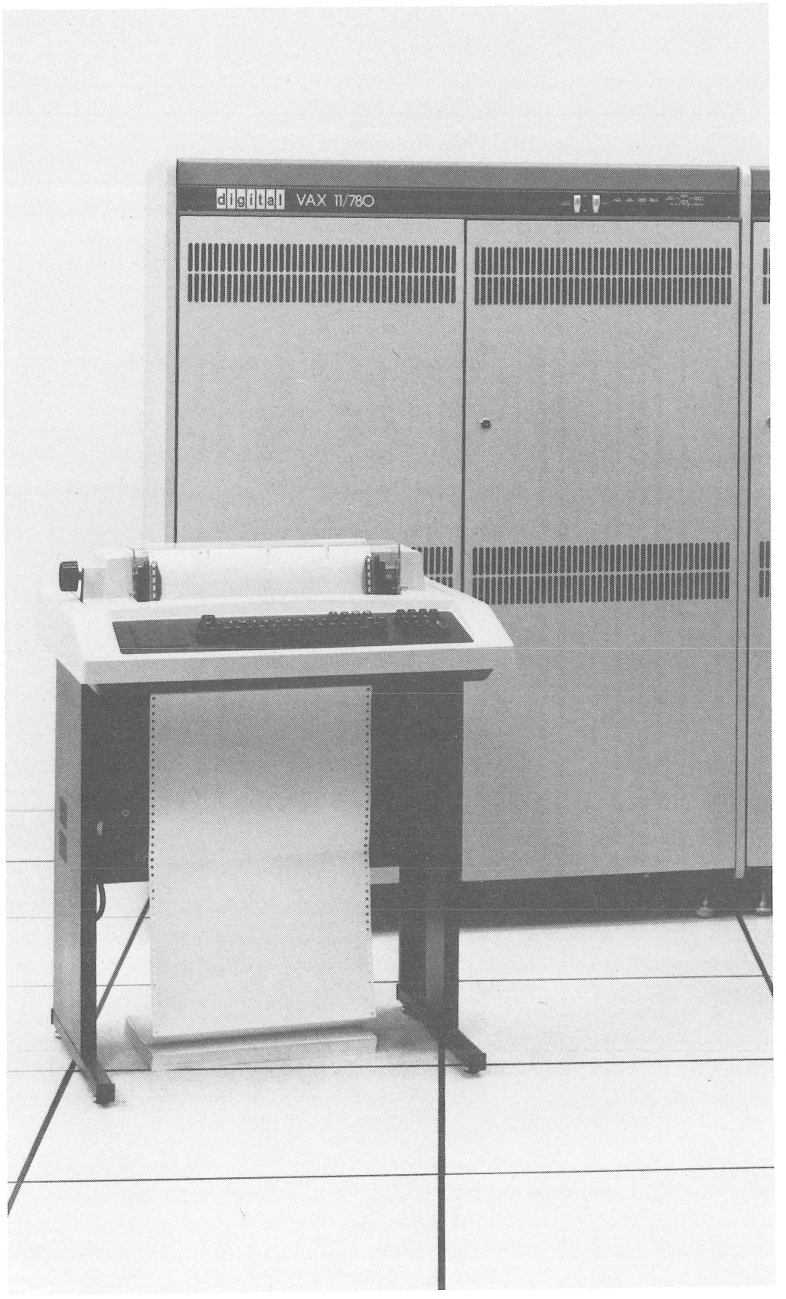
Presented is a program to read data from the RK06 disk subsystem into memory. The program is fully documented and is designed to demonstrate the loading of the UBA map registers, the use of a buffered data path (including the purge), access to UNIBUS device registers, and initialization of the UNIBUS. In order to run the program, it must be loaded from the floppy disk by the console into memory. Initially, the program can be assembled and linked under VAX/VMS and then transferred to the floppy disk using the RSX-11M utility program FLX. The file structure of the floppy is RT-11 format.

```
;       PROGRAM TO READ FROM THE RK06 INTO MEMORY
;               THIS PROGRAM WILL TRANSFER 3210 BYTES FROM THE RK06
;               DISK TO MEMORY STARTING AT ADDRESS 4567.  THE TRANSFER
;               WILL USE A BLOCK OF MAP REGISTERS STARTING AT
;               MAP REGISTER 25 AND WILL USE A BUFFERED DATA PATH (DP5).
;               IT WILL READ DATA FROM DRIVE 0, TRACK 2, CYLINDER 4, SECTOR 6.
;               IT WILL RING THE BELL OF THE CONSOLE IF NO ERRORS ARE
;               DETECTED.

;               THE PROGRAM IS DESIGNED TO DEMONSTRATE THE LOADING OF THE
;               UBA MAP REGISTERS, USE OF A BUFFERED DATA PATH (INCLUDING
;               THE PURGE), ACCESS TO UNIBUS DEVICE REGISTERS, AND
;               INITIALIZATION OF THE UNIBUS.  NO CLAIM IS MADE FOR
;               ELEGANCE OF PROGRAMMING STYLE.




;       SYMBOL DEFINITIONS

;   UBA RELATED SYMBOLS:
        UBA_BASE = ^X20006000           ; UBA AT TR = 3
        UBA_CNFR = ^X0                  ; OFFSET TO UBA CONFIGURATION REGISTER
          UBA_UBIC = ^X10000            ;   UNIBUS INITIALIZATION COMPLETE
        UBA_CR   = ^X4                  ; OFFSET TO UBA CONTROL REGISTER
          UBA_ADINIT = ^X1              ;   ADAPTOR INIT AND UNIBUS INITIALIZATION
        UBA_SR   = ^X8                  ; OFFSET TO UBA STATUS REGISTER
        UBA_DPO  = ^X40                 ; OFFSET TO UBA DATA PATH REGISTER 0
         UBA_DP_BNE = ^X80000000        ;   BUFFER NOT EMPTY BIT
                                        ;    USED TO PURGE BUFFER DATA
                                        ;    PATH AT END OF XFER.
         UBA_DP_BTE = ^X40000000        ;   BUFFER TRANSFER ERROR BIT
        UBA_MRO  = ^X800                ; OFFSET TO UBA MAP REGISTER 0
         MAP_VALID = ^X80000000         ;   VALID BIT IN MAP REGISTER
         BYTE_OFST = ^X2000000          ;   BYTE OFFSET BIT IN MAP REGISTER


;  RK611 RELATED SYMBOLS:
        UNIBUS_BASE = ^X20100000        ; BASE ADDRESS FOR UNIBUS ADDRESS 0
        DK_BASE_ADD = ^O777440          ; UNIBUS BASE ADDRESS OF DK611
        DK_CS1 = ^O0                    ; RK611 CONTROL STATUS REGISTER 1
        DK_CS2 = ^O10                   ; RK611 CONTROL STATUS REGISTER 2
        DK_DS  = ^O12                   ; RK611 DRIVE STATUS REGISTER
        DK_DC  = ^O20                   ; RK611 DESIRED CYLINDER REGISTER
        DK_DA  = ^O6                    ; RK611 DISK ADDRESS REGISTER
        DK_WC  = ^O2                    ; RK611 WORD COUNT REGISTER
        DK_BA  = ^O4                    ; RK611 BUS ADDRESS REGISTER
        SCLR   = ^O40                   ; RK611 SUBSYSTEM CLEAR
        PACACK = ^O3                    ; RK611 PACK ACKNOWLEDGE AND GO BIT SET
        READ   = ^O21                   ; RK611 DISK READ AND GO BIT SET.
        ;  MISC SYMBOLS:
        BELL   = ^O7                    ; ASCII BELL
        TXDB   = 35                     ; CONSOLE TRAANSMIT DATA BUFFER




;  THIS SECTION LOADS THE UBA_BASE ADDRESS INTO R0 AND THEN INITIALIZES
;  THE UNIBUS.

BEGIN:  MOVL    #UBA_BASE, R0           ; LOAD UBA'S ADDRESS INTO R0
INIT:   MOVL    #UBA_ADINIT,UBA_CR(R0)  ; INIT UBA AND UNIBUS

;  THE UBA AND UNIBUS ARE BEING INITIALIZED.  THE PROGRAM CANNOT MAKE
;  ANY ACCESSES TO THE UBA OR THE UNIBUS DURING THIS PERIOD OF TIME.
;  BUT THAT'S OK BECAUSE WE HAVE LOTS TO DO IN THE MEAN TIME...


;  THIS SECTION WILL SET UP THE MAP REGISTERS TO BE USED FOR THE TRANSFER.

;    FIND THE OFFSET OF THE INITIAL MAP REGISTER  AND PUT INTO R1.

        ASHL    #2, MAP_REG, R1         ; MULTIPLY THE MAP REGISTER BY 4
                                        ;   TO FIND ITS OFFSET AND PUT INTO R1.

        ADDL    #UBA_MRO, R1            ; ADD THE BASE ADDRESS OF THE MAP REGISTERS
                                        ;  TO THE OFFSET AND PUT IN R1.
        ADDL    #UBA_BASE, R1           ; ADD IN THE UBA BASE ADDRESS AND PUT IN R1.
```

```
;  R1 NOW CONTAINS THE ADDRESS OF THE FIRST MAP REGISTER TO BE LOADED.

;  THIS SECTION WILL DETERMINE THE CONTENTS OF THE MAP REGISTERS AND
;  STORE IT IN R2.

        ;  THIS SECTION WILL DETERMINE THE PAGE FRAME NUMBER OF THE FIRST
        ;  MEMORY PAGE TO BE ACCESSED.  THE PHYSICAL MEMORY ADDRESS IS SHIFTED
        ;  RIGHT NINE BITS TO BECOME THE PAGE FRAME NUMBER.

        ;  THE DATA PATH NUMBER TO BE USED FOR THE TRANSFER WILL THEN BE INSERTED
        ;  INTO THE DATA PATH DESIGNATOR FIELD AND THE VALID BIT IS SET.

        ASHL    #-9, MEMSAD, R2         ; TURN START ADDRESS INTO THE
                                        ;  PAGE FRAME NUMBER.
        INSV    DP_NUM, 21, 4, R2       ; INSERT BITS 0-3 OF DP_NUM INTO
                                        ;  BITS 21-24 OF R2.
        BISL    #MAP_VALID, R2          ; SET MAP VALID BIT.


        ;  DETERMINE IF THE BYTE ALIGNMENT BIT OF THE BUFFERED DATA PATHS IS
        ;  REQUIRED.  THE RK611 ONLY KNOWS ABOUT WORD ALIGNED TRANSFERS.  IF
        ;  THE START MEM ADDRESS IS ODD THEN THE BYTE OFFSET BIT OF THE
        ;  MAP REGISTERS MUST BE SET.

        BITL    ^X1, MEMSAD             ; IS MEM ADDRESS ODD?
        BEQL    CONT$                   ; IF NOT THEN CONTINUE.
        BISL    #BYTE_OFST, R2          ;   YES -- SET BYTE OFFSET BIT

CONT$:  NOP                             ;CONTINUE


;  THIS SECTION COMPUTES THE CONTENTS OF THE RK611 BUS ADDRESS REGISTER
;  AND THE EXTENDED ADDRESS BITS OF CONTROL STATUS REGISTER 1.

;  THE RESULT OF THIS SECTION WILL BE THAT WHEN THE RK611 ASSERTS THE
;  ADDRESS ONTO THE UNIBUS, UNIBUS ADDRESS BITS <17:09> WILL POINT TO
;  THE MAP REGISTER THAT CONTAINS THE PAGE FRAME NUMBER FOR THE TRANSFER
;  AND UNIBUS ADDRESS BITS <8:0> WILL CONTAIN THE BYTE OFFSET WITHIN THE
;  PAGE.
;  THE CONTENTS OF THE REGISTERS WILL BE AS FOLLOWS:
;       DK_CS1  BA <17 : 16> AND
;       DK_BA   BA <15 : 09> WILL CONTAIN THE POINTER
;               TO THE MAP REGISTER THAT CONTAINS THE PAGE FRAME NUMBER
;               FOR THE TRANSFER.
;       DK_BA   BA <08 : 00> WILL CONTAIN THE BYTE OFFSET WITHIN THE PAGE.
;  R3 WILL BE USED TO SET UP TO CONTAIN THE INITIAL UNIBUS ADDRESS OF
;  THE TRANSFER.
;
        ASHL    #9, MAP_REG, R3         ; SHIFT MAP REGISTER NUMBER TO FORM
                                        ;   MAP REGISTER POINTER.
        BICL3   #^XFFFFFE00, MEMSAD, R4 ; CLEAR ALL BUT BYTE OFFSET WITHIN
                                        ;   THE PAGE
        BISL    R4, R3                  ; AND COMBINE WITH MAP REGISTER POINTER
                                        ;   IN R3.

;  THIS SECTION WILL DETERMINE THE WORD COUNT FOR THE TRANSFER.
;  CONVERT BYTE COUNT TO WORD COUNT FOR THE RK611.  IF BYTE COUNT IS ODD
;  THEN THE WORD COUNT MUST BE INCREMENTED TO CONTAIN ALL BYTES OF THE
;  TRANSFER.

        INCL    BCOUNT                  ; INCREMENT BYTE COUNT TO ACCOUNT
                                        ;  FOR ODD BYTE COUNT.
        ASHL    #-1, BCOUNT, R4         ; CONVERT TO WORD COUNT AND LOAD
                                        ;  INTO R4.


;  THIS SECTION WILL SET UP DISK ADDRESS  TRACK AND SECTOR - WILL USE R5.

        MOVL    SECTOR, R5              ; GET SECTOR NUMBER.
        INSV    TRACK, #8, #3, R5       ; INSERT BITS 0-2 OF TRACK INTO
                                        ;  BITS 8-10 OF R4.


;  AT THIS POINT ALL OF THE VALUES REQUIRED FOR THE TRANSFER HAVE BEEN
;  DETERMINED.  THE VALUES OF THE REGISTERS ARE:

;               R0 = UBA_BASE ADDRESS
;               R1 = ADDRESS OF FIRST MAP REGISTER TO BE USED FOR TRANSFER
;               R2 = CONTENTS FOR THE INITIAL MAP REGISTER
;               R3 = INITIAL UNIBUS ADDRESS FOR TRANSFER
;               R4 = WORD COUNT
;               R5 = SECTOR AND TRACK FOR DK_DA REGISTER
```

```
;  THE REMAINING SECTIONS INVOLVE ACCESSES TO THE UNIBUS AND THE UBA.
;  THE INITIALIZATION SEQUENCE MUST BE COMPLETE BEFORE MAKING ACCESSES
;  TO THE UBA (OTHER THAN THE CONFIGURATION REGISTER) OR THE UNIBUS.

1$:     BITL    #UBA_UBIC, UBA_CNFR(R0) ; IS UNIBUS INITIALIZATION COMPLETE?
        BEQL    1$                      ; NO - KEEP TESTING
                                        ; YES - CONTINUE


;  THIS SECTION WILL LOAD THE UBA MAP REGISTERS THAT WILL BE USED FOR THE
;  TRANSFER.  THE MAP REGISTERS USED FOR THE BLOCK TRANSFER MUST BE
;  CONTIUOUS.  THIS PROGRAM ASSUMES THAT CONTIGUOUS PHYSICAL MEMORY
;  PAGES ARE USED FOR THE TRANSFER.  THE CONTENTS OF THE INITIAL MAP
;  REGISTER WAS PREVIOUSLY DETERMINED AND STORED IN R2.  THE PHYSICAL
;  ADDRESS OF THE INITIAL MAP REGISTER WAS DETERMINED ABOVE AND STORED
;  IN R1.

        MOVL    BCOUNT, R6              ; LOAD BYTE COUNT INTO R6
2$:     MOVL    R2, (R1)+               ; LOAD MAP REGISTER WITH CONTENTS
                                        ; OF R2.
        INCL    R2                      ; INCREMENT PAGE FRAME NUMBER.
                                        ;  (ASSUMES CONTIGUOUS PAGES OF
                                        ;    PHYSICAL MEMORY).
        SUBL    #^X200, R6              ; THERE ARE 200(HEX) BYTES PER PAGE.
                                        ; ARE ALL MAPS SET UP?
        BGTR    2$                      ;   NO SET UP NEXT MAP REGISTER.
        MOVL    R2, (R1)+               ; SET UP ONE MORE SINCE TRANSFER
                                        ;   TRANSFER MAY NOT BE PAGE ALIGNED.
        CLRL    (R1)                    ; INVALIDATE NEXT MAP REGISTER TO
                                        ;   STOP THE UBA SHOULD THE TRANSFER
                                        ;   GO BEYOND ITS EXPECTED LIMIT.

;  THIS SECTION WILL PERFORM THE DISK TRANSFER SEQUENCE.
;    A SUBSYSTEM CLEAR WILL BE ISSUSED TO THE RK611, THE PACK AKNOWLEDGE
;    FUNCTION WILL BE ISSUED, AND THE DISK TRANSFER WILL BE INITIATED.
; NOTE THAT ALL UNIBUS ACCESSES MUST BE OF WORD OR BYTE FORMAT..

;  WE ARE NOW FINISHED WITH R1 AND R2.
;  FIND BASE ADDRESS OF RK611 AND LOAD INTO R1.

        ADDL3   #DK_BASE_ADD, #UNIBUS_BASE, R1  ;BASE ADDRESS OF RK611 TO R1
        MOVW    #SCLR, DK_CS2(R1)       ; ISSUE A RK611 SUBSYSTEM CLEAR
        MOVW    DRIVE, DK_CS2(R1)       ; SELECT DRIVE NUMBER
        MOVW    R5, DK_DA(R1)           ; LOAD DISK ADDRESS SECTOR AND TRACK
                                        ; STORED IN R5 FROM ABOVE.
        MOVW    #PACACK, DK_CS1(R1)     ; ISSUE PACK ACKNOWLEDGE FUNCTION
3$:     TSTB    DK_CS1(R1)              ; WAIT FOR READY
        BGEQ    3$                      ;  NOT READY KEEP WAITING

        MOVW    CYLINDER, DK_DS(R1)     ; LOAD CYLINDER ADDRESS
        MNEGW   R4, DK_WC(R1)           ; LOAD 2'S COMPLIMENT OF WORD COUNT
        MOVW    R3, DK_BA(R1)           ; LOAD LOW ORDER 16 BITS OF UNIBUS
                                        ;   START ADDRESS INTO DK_BA REG

        ASHL    #-16, R3, R3            ; SHIFT UNIBUS ADDRESS RIGHT 16 BITS
        MOVL    DK_FUNC, R4             ; LOAD FUNCTION INTO R4 AND
        INSV    R3, #8, #2, R4          ; INSERT ADDRESS BITS 17 AND 16 FROM
                                        ;   R3 BITS 1:0 INTO BITS 9:8 OF
                                        ;   R4.  EXTENDED UNIBUS ADDRESS
                                        ;   BITS.
        MOVW    R4, DK_CS1(R1)          ; ISSUE FUNCTION AND GO TO RK611

4$:     TSTB    DK_CS1(R1)              ; WAIT FOR TRANSFER TO COMPLETE
        BGEQ    4$                      ;  NOT COMPLETE - KEEP WAITING.

;  THE RK611 HAS BEEN COMPLETED - THE UBA BUFFERED DATA PATH MUST NOW BE
;   PURGED.  THE PURGE IS REQUIRED TO MOVE ANY DATA REMAINING IN THE UBA
;   TO MEMORY AND TO INITIALIZE THE BUFFERED DATA PATH FOR ANY SUBSEQUENT
;   TRANSFERS.  THE PURGE IS ACCOMPLISHED BY SETTING THE BNE BIT OF THE
;   DATA PATH REGISTER USED BY THE TRANSFER.

;  COMPUTE OFFSET OF DATA PATH REGISTER USED FOR TRANSFER AND PUT IN R2

        ASHL    #2, DP_NUM, R2          ; MULTIPLY DP_NUM BY 4 -> R2
        ADDL    #UBA_DP0, R2            ; ADD IN OFFSET OF DATA PATH REG 0
        MOVL    #UBA_DP_BNE, UBA_BASE(R2); SET BNE BIT OF DATA PATH REGISTER
                                        ; USED BY THE TRANSFER.
        BITL    #UBA_DP_BTE, UBA_BASE(R2)
                                        ; TEST FOR ANY ERRORS THAT MAY HAVE
                                        ; OCCURRED WITHIN THE UBA BUFFER
                                        ; TRANSFER.
```

```
        BEQL    5$                      ;   CONTINUE IF THERE WERE NO ERRORS
        HALT                            ;   HALT FOR ERROR DETECTED BY DATA
                                        ;   PATH REGISTER.  UBA STATUS REGISTER
                                        ;   SHOULD CONTAIN ERROR BIT.

5$:     TSTW    DK_CS1(R1)              ; TEST FOR RK611 ERROR
        BGEQ    6$                      ;   CONTINUE IF THERE WERE NO ERRORS
        HALT                            ;  HALT FOR ERROR DETECTED IN RK611

6$:     MTPR    #BELL, #TXDB            ; RING BELL ON CONSOLE IF NO ERRORS
        HALT

;  THE TRANSFER PERAMETERS ARE SPECIFIED BELOW:

MEMSAD:         .LONG   4567    ; MEMORY START ADDRESS
BCOUNT:         .LONG   3210    ; NUMBER OF BYTES OF TRANSFER
MAP_REG:        .LONG   25      ; STARTING MAP REGISTER TO BE USED FOR TRANSFER.
DP_NUM:         .LONG   5       ; UBA DATA PATH TO BE USED FOR TRANSFER.
DRIVE:          .LONG   0       ; DRIVE NUMBER TO BE USED FOR TRANSFER
TRACK:          .LONG   2       ; STARTING TRACK
CYLINDER:       .LONG   4       ; STARTING CYLINDER
SECTOR:         .LONG   6       ; STARTING SECTOR
DK_FUNC:        .LONG   READ    ; DISK FUNCTION

        .END BEGIN
```


digital VAX 11/780

CHAPTER 18

VAX-11/780 MASSBUS SUBSYSTEM

FEATURES	BENEFITS
Direct memory access (DMA) data transfers	Eliminate processor intervention for high data throughput
32-byte silo data buffer for each MASSBUS	Permits transfers at rates up to 1.3 MB/s (2.0 MB/s with two controllers interleaved)
Built-in diagnostic features	Allow on-line diagnosis of the MASSBUS and MASSBUS peripherals
MASSBUS device registers are addressed like memory locations	Simplifies I/O programming

INTRODUCTION

The MASSBUS adapter (MBA) is the hardware interface between the synchronous backplane interconnect and the high speed MASSBUS storage devices. The MASSBUS is the communication path linking the MASSBUS adapter to the mass storage device drives.

The MASSBUS adapter performs the following functions:

- Mapping of addresses from virtual (program) to physical (SBI).
- Data buffering between main memory transfer to the MASSBUS and vice versa.
- Transfer of interrupts from MASSBUS device to the SBI.

The VAX-11/780 will support a maximum of four MASSBUS adapters, each adapter supporting up to eight device controllers. A MASSBUS adapter will support any combination of mass storage devices. Each magnetic tape controller will support up to eight tape drives. Each disk controller will support a single disk drive. Only one controller can transfer data at any one given time. The data transfer rate is dependent upon the particular mass storage device being accessed. Figure 18-1 illustrates a typical MASSBUS subsystem configuration.

The MASSBUS is comprised of 54 signal lines divided into two independent groups: the asynchronous control path (bus) and the synchronous data path (bus). Table 18-1 describes individual MASSBUS signal line function.

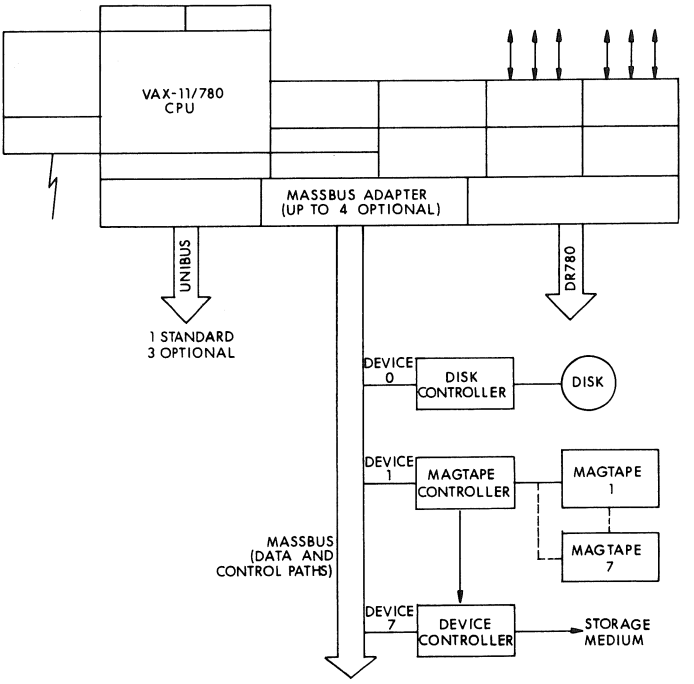


Figure 18-1 MASSBUS Subsystem Configuration

Table 18-1 MASSBUS Line Descriptions

SIGNAL LINE	DESCRIPTION
CONTROL BUS	
Control and Status (C00-15)	Transfers 16 parallel control or status bits to or from the drive.
Control Bus Parity (CPA)	Transfers odd control bus parity to or from the drive. Parity is simultaneously transferred with control bus data.
Drive Select (DS0-2)	Transfers a 3-bit binary code from the MBA to select a controller. The drive responds when the (unit) select switch in the controller corresponds to the transmitted binary code.

Register Select (RSO-4)	Transfers a 5-bit binary code from the MBA to select a particular drive register.
Controller to Drive (CTOD)	Indicates in which direction information is to be transferred on the control bus. For a controller-to-drive transfer, the MBA asserts CTOD; for a drive-to-controller transfer, the MBA negates CTOD.
Demand (DEM)	Asserted by the MBA to indicate a transfer is to take place on the control bus. For a controller-to-drive transfer, DEM is asserted by the MBA when data is present. For a drive-to-controller transfer, DEM is asserted by the MBA to request data and is negated when the data has been strobed from the control bus. In both cases, the RS, DS, and CTOD lines are asserted and allowed to settle before assertion of DEM.
Transfer (TRA)	Asserted by the drive in response to DEM. For a controller-to-drive transfer, TRA is asserted when the data is strobed and negated when DEM is removed. For a drive-to-controller transfer, TRA is asserted when the data is asserted on the bus and negated when the negation of DEM is received.
Attention (ATTN)	The drive asserts this line to signal the MBA of any change in drive status or an abnormal condition. ATTN is asserted any time a drive's ATA status bit is set. ATTN is common to all drives and may be asserted by more than one drive at a time.
Initialize (INIT)	Asserted by the MBA to initialize all drives on the bus. This signal is transmitted whenever the MBA receives an initialize command.
Fail (FAIL)	When asserted, this line indicates a power fail condition has occurred in the MBA or the MBA is in maintenance mode.
DATA BUS	
Data (D00-15)	These bidirectional lines transfer 16 parallel data bits between the MBA and drives.

Data Bus Parity (DPA)	Transfers an odd parity bit to or from the drive. Parity is simultaneously transferred with bits on the data bus.
Sync Clock (SCLK)	Asserted by the drive during a read operation to indicate when data on the data bus is to be strobed by the MBA. During a write operation SCLK is asserted to the MBA to indicate the rate at which data would be presented by the MBA on the data bus.
Write Clock (WCLK)	Asserted by the MBA to indicate when data written to the drive is to be strobed.
Run (RUN)	Asserted by the MBA to initiate data transfer command execution. During a data transfer, the drive samples RUN at the end of each sector. If RUN is still asserted, the drive continues the transfer into the next sector; if RUN is negated, the drive terminates the transfer.
End-of-Block (EBL)	Asserted by the drive at the end of each sector. For certain error conditions where it is necessary to terminate operations immediately, EBL is asserted prior to the normal time. In this case, the transfer is terminated prior to the end of the sector.
Exception (EXC)	Asserted by the drive or MBA to indicate an error condition during a data transfer command. EXC remains asserted until the trailing edge of the last EBL pulse.
Occupied (OCC)	Indicates acceptance of a valid data transfer command.

Figure 18-2 illustrates the MASSBUS signal line configuration.

MASSBUS ADAPTER OPERATION

The MASSBUS adapter consists of an SBI/MBA interface, internal registers, control paths and data paths. Figure 18-3 is a simplified block diagram of the MBA. A tristate internal bus connects the SBI module to the internal registers, control paths, and data paths, and provides for the passage of data to the various functional blocks.

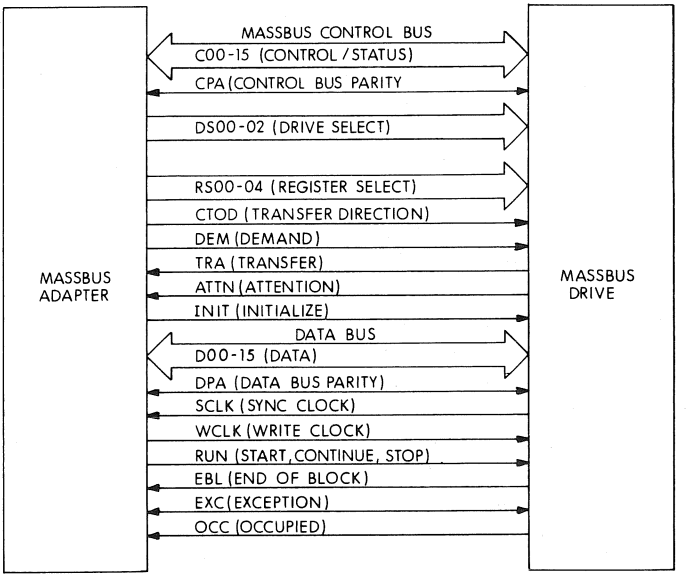


Figure 18-2 MASSBUS Signal Line Configuration

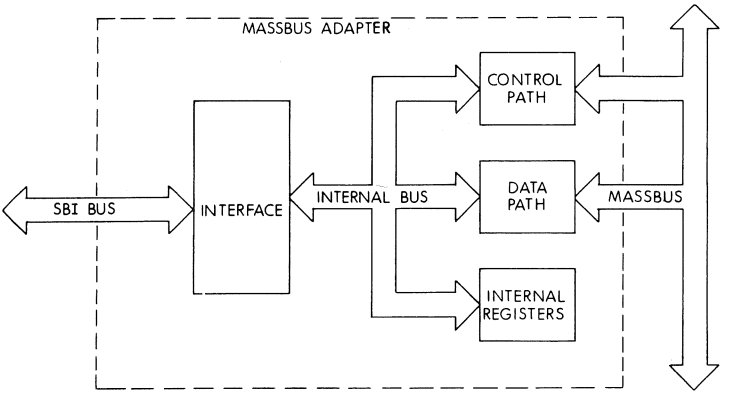


Figure 18-3 MASSBUS Adapter

The MBA accepts and executes commands from the CPU and reports the necessary status changes and fault conditions to the CPU. The MBA can transfer register data or a block of data to or from a MASSBUS device. A 256 X 32-bit (bits <30:21> are not writable) RAM stores the physical page addresses of the block of data to be transferred. The memory data (64 bits) will be sent in words (16 bits) to the MASSBUS drive in the order of the first word (bits 15 to 0), followed by the second word (bits 31 to 16) of a long word. Special diagnostic features are built in the hardware to allow on-line diagnosis of the MBA and MASSBUS drives.

The MBA is capable of handling a MASSBUS drive with a maximum data transfer rate of 16 bits per μsec via the 16-bit wide MASSBUS data path. The MASSBUS adapter controls data transfers between MASSBUS devices and physical memory. A MASSBUS adapter can transfer 16 bits at a time to a mass storage device or it can receive 16 bits at a time from a MASSBUS drive. The MBA contains a 32-byte buffer used to store data enroute to either main memory or mass storage. Transfers (data only) along the SBI, to or from main memory, occur in 64-bit (8-byte) increments. Therefore, there are four MASSBUS transfers (16 bits each) per SBI transaction. The MASSBUS adapter will accept only aligned longword reads and writes to its external or internal registers. An attempt to address a nonexistent register in the MASSBUS adapter will prompt a no-response confirmation.

MBA Registers

There are two sets of registers in the MBA address space: internal and external. The MBA internal registers are the registers which are physically located in the MBA. The external registers are located in the MASSBUS drives and are drive-dependent.

There are eight internal registers and a 256 × 32-bit RAM. The internal registers primary function is to monitor MBA and operating status conditions. The internal registers also control certain phases of the data transfers between the SBI and the MASSBUS device such as:

- maintaining a byte count to ensure that all of the data to be transferred has been accounted for
- converting virtual addresses to physical addresses for referencing data in memory

The eight internal registers are:

MBA Configuration Register (CSR)

MBA Control Register (CR)

MBA Status Register (SR)

MBA Virtual Address Register (VAR)

MBA Byte Count Register (BCR)
MBA Diagnostic Register (DR)
MBA Selected Map Register (SMR)
MBA Command Address Register (CAR)

NOTE

The selected map register and the command address register are read only and are valid only during data transfers.

The MBA contains 256 32-bit map registers which are used to map program virtual addresses into SBI physical addresses. Bits <30:21> of the map register are reserved and are not writable. The mapping registers allow transfers to or from contiguous or non-contiguous physical memory. Figure 18-4 illustrates the mapping of a virtual address to an SBI address.

CONTROL PATH

The control path handles the transfer of control data to and from the MASSBUS devices. Certain sections of the MBA address space map into registers physically located within MASSBUS devices. The MASSBUS control path is used to communicate with these data path registers.

The data path controls the data transferred to and from the MASSBUS device and the SBI. The 32-bit SBI data word is divided into 16-bit (2-byte) segments required as data on the MASSBUS. When performing a read from MASSBUS device the data path assembles the two 8-bit bytes from the MASSBUS into the 32-bit SBI format. A silo and input/output data buffer provide the means for smoothing the data transfer rate. The data path also contains a write check circuit which can be used under program control to verify the accuracy of the data transfer function.

MBA ACCESS

Each SBI device (NEXUS) is assigned a 2048, 32-bit longword (8 KB) control address space. This space is accessible as part of the SBI I/O longword address space. The command/address format used to access the MBA registers is illustrated in Figure 18-5.

Bit <29> = 1	I/O Address space
Bits <28:17>	All zeros

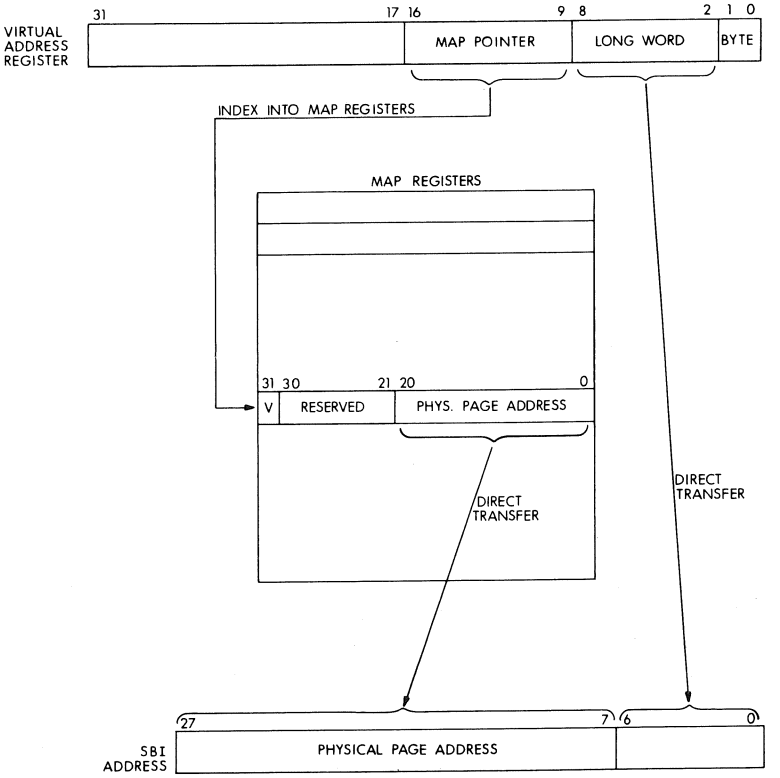


Figure 18-4 Virtual to SBI Address Translation

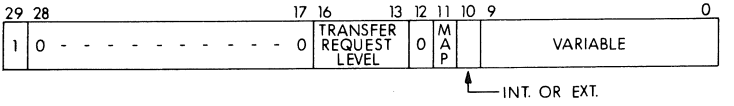


Figure 18-5 MASSBUS Adapter Addressing Format (Physical Byte Address)

Bits <16:13>	Transfer request number of this MBA
Bits <11:10>	
0 0	MBA internal register Bits<9:5> = must be zero Bits<4:0> = register select offset
0 1	MBA external register Bits<9:7> = device select Bits<6:0> = register select
1 0	MBA MAP Bits<9:0> = MAP address
1 1	Invalid (No response to an address with these bits on)

INTERNAL REGISTERS

The MBA internal registers are described as follows:

MBA Configuration/Status Register (Byte Offset=0)

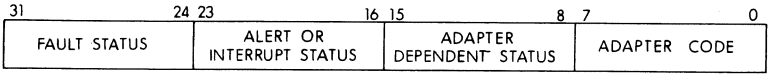


Figure 18-6 MBA Configuration/Status Register

Bit: 31 **Name:** SBI parity error
Function: Set when an SBI parity error is detected. Cleared by power fail or the deassertion of fault signal. Setting of this bit will cause fault to be asserted on SBI.

Bit: 30 **Name:** Write data sequence (WS)
Function: Set when no write data is received (neither tag = write data nor ID) following a write command. Cleared by power fail or the deassertion of fault signal. The setting of this bit will cause the assertion of fault on SBI.

Bit: 29 **Name:** Unexpected read data (URD)
Function: Set when read data is received when it is not expected.

Cleared by power fail or the deassertion of fault signal. The setting of this bit will cause assertion of fault on SBI.

Bit: 28 **Name:**
Function: This bit must be zero.

Bit: 27 **Name:** Multiple transmitter (MT)
Function: Set when the ID on the SBI does not agree with the ID transmitted by MBA while MBA is transmitting information on the SBI. Cleared by power fail or the de-assertion of fault signal. The setting of this bit will cause the assertion of fault on SBI.

(Fault signal will be asserted at the normal confirmation time for one cycle if MBA detects one of the fault conditions. The negation of the fault signal on the SBI will clear all the fault status bits).

Bit: 26 **Name:** XMTFLT
Function: Set when SBI fault is detected at the second cycle after MBA transmits information to the SBI. Cleared by power fail or the deassertion of fault signal.

Bit: 25 **Name:** Zero
Function: Reserved for future use.

Bit: 23 **Name:** Adapter power down (PD)
Function: Set when the MBA receives assertion of AC LO. Clear when MBA power goes up. Cleared by assertion of INIT, UNJAM, DC LO, or writing one to this bit. The setting of this bit will cause interrupt to CPU.

Bit: 22 **Name:** Adapter power up (PU)
Function: Set when MBA receives the deassertion of AC LO. Reset when MBA power goes down. Cleared by assertion of INIT, UNJAM, DC LO or writing a one to this bit. The setting of this bit will set IE bit and interrupt CPU.

Bit: 21 **Name:** Over temperature (OT)
Function: Zero

Bit: 20:8 **Name:** All zeros
Function: Reserved for future use.

Bit: 7:0 **Name:**
Function: Each adapter is assigned a unique code identifying it. MBA adapter code is: bits <7:0>=00100000.

MBA Control Register (Byte Offset =4)

Bit: 31:4 **Name:** All zeros
Function: Reserved for future use.

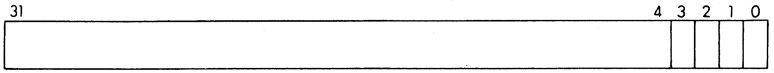


Figure 18-7 MBA Control Register

Bit: 3 **Name:** MB Maintenance Mode
Function: The setting of this bit will put MBA in the maintenance mode which will allow the diagnostic programmer to exercise and examine the MASSBUS operations without a MASSBUS device. When this bit is set, MBA will block RUN, DEM, and assert FAIL to MASSBUS so that all the devices on MASSBUS will detach from the MASSBUS. The MBA cannot be put in maintenance mode while a data transfer is in progress.

Bit: 2 **Name:** Interrupt Enable
Function: Set by writing a one or power up which allows MBA to interrupt CPU when certain conditions occur. Cleared by writing zero or INIT.

Bit: 1 **Name:** ABORT
Function: Abort data transfer. Write one to set. The setting of this bit will initiate the data transfer abort sequences which will stop sending commands, stop address and byte counter.

- Negate Run
- Assert EXEC to MASSBUS
- Wait for EBL
- Set DTABT to one at the trailing edge of EBL
- Interrupt CPU if IE bit is one

This bit will be cleared by writing a zero, INIT or UNJAM.

Bit: 0 **Name:** Initialization (INIT)
Function: The bit is self-clearing. It will always read as zero. The setting of this bit will:

- Clear status bits in MBA Configurator register
- Clear ABORT and IE in MBA Control register
- Clear MBA Status register
- Clear MBA Byte Count register
- Clear control and status bits of diagnostic registers
- Cancel all pending commands except Read Data Pending
- Abort data transfer
- Assert MASSBUS INIT

MBA Status Register (Byte Offset=8)

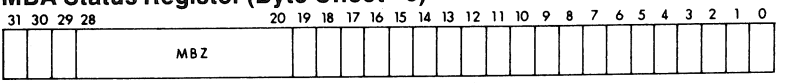


Figure 18-8 MBA Status Register

Bit: 31 **Name:** DTBUSY
Function: Data transfer busy. Bit is set when a data transfer command is received. It is cleared when data transfer is terminated normally or when a data transfer is aborted.

Bit: 30 **Name:** NRCONF
Function: No response confirmation. This bit is set when the MBA receives a no response confirmation for the read command or write command and write data sent to the SBI. It is cleared by writing a one to the bit or INIT. The setting of this bit will cause retry of the command.

Bit: 29 **Name:** CRD
Function: Corrected read data. This bit is set when TAG of read data received from memory is CRD. It is cleared by writing a one to this bit or by INIT.

Bit: 28:20 **Name:** All zeros.
Function: Reserved for future use.

Bit: 19 **Name:** PGE
Function: The PGE bit is set when one or more of the following conditions exists:

Program tries to initiate a Data Transfer when MBA is currently performing one.

Program tries to load MAP, VAR, or Byte counter when MBA is currently performing a Data Transfer operation.

Program tries to set MB Maintenance Mode during a Data Transfer operation.

The bit is cleared by writing a one. The setting of this bit will cause an interrupt to the CPU if IE is set.

Bit: 18 **Name:** NFD
Function: Nonexisting drive. This bit is set when drive fails to assert TRA within 1.5 μsecs after assertion of DEM. The bit is cleared by writing a one to the bit. The setting of this bit will send zero read data back to the SBI, and interrupt CPU if IE is set.

Bit: 17 **Name:** MCPE

Function: MASSBUS control parity error. This bit is set when a MASSBUS control parity error occurs. It is cleared by writing a one to the bit. The setting of this bit will cause an interrupt to CPU if IE is set.

Bit: 16 **Name:** ATTN
Function: Attention from MASSBUS. Asserted when the attention line in the MASSBUS is asserted. The assertion of this bit will cause an interrupt to the CPU if IE is set.

Bit: 15:14 **Name:** All zeros.
Function: Reserved for future use.

Bit: 13 **Name:** DT CMP
Function: Data transfer completed. This bit is set when the data transfer is terminated either due to an error or normal completion. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause an interrupt to the CPU if IE bit in control register is set.

Bit: 12 **Name:** DTABT
Function: Data transfer aborted. This bit is set with the trailing edge of the EBL when data transfer has been aborted. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause an interrupt to the CPU if IE bit is set.

Bit: 11 **Name:** DLT
Function: Data late. This bit is set when:

1) For a write data transfer or write check data transfer, data buffer is empty when WCLK is sent to MASSBUS.

or

2) for a read data transfer, the data buffer is full while SCLK is received from MASSBUS. This bit is cleared by writing a one to it or INIT. The setting of this bit will cause the data transfer operation to be aborted.

DLT will most likely be set if the system is in single step operation and if the MBA is not in maintenance mode.

Bit: 10 **Name:** WCK UP ERR
Function: Write Check Upper Error. This bit is set when a compare error is detected in the upper byte while MBA is performing a write check operation. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 9 **Name:** WCK LWR ERR
Function: Write Check Lower Error. This bit is set when a compare error is detected in the lower byte while MBA is performing a write check operation. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 8 **Name:** MXF
Function: Miss transfer error. This bit is set when an SCLK or OCC is not received within 500 μsec after data transfer busy is set. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause an interrupt to the CPU if IE bit in control register is set.

Bit: 7 **Name:** MBEXC
Function: MASSBUS Exception. This bit is set when EXC is received from MASSBUS. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 6 **Name:** MDPE
Function: MASSBUS data parity error. This bit is set when the MASSBUS data parity error is detected during a read data transfer operation. It is cleared by writing a one to the bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 5 **Name:** MAPPE
Function: Page Frame Map Parity Error. This bit is set when a parity error is detected on the page frame number read from the PF map. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 4 **Name:** INVMAP
Function: Invalid map. This bit is set when the valid bit of the next page frame number is zero when byte count is not zero. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 3 **Name:** ERR CONF
Function: Error Confirmation. This bit is set when the MBA receives an error confirmation for the read command or write command. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 2 **Name:** RDS
Function: Read Data Substitute. This bit is set when the tag of the read data received from memory is read data substitute. It is cleared by writing a one to this bit or INIT. The setting of this bit will cause the data transfer operation to be aborted.

Bit: 1 **Name:** IS TIMEOUT
Function: Interface Sequence Timeout. An interface sequence is defined as the time from when arbitration for the SBI is begun until:

1) ACK is received for a command address transfer that specifies read or,
2) ACK is received for a command address transfer that specifies write and ACK is also received for each transmission of write data

or,

3) ERR confirmation is received for any command/address transfer. The maximum timeout is 102.4 μsecs. The setting of this bit will cause data transfer abort. Cleared by writing a one to this bit or INIT.

Bit: 0 **Name:** RD TIMEOUT
Function: Read Data Timeout is defined as the time from when an interface sequence that specifies a read command is completed to the time that the specified read data is returned to the commander. The maximum time out is 102.4 μsecs. The setting of this bit will cause data transfer abort. Cleared by writing a one to this bit or INIT.

MBA Virtual Address Register (Byte Offset=12)

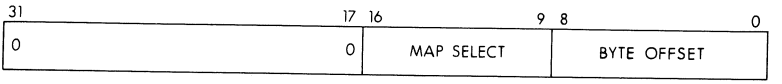


Figure 18-9 MBA Virtual Address Register

The program must load an initial virtual address (pointing to the first byte to be transferred) into this register before a data transfer is initiated. Bits 9 through 16 select one of 256 map registers. The contents of the selected map register and the values in bits 0 through 8 are used to

assemble a physical SBI address to be sent to memory. Bits 0 through 8 indicate the byte offset into the page of the current data byte. Note the MBA virtual address register is incremented by 8 after every memory read or write and will not point to the next byte to be transferred if the transfer does not end on a quadword boundary. (It will point 8 bytes ahead.) Also upon a write check error, the virtual address register will not point to the failing data in memory due to the preloading of the silo data buffer. The virtual address of the bad data may be found by determining the number of bytes actually compared to the MASSBUS (the difference between bits 16 to 31 of RS04 and their initial value) and adding that difference to the initial virtual address.

MBA Byte Counter (Byte Offset=16)

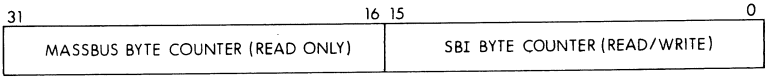


Figure 18-10 MBA Byte Counter

Program loads the 2's complement of the number of bytes for the data transfer to bits <15:0> of this register. MBA hardware will load these 16 bits into bits <31:16>. Bits <31:16> serve as the byte counter for the number of bytes transferred through the SBI interface. The starting byte count with 16 bits of 0's is the maximum number of bytes of a data transfer.

Diagnostic Register (Byte Offset=20)

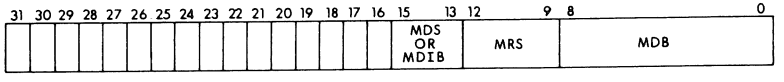


Figure 18-11 Diagnostic Register

The diagnostic register may only be read or written while in maintenance mode. Care must be taken while reading or bit-setting this register to insure that the data path is not loading the silo. If the data path is loading the silo while this register is read, the data may be altered.

Bit: 31 **Name:** IMDPG
Function: Invert MASSBUS Data Parity Generator.

Bit: 30 **Name:** IMCPG
Function: Invert MASSBUS Control Parity Generator.

Bit: 29 **Name:** IMAPP
Function: Invert Map Parity.

Bit: 28 **Name:** BLKSCOM
Function: Block Sending Command to SBI. During a data transfer, the setting of this bit will eventually cause a DLT bit set and CPU interrupt.

Bit: 27 **Name:** SIMSCLK
Function: Simulate SCLK. When MMM bit is set, writing a one to this bit will simulate the assertion of SCLK, and writing a zero to this bit will simulate the deassertion of SCLK.

Bit: 26 **Name:** SIMEBL
Function: Simulate EBL. When MMM bit is set, writing a one and writing a zero to this bit will simulate the assertion and deassertion of EBL.

Bit: 25 **Name:** SIMOCC
Function: Simulate OCC. When MMM bit is set, writing a one and writing a zero to this bit will simulate the assertion and deassertion of OCC.

Bit: 24 **Name:** SIMATTN
Function: Simulate ATTN. When MMM bit is set, writing a one and writing a zero to this bit will simulate the assertion and deassertion of ATTN.

Bit: 23 **Name:** MDIB SEL
Function: Maintenance MASSBUS Data Input Buffer Select. When the bit is set to one, the upper eight bits (B<15:8>) of MDIB will be sent out from B<7:0> of Diagnostic Register if the Diagnostic Register is read. When this bit is zero, the lower eight bits (B<7:0>) of MDIB will be sent out from B<7:0> of Diagnostic Register if a bit is read.

Bit: 22:21 **Name:** MAINT ONLY
Function: Read/write with no effect. (Used to test writability of these bits).

Bit: 20 **Name:** MFAIL
Function: MASSBUS Fail (read-only). Fail is asserted when MMM is set.

Bit: 19 **Name:** MRUN
Function: Maintenance MASSBUS Run (read-only).

Bit: 18 **Name:** MWCLK
Function: Maintenance MASSBUS WCLK (read-only).

Bit: 17 **Name:** MFXC
Function: Maintenance MASSBUS FXC (read-only).

Bit: 16 **Name:** MCTOD
Function: Maintenance MASSBUS MCTOD (read-only).

Bit: 15:13 **Name:** MDS
Function: Maintenance MASSBUS Device Select (read-only).

Bit: 12:8 **Name:** MRS
Function: Maintenance MASSBUS Register Select (read-only).

Bit: 7:0 **Name:** U/L MDIB
Function: Maintenance Upper/Lower MDIB.

Selected Map Register (Byte Offset =24)
This register is read-only and has the same format as a map register but is valid only when DT Busy is set. This is the contents of the map register pointed to by bits 16 through 9 of the virtual address register.

Command Address Register (Byte Offset=28)
This register is read-only and valid only when DT Busy is set. It is the value of bits <31:0> of the SBI during the Command/Address part of the MBA's next data transfer cycle.

MBA External Register (Byte Offset=400 to 7FC)
External registers are MASSBUS device-dependent. Each device has a maximum of 32 registers.

MBA Map (Byte Offset=800 to BFC)
Bit: 31 **Name:** Valid Bit

Bit: 30:21 **Name:**
Function: Zeros. Reserved for future use.

Bit: 20:0 **Name:** Physical Page Frame Number

The MBA contains 256 map registers, each of which may be selected by address bits 0 to 9 when bits <11:10> are 10. Map registers can

only be written when there is no data transfer operation in progress. A write to a map register during a data transfer will be ignored and cause the setting of PGE.

Data Transfer Program Flow

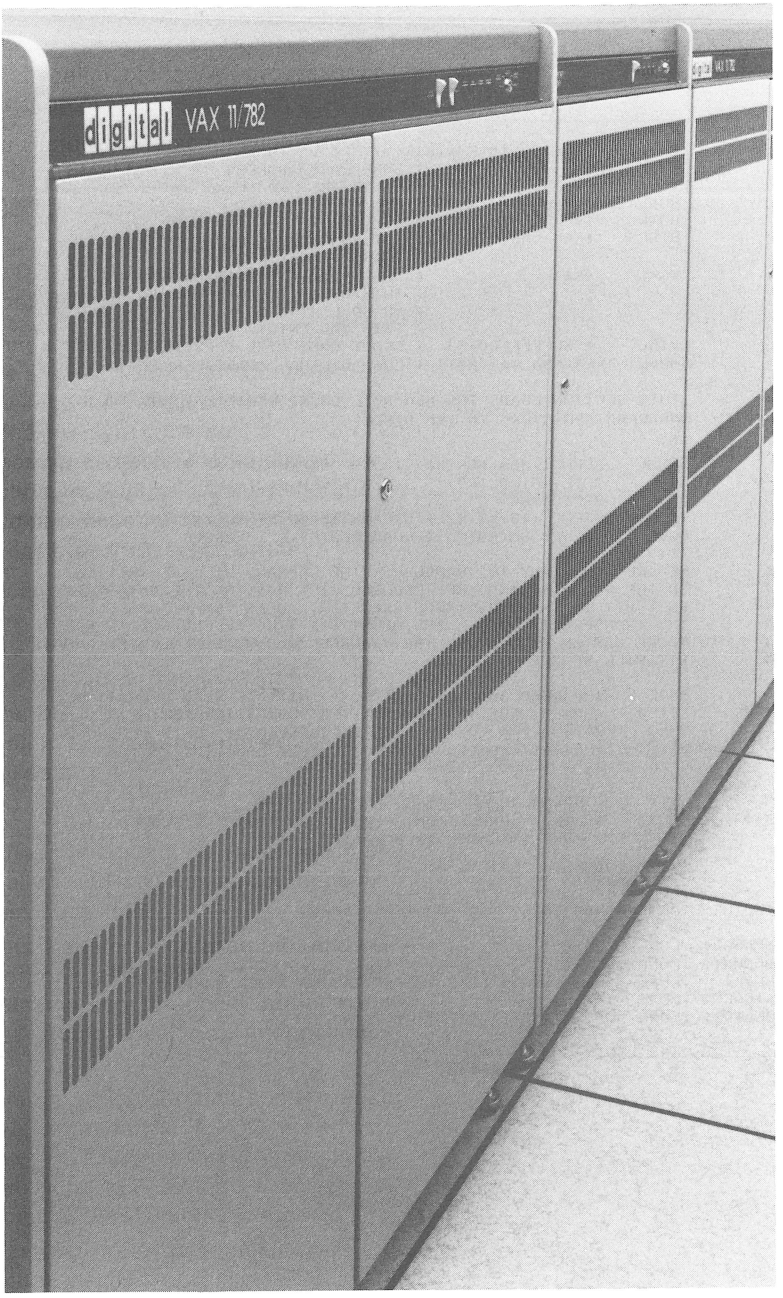
1) Initialize MASSBUS Adapter.
2) Mount pack into drive.
3) Start drive spinning.
4) Wait for ready light.
5) Issue Pack ACK to drive.
6) Load desired cylinder, sector, track, and registers in drive.
7) Load starting virtual address into MBA's virtual address register.
8) Load 2's complement of number of bytes to be transferred into byte count register in MBA.
9) Load starting map (pointed to by bits <9:16> of VAR) with physical page address.
10) Load successive maps with physical addresses to rest of pages.
11) Issue read/write command to drive.

EXAMPLE

Presented is a program to read data from the RP05/RP06 disk subsystem into memory. The program is written in the VAX-11 MACRO assembly language. It is not meant to run with memory management enabled, and will not run under VAX/VMS. This program illustrates the procedures involved in setting up the MASSBUS adapter to transfer bytes of data to memory. In order to run the program, it must be loaded from the floppy disk by the console into memory. Initially, the program can be assembled and linked under VAX/VMS and then transferred to the floppy disk using the RSX-11M utility program FLX.

```
;       PROGRAM TO READ FROM THE RP05/6 INTO MEMORY
;
;       SYMBOL DEFINITIONS

        MBA_BASE=^X20010000     ; MBA AT TR=8
        MBA_CSR =^X0            ;  OFFSET TO SBI STATUS REGISTER
        MBA_CR  =^X4            ;  OFFSET TO MBA CONTROL REGISTER
         MBA_INIT=^X1           ;   INITIALIZE BIT
        MBA_SR  =^X8            ;  OFFSET TO MBA STATUS REGISTER
         MBA_DT_BUSY=^X80000000 ;   DTBUSY STATUS BIT
         MBA_DT_ABORT=^X2000    ;   DTABORT STATUS BIT
        MBA_VAR =^XC            ;  OFFSET TO MBA VIRTUAL ADDRESS REGISTER
        MBA_BCR =^X10           ;  OFFSET TO MBA BYTE COUNT REGISTER
        MBA_MAP0=^X800          ;  OFFSET TO FIRST MAP REGISTER
         MAP_VALID=^X80000000   ;   VALID BIT IN MAP REGISTER
        RP1_CSR =^X480          ;  OFFSET TO CSR ON RP DRIVE 1
         RP_SET_VV_CMD=^X13     ;   SET VOLUME VALID COMMAND
         RP_READ_CMD  =^X39     ;   READ COMMAND
        RP1_DA  =^X494          ;  OFFSET TO SECTOR AND TRACK REGISTER
        RP1_OF  =^X4A4          ;  OFFSET TO TRACK OFFSET REGISTER?
         RP_16BIT_FORMAT=^X1000 ;   16BIT FORMAT ON DISC SURFACE
        RP1_DC  =^X4A8          ;  OFFSET TO DESIRED CYLINDER REGISTER

;       THIS SECTION LOADS 0 WITH THE ADDRESS OF THE MBA,
;       INITIALIZES THE MBA, SET VOLUME VALID AND THE 16BIT
;       FORMAT BITS IN THE DRIVE.  THESE OPERATIONS (SET VV
;       AND 16BIT) ONLY NEED TO BE DONE WHEN A NEW PACK IS
;       MOUNTED OR WHEN POWER COMES UP.

BEGIN:  MOVL    #MBA_BASE,R0    ; LOAD MBA'S ADDRESS INTO R0
        MOVL    #MBA_INIT,MBA_CR(R0)  ; INITIALIZE MBA
        MOVL    #RP_SET_VV_CMD,RP1_CSR(R0) ; SET VOLUME VALID
        MOVL    #RP_16BIT_FORMAT,RP1_OF(R0) ; SET 16BIT FORMAT

;       THIS SECTION LOADS THE MAPS.  THE STARTING PHYSICAL
;       ADDRESS IS SHIFTED RIGHT 9 BITS TO BECOME A PHYSICAL
;       PAGE ADDRESS.  THE MAP VALID BIT IS ORED IN AND MAP
;       REGISTER 0 IS LOADED.  SUCESSIVE MAP REGISTERS ARE
;       LOADED WITH THE PHYSICAL PAGE ADDRESSES OF THE REST
;       OF THE DATA AREA IN MEMORY.  THEN THE VIRTUAL ADDRESS
;       REGISTER IS LOADED WITH THE OFFSET TO THE FIRST
;       BYTE OF THE DATA AREA.

        MOVL    BCOUNT,R2       ; # OF BYTES TO TRANSFER
        MOVL    MEMSAD,R3       ; STARTING BYTE ADDRESS OF TRANSFER
        MOVAL   MBA_MAP0(R0),R1 ; R1 HAS ADDRESS OF FIRST MBA MAP
        ASHL    #-9,R3,R4       ; TURN STARTING ADDRESS INTO PAGE ADDRESS
```

```
1$:     BISL3   #MAP_VALID,R4,(R1)+ ; SET VALID BIT AND MOVE INTO MAP
        INCL    R4                  ; BUMP PAGE ADDRESS
                                    ; THERE ARE 200 (HEX) BYTES PER PAGE
        SUBL    #^X200,R2           ; SET UP ALL MAPS?
        BGTR    1$                  ; NO, SO SET UP NEXT MAP
        BISL3   #MAP_VALID,R4,(R1)+ ; ENABLE ONE MORE MAP SINCE
                                    ; TRANSFER MAY NOT BE PAGE ALIGNED
        CLRL    (R1)                ; CLEAR VALID BIT IN UNUSED MAP
                                    ; THIS WILL STOP THE MBA IF IT SHOULD
                                    ; DO SOMETHING WRONG AND TRY TO USE
                                    ; THIS MAP ENTRY
        BICL    #^XFFFFFE00,R3      ; LEAVE ONLY BYTE OFFSET INTO PAGE
        MOVL    R3,MBA_VAR(R0)      ; LOAD VIRTUAL ADDRESS REGISTER.

;       THIS SECTION LOADS THE MBA BYTE COUNT REGISTER, AND THE
;       ADDRESS REGISTERS IN THE DISC

        MNEGL   BCOUNT,MBA_BCR(R0) ; 2'S COMPLEMENT OF # BYTES TO XFER
        MOVL    SCYL,RP1_DC(R0) ; SET DESIRED CYLINDER
        MOVL    SECTOR,R2
        INSV    STRACK,#8,#8,R2 ; CREATE IMAGE OF SECTOR/TRACK REGISTER
        MOVL    R2,RP1_DA(R0)   ; LOAD REGISTER IN DRIVE

;       WE ARE NOW READY TO TRANSFER.  THE COMMAND WILL BE WRITTEN
;       TO THE RP0_CSR, AND THE PRROGRAM WILL MONITOR THE DATA TRANSFER
;       BUSY BIT IN THE MBA STATUS REGISTER.  WHEN THE BIT IS CLEARED
;       THE TRANSFER IS OVER.  THE ABORT BIT WILL BE CHECKED TO SEE IF
;       THERE WAS AN ERROR.  IF THE TRANSFER WAS SUCESSFULL THE ERROR
;       BIT WILL BE CLEAR.

        MOVL    #RP_READ_CMD,RP1_CSR(R0) ; INITIATE READ OPERATION
2$:     BITL    #MBA_DT_BUSY,MBA_SR(R0) ; DTBUSY STILL SET?
        BNEQ    2$                  ; YES -> TRANSFER STILL IN PROGRESS
        BITL    #MBA_DT_ABORT,MBA_SR(R0) ; XFER ABORTED?
        BNEQ    3$                  ; YES

        HALT    ; NORMAL COMPLETION. SUCESS
3$:     HALT    ; ERROR COMPLETION. MBA STATUS REGISTER SHOULD
                ; CONTAIN ERROR INFORMATION.


;  THE STARTING ADDRESSES ARE SPECIFIED BELOW

MEMSAD: .LONG   3456                ; MEMORY STARTING ADDRESS
BCOUNT: .LONG   4321                ; NUMBER OF BYTES TO TRANSFER
SCYL:   .LONG   322                 ; STARTING CYLINDER NUMBER
STRACK: .LONG   4                   ; STARTING TRACK
SECTOR: .LONG   6                   ; STARTING SECTOR

        .END BEGIN
$
```


digital VAX 11/782

CHAPTER 19

INTERCONNECTS AND THE VAX-11/782 ATTACHED PROCESSOR SYSTEM

OVERVIEW

All VAX-11/780 hardware system elements, such as the CPU, main memory, and MASSBUS and UNIBUS adapters, are connected to the Synchronous Backplane Interconnect (SBI) by devices called interconnects. In the case of the MASSBUS and UNIBUS adapters, the interconnect also allows the connection of various DIGITAL- and user-designed peripherals to the system via the appropriate bus structure.

The MASSBUS and UNIBUS adapters interconnects are designed to process data transfers at rates up to the potentials of the compatible peripherals. However, the SBI operates at even faster speeds, up to 13.3 MB/second, and in 32-bit versus 16-bit increments. To permit VAX users with exceptionally fast I/O requirements to take advantage of the speed and capabilities of the SBI, a special high performance 32-bit parallel interface, the **DR780**, is available.

Other VAX users with multiprocessor configurations working on complex and intricate problems can enhance the capabilities of their systems with the VAX-11/780 multiport memory option, **MA780**. The user can utilize shared memory among up to four processors with a throughput rate of up to 11 MB/second.

For additional computational power, the **VAX-11/782** attached processor upgrade or packaged systems can provide a significant performance improvement over a single VAX-11/780 processor. The VAX-11/782 is a tightly-coupled asymmetrical multiprocessor system based on the MA780 shared memory subsystem. It is comprised of two VAX-11/780 CPUs and supports up to 8 MB of MA780 shared memory.

This chapter will discuss the features, benefits and operation of these optional, very high performance VAX-11/780 interconnects and the VAX-11/782 attached processor computer system.

DR780 HIGH PERFORMANCE 32-BIT PARALLEL INTERFACE

Features	Benefits
6.67 MB/second transfer rate	Exceptionally fast, full 32-bit I/O operations
Command chaining	Allows multiple I/O operations to occur without software intervention
Dynamic memory mapping performed in hardware	Reduced I/O overhead; applications programs and I/O driver completely independent from memory mapping
Full VAX/VMS software support	Library of high-level language support routines and complete I/O driver reduce software development time for customer applications
Symmetric point-to-point interconnect	Enables two VAX-11/780 systems to be connected through two DR780 options in a user written application
Separate control and data interconnects	Provides for concurrent 32-bit data and 8-bit control transfers
User generated command packets processed without operating system intervention	Provides a direct link between the DR780 and the user process
Unrestricted maximum transfer size	Transfers can be of indefinite length, limited only by the size of physical memory

INTRODUCTION

The DR780 is a very high performance interface that allows user devices to be connected directly to the VAX-11/780 SBI. It is capable of transferring data to and from memory at speeds of up to 6.67 MB per second. The DR780 also permits intelligent 32-bit parallel data transfers to a second DR780-equipped VAX-11/780 system, enabling high speed interprocessor communication.

The DR780 is fully software supported by the VAX/VMS operating system with a simple, easy-to-use I/O driver and a library of high level language support routines. This allows the user to avoid much of the work and time associated with writing device handlers.

Data verification is supported on the DR780 by parity checking on both control and data transfers. Other DR780 features that enhance system integrity and maintainability include: error detection and logging, address range checking on data transfers and on-line diagnosis.

Applications that previously required very complex and time-consuming hardware and software development on the part of the user can be much more readily implemented with the DR780.

Figure 19-1 shows a typical VAX-11/780 configuration with a DR780 incorporated in block diagram form.

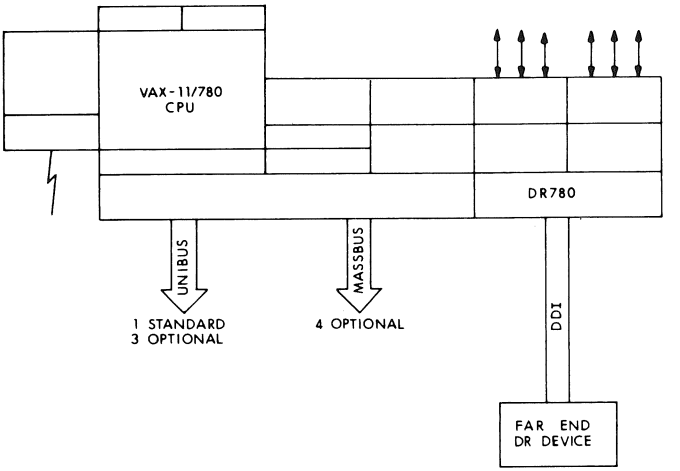


Figure 19-1 VAX-11/780 Configured with DR780

DR32 DEVICE INTERCONNECT (DDI)

The DR780 is an interface adapter that connects the internal memory bus of the VAX-11/780 processor to a user-accessible bus called the DR32 Device Interconnect (DDI). Two DR780s can be connected to form a VAX-11/780 processor-to-processor link. Figure 19-2 shows the relationship of the DR780 to the VAX-11/780 and the DDI.

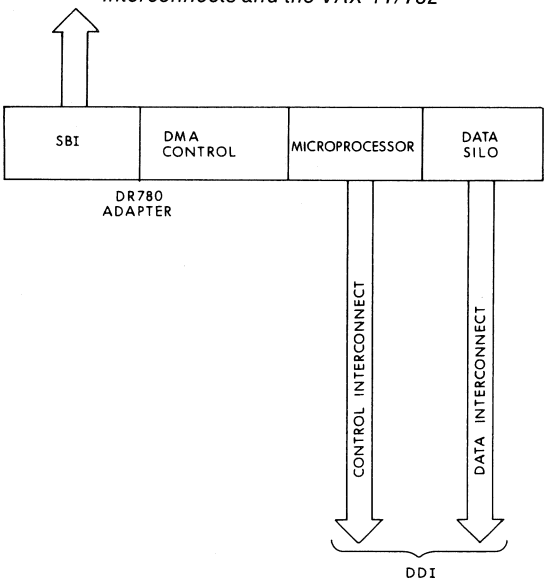


Figure 19-2 DR780-DDI Relationship

As a general purpose data port, the DR780 is a capable of moving continuous streams of data to or from memory at high speed. Data moved from a user device to disk storage must go through an intermediate buffer in main memory.

The DR32 Device Interconnect (DDI) is a bidirectional bus for the transfer of data and control signals. Control signals sent over the DDI are asynchronous and interlocked; data transfers are synchronized with clock signals. Any connection to the DDI is called a DR-device. The DDI provides a point-to-point connection between two DR-devices, one of which must be a VAX processor. The DR-device connected to the external end of the DDI is called the far end DR-device.

The DDI actually consists of two separate and independent interconnects: one for control signals and one for data. The control interconnect is an asynchronous 8-bit bidirectional path for transferring control information to and from a user device. A great deal of flexibility for the design of the user device is available since the width of the control interconnect (8 bits) enables the DR780 to address up to 256 individual registers. These registers can pass various kinds of information, such as buffer descriptors, commands, or status, to the user process.

The data interconnect is a synchronous 32-bit bidirectional path for data which is synchronized to a single clock. The user can choose to use the internal clock to the DR780 or provide one in the far end device. The transfer rate of the DR780 is selectable under program control in the range of 0.156 to 6.67 MB/sec. Actual throughput obtained, however, is configuration and application dependent.

Command Chaining

Command chaining is the execution of commands without software intervention for each command. Commands are chained in the sense that they follow each other on a queue. After a QIO function starts the DR780, any number of DR780 commands can be executed during that QIO operation. This process continues until the transfer is halted (a command packet is fetched that specifies a Halt command) or an error occurs.

The DR780 can perform **dynamic command chaining**. This means that commands can be added and even deleted from the command queue while the DR780 is operating. Four features of the DR780 allow command chaining as follows:

- **Fetches commands continuously from main memory**—The DR780 is an intelligent controller whose microprocessor requests command information and transfers data continuously from the virtual memory of the user process without program intervention. This is accomplished through the use of queues.
- **Direct communication between the DR780 and the user process via queues**—This feature provides for greater efficiency and flexibility by allowing the user process to control the DR780 directly. It also means that the I/O driver can be the same for a wide range of applications. Since the user process has the task of building command packets, the I/O driver just performs privileged tasks for the user process, such as supplying the DR780 with the address range of operation, fielding interrupts and aborting the current operation. Also, the time for the user process to supply the DR780 with a command packet or a data buffer is minimized.
- **Dynamic memory mapping**—This is the ability to perform continuous data transfers of arbitrary length without having to reload address translation map registers. Data transfers are specified as virtual addresses which the DR780 dynamically translates to physical address locations. This eliminates the need for mapping registers and allows for very large data transfers. The DR780 buffer size is limited only by the amount of physical memory available (see data chaining). The DR780 uses the same page table entries as the CPU.

- **Concurrent data and command transfers on the DDI**—The DDI's two separate and independent interconnects allow concurrent operations to occur, such as establishing the next command sequence in parallel with completing the current data transfer. Thus, the DR780 can be ready for the next data transfer immediately upon the completion of the current one.

Data Chaining

Command packets can specify data chaining. In data chaining, a number of main memory buffers appear as one large buffer to the far end DR-drive. Data chaining is completely transparent to this device; transfers are seen as a continuous stream of data. Chained buffers can be of arbitrary byte alignment and length. The length of a transfer appears to the far end DR-device to be the total of all the byte counts in the chain, and since chains in the DR780 can be of unlimited length, the device sees the byte count as potentially infinite.

Far End DR-Device Initiated Transfer

The DR780 provides the capability for the far end DR-device to initiate data transfer to the VAX memory, that is, random access mode. This mode is used when two DR780s are connected to form a processor-to-processor link. Random access consists of data transfers to or from the VAX memory without notification of the VAX processor. Random access can be discontinued by either specifying a command packet with random access disabled or by an abort from either the controlling process or the far end DR-device.

Power Failure

If power fails on the DR780, but not on the system, the DR780 driver aborts the active data transfer and returns the status code SS$_POWERFAIL in the I/O status block. If a system power failure occurs, the DR780 driver completes the active data transfer when power is recovered and returns the status code SS$_POWERFAIL.

Interrupts

The DR780 can interrupt the DR780 driver for any of the following reasons:

- An abort has occurred. The QIO is completed.
- A DR780 power down or power up sequence has occurred.
- An unsolicited control message has been sent to the DR780. If the command packet's interrupt control field is properly set up, a packet AST interrupt occurs. The interrupt occurs after the command packet obtained from FREEQ is placed on TERMQ. (Discussion of the various queues and their functions is found in the PROGRAMMING INTERFACE section of this chapter.)

- The DR780 enters the Halt state. The QIO is completed.
- A command packet that specifies an unconditional interrupt has been placed onto TERMQ. This results in a packet AST.
- A command packet with the "interrupt when TERMQ empty" bit set was placed on an empty TERMQ. This results in a packet AST.

PROGRAMMING INTERFACE

The DR780 is supported by a device driver and a high-level language procedure library of support routines.

After the driver initializes, the DR780 application programs communicate directly by inserting command packets onto queues. (Command packets contain commands for the DR780, such as the direction of transfer and/or messages to be sent to the user device.) This direct link between the application program and the DR780 provides faster communication by avoiding the necessity of going through the I/O driver.

Two interfaces are provided for accessing the DR780: a QIO driver and a set of support routines. The QIO driver requires that the application program build command packets and insert them into the DR780 queues. The set of support routines, on the other hand, provide procedures for building and manipulating command packets and, in addition, perform housekeeping functions, such as maintaining command memory.

The DR780 program interface contains three queues: input, termination, and free. These three queues control the flow of command packets to and from the DR780. The application program builds a command packet and inserts it onto the input queue. The DR780 removes the packet, executes the specified command and then inserts the command packet into the termination queue. Unsolicited input, such as a control message from the user device, is placed in packets removed from the free queue and inserted onto the termination queue for later processing. Figure 19-3 shows the flow of command packets as they are moved to the three queues.

The support routine interface has been designed to be called from high-level languages (such as FORTRAN) where the data manipulation which would be required by the QIO interface might be awkward. Note, however, that the support routine user must be equally as knowledgeable as the user of the QIO interface in terms of knowledge of the DR780 and the meaning of the fields in the command packets.

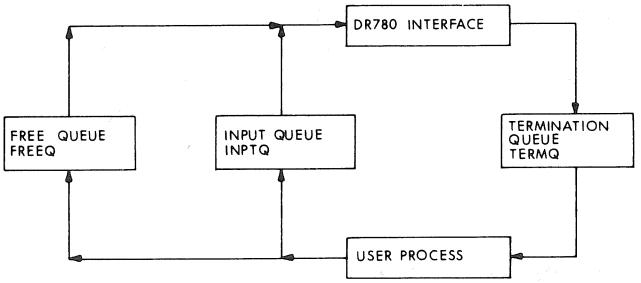


Figure 19-3 Command Packet Flow in the Operation of the DR780

Application Program Interface

The application program interfaces with the DR780 through two memory areas: the command block and the buffer block. The command block contains the headers for the three queues that provide the communication path between the DR780 and the application program, as well as space for the command packets to be built. The buffer block defines the area of memory that is accessible to the DR780 for the transfer of data between the user device and the DR780 itself.

Initiating Command Sequences

If a command packet is inserted onto an empty INPTQ, the application program must notify the DR780 of this event. This is accomplished by setting bit 0, the GO bit, in a DR780 register. The IO$_STARTDATA QIO returns the GO bit's address to the application program. After notification by the GO bit that there are command packets on its INPTQ, the DR780 continues to process the packets until INPTQ is empty.

The GO bit can be safely set at any time. While processing command packets, the DR780 ignores the GO bit. If the GO bit is set when the DR780 is idle, the DR780 will attempt to remove a command packet from INPTQ. If INPTQ is empty at this time, the DR780 clears the GO bit and returns to the idle state.

Device-Initiated Command Sequence

If the DR-device that interfaces to the far end of the DDI is capable of transmitting unsolicited control messages, they can be transmitted to the local DR780. These messages are not synchronized to the application program command flow. Therefore, the DR780 uses a third queue, FREEQ, to handle unsolicited messages. Normally, the application program inserts a number of empty command packets into FREEQ to allow the external device to transmit control messages.

If a control message is received from the far end DR-device, the DR780 removes an empty command packet from the head of FREEQ, fills the device message field of this packet with the control message and, when the transmission is completed, inserts the packet onto the tail of TERMQ. (The device message field in this command packet must be large enough for the entire message or a length error will occur.) The application program then removes the packet from TERMQ. If the command packet is from FREEQ, the XF$M_PKT_FREQPK bit in the DR780 Status Longword is set.

Command Packets

To provide for direct communication between the controlling process and the DR780, the DR780 fetches commands from user-constructed command packets located in main memory. Command packets contain commands for the DR780, such as the direction of transfer, or messages to be sent to the far end DR-device. The DR780 is simply the conveyer of these messages; it does not examine or add to their content. The controlling process builds command packets, and manipulates the three queues, using the four VAX self-relative queue instructions. Figure 19-4 shows the contents of a DR780 command packet.

PROGRAMMING HINTS

This section contains information on programming considerations relevant to users of the DR780. A complete discussion of the programming aspects of the DR780 can be found in the **VAX/VMS I/O Users Guide.**

Command Packet Prefetch

The DR780 can prefetch command packets from INPTQ. While executing the command specified in one packet, the DR780 can prefetch the next packet, decode it, and be ready to execute the specified command at the first opportunity. When the command is executed depends on which command is specified. For example, if two read device or write device command packets are on INPTQ, the DR780 fetches the first packet, decodes the command, verifies that the transfer is legal, and starts the data transfer. While the transfer is taking place, the DR780 prefetches the next read device or write device command packet, decodes it, and verifies the transfer legality. The second transfer begins as soon as the first transfer is completed.

On the other hand, if the two command packets on INPTQ are read device (or write device) and write device control message, in that order, the DR780 prefetches the second packet and immediately executes the command, because control messages can be overlapped with data transfers. The DR780 then prefetches the next command

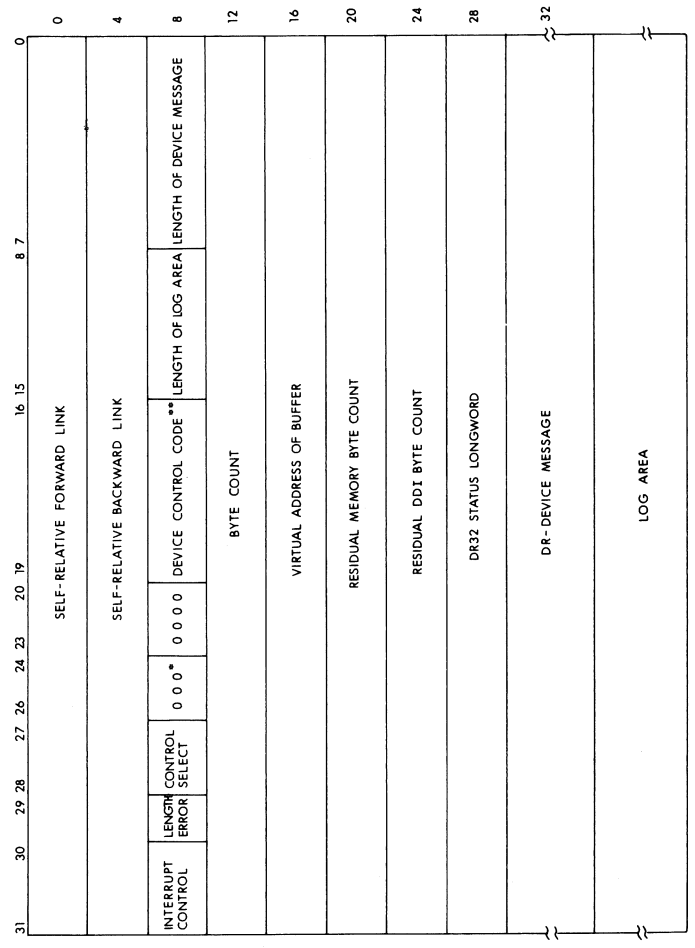


Figure 19-4 DR780 Command Packet

packet. In an extreme case, the DR780 can send several control messages over the control portion of the DDI while a single data transfer takes place on the data portion of the DDI.

The prefetch capability and the overlapping of control and data transfers can cause unexpected results when programming the DR780. For instance, if the application program calls for a data transfer to the far end DR-device followed by notification of the far end DR-device that data are present, the program cannot simply insert a write device command packet and then a write control message command packet onto INPTQ—the control message may very likely arrive before the data transfer completes.

A better way to synchronize the data transfer with notification of data arrival is to request an interrupt in the interrupt control field of the data transfer command packet. Then, when the data transfer command packet is removed from TERMQ, the application program can insert a write control message command packet into INPTQ to notify the far end DR-device that the data transfer has completed.

Another consequence of command packet prefetching occurs when, for example, two write device command packets are inserted into INPTQ while the first data transfer takes place, the second command packet is prefetched and decoded. If an unusual event occurs and the application program must send an immediate control message to the far end DR-device, the application program may insert a write device control message packet onto INPTQ. However, this packet is not sent immediately because the second write device command packet has already been prefetched; the control message is sent after the second data transfer starts.

If the application program needs to send a control message with minimum delay, use one of the following techniques:

- Insert only one data transfer function onto INPTQ at a time. If this is done, a second transfer function will not be prefetched and a control message can be sent at any time.
- Use smaller buffers or a faster data rate to reduce the time necessary to complete a given command packet.
- Issue a $CANCEL system service call followed by another IO$_STARTDATA QIO. This technique will have determinant effect on data throughput.

Action Routines

Action routines provide a useful DR780 programming technique. They can be used in application programs written in either assembly language or a high level language. When a command packet is built, the

address of a routine to be executed when the packet is removed from TERMQ is appended to the end of the packet. Then, rather than having to determine what action to perform for a particular packet when it is removed from TERMQ, the specified action routine is called.

Error Checking

Bits <23:0> in the second longword of the I/O status block correspond to the same bits in the DR780 Status Longword (DSL). Although the I/O status block is written only after the QIO function completes, the DSL is stored in every command packet. However, because there is no command packet in which to store a DSL for certain error conditions (for example, FREEQ empty) some errors are reported only in the I/O status block. To check for an error under these conditions, the user should examine the DSL in each packet for success or failure only. Then, if a failure occurs, the specific error can be determined from the I/O status block. The I/O status block should also be checked to verify that the QIO has not completed prior to a wait for the insertion of additional command packets into TERMQ. In this way, the application program can detect asynchronous errors for which there is no command packet available.

Queue Retry Macro

When an interlocked queue instruction is included in the application program, the code should perform a retry if the queue is locked. However, the code should not execute an indefinite number of retries. A programming error can cause the queue header to become corrupted. Consequently, all retry loops should contain a maximum retry count.

Diagnostic Functions

The diagnostic functions listed in Table 19-1 can be used to test the DR780 without the presence of a far end DR-device. For the DR780, the user should perform the following test sequence:

1. Insert a set self-test command packet into INPTQ.
2. Insert a diagnostic write internal command packet that specifies a 128-byte buffer into INPTQ. This packet copies 128 bytes from memory into the DR780 internal data silo.
3. Insert a diagnostic read DDI command packet into INPTQ. This packet transmits the 128 bytes of data from the silo over the DDI and returns it to the silo.
4. Insert a diagnostic read internal command packet that specifies another 128-byte buffer in memory into INPTQ. This packet copies 128 bytes of data from the silo into memory.

5. Compare the two memory buffers for equality. Note that on the DR780, the diagnostic read internal function destroys the first four bytes in the silo before storing the data in memory. Therefore, compare only the last 124 bytes of the two buffers.
6. Insert a clear self-test command packet into INPTQ.

Table 19-1 Device Control Code Descriptions

Function	Meaning
Read Device	This function specifies a data transfer from the far end DR-device to the DR780. The control select field describes the information to be transferred prior to the initiation of the data transfer.
Read Device Chained	This function specifies a data transfer from the far end DR-device to the DR780. The DR780 data chains to the buffer specified in the next command packet in INPTQ. A command packet that specifies Read Device Chained must be followed by a command packet that specifies either Read Device Chained or Read Device. All other device control codes cause an abort. If Read Device Chained is specified, the chain continues. However, if Read Device is specified, that command packet is the last packet in the chain.
Write Device and Write Device Chained	These functions specify data transfers from the DR780 to the far end DR-device. Otherwise, they are similar to Read Device and Read Device Chained.
Write Device Control Message	This function specifies the transfer of a control message to the far end DR-device. This message is contained in the device message field of this command packet. The Write Device Control Message function directs the controlling DR780 to ignore the byte count and virtual address fields in this command packet.

Function	Meaning
Set Self-Test	This function directs the DR780 to set an internal self-test flag and to set a disable signal on the DDI. This signal informs the far end DR-device that the DR780 is in self-test mode. In this condition the DR780 can no longer communicate with the far end DR-device.
Clear Self-Test	This function directs the DR780 to clear the internal self-test flag set by the Set Self Test function and return to the normal mode of operation.
No Operation	The NOP function specifically does nothing.
Diagnostic Read Internal	This function directs the DR780 to fill the memory buffer, which is described by the virtual address and byte count specified in the current command packet, with the data that are stored in the DR780 data silo. The buffer is filled in a cyclic manner. For example, on the DR780 every 128-byte section of the buffer receives the silo data. The amount of data stored in the buffer equals the DDI byte count minus the SBI byte count. The DDI byte count is equal to the original byte count. No data transmission takes place on the DDI for this function. On the DR780, the Diagnostic Read Internal function destroys the first four bytes in the silo before storing the data in the buffer.
Diagnostic Write Internal	This function, together with the Diagnostic Read Internal function, is used to test the DR780 read and write capability. The Diagnostic Write Internal function directs the DR780 to store data, which are contained in the memory buffer described by the current command packet, in the DR780 data silo, a FIFO-type buffer. No data transmission takes place on the DDI for this function. The Diagnostic Write Internal function terminates when:

Function	Meaning
	1. The memory buffer is empty (the SBI byte count is 0). 2. An abort has occurred. At the time the function terminates, the amount of data in the silo equals the DDI byte count minus the SBI memory byte count.
Diagnostic Read DDI	This function tests transmissions over the data portion of the DDI. The DR780 must be in the self-test mode. If not, an abort will occur. On the DR780, the Diagnostic Read DDI function transmits the contents of DR780 data silo locations 0-127 over the DDI and returns the data to the same locations. If data transmission is normal, that is, without errors, the residual memory count is equal to the original byte count, the residual DDI count is 0, and the contents of the silo remain unchanged.
Diagnostic Write Control Message	This function tests transmissions over the control portion of the DDI. The DR780 must be in self-test mode. If not, an abort will occur. The Diagnostic Write Control Message function directs the DR780 to remove the command packet on FREEQ and check the length of message field. Then the first byte of the message in the command packet on INPTQ is transmitted and read back on the control portion of the DDI. This byte is then written into the message space of the packet from FREEQ. The updated packet from FREEQ is inserted into TERMQ and is followed by the packet from INPTQ.
Set Random Enable and Clear Random Enable	The Set Random Enable function directs the DR780 to accept read and write commands sent by the far end DR-device. Range checking is performed to verify that all addresses specified by the far end DR-device for access are within the buffer block. Far end DR-device initiated transfers to or from VAX memory are conducted

Function	Meaning
	without notification of the VAX processor or the application program. The Clear Random Enable function directs the DR780 to reject far end DR-device initiated transfers. Random access mode must be enabled when the DR780 is used in a processor-to-processor link.
Set HALT	This function places the DR780 in a halt state. The Set HALT function always generates a packet interrupt regardless of the value in the interrupt control field. If an AST routine was requested on completion of the QIO function, the routine is called after the command packet containing the Set HALT function has been processed by the DR780.

The NOP Command Packet

It is often useful to insert a NOP command packet into INPTQ to test the state of the DDI disable bit in the DSL. By checking this bit before initiating a data transfer, an application program can determine if the far end DR-device is ready to accept data.

Interrupt Control Field

The interrupt control field determines the conditions under which an interrupt is generated: unconditionally, if TERMQ was empty, or never. There are three general applications of this field:

1. If a program performs five data transfers, for example, and requires notification of completion only after all five have completed, the first four command packets should specify no interrupt and the fifth command packet should specify an unconditional interrupt.
2. If a program performs a continuous series of data transfers, each command packet can specify interrupt only if TERMQ was empty. Then, every time an event flag or AST notifies the program that a command packet was inserted into TERMQ, the program removes and processes all packets on TERMQ until it is empty.
3. Command packets that specify no interrupt should never be mixed with command packets that specify interrupt if TERMQ was

empty. If this were done, a command packet that specifies no interrupt could be inserted onto TERMQ, followed by a command packet that specifies interrupt if TERMQ was empty; the latter packet will not interrupt and the program is never notified that command packets were inserted onto TERMQ.

PHYSICAL CHARACTERSITICS

The DR780 has the same physical characteristics as both the UNIBUS Adapter (UBA) and the MASSBUS Adapter (MBA). It requires one Option Panel Space and can therefore mount in either the central processor cabinet or the CPU expander cabinet. All necessary cabling and a power supply are included. The cable used with the DR780 is a shielded version of the standard round MASSBUS cable for use when connections from cabinet to cabinet are made. This cable is 25 feet in length. Note that while the DR780 uses MASSBUS cable, the DDI electrical characteristics are quite different from those of the MASSBUS. All user devices must be mounted in either a free-standing cabinet or in an expansion box. When the user device is to be mounted in an expansion box, a flat MASSBUS cable may be used. This cable is available in various lengths; a 10 foot flat MASSBUS cable is included with the DR780 option.

CONFIGURING THE DR780 IN VAX-11/780 SYSTEMS

The DR780 can transfer data at a maximum rate of 6.67 MB/second. This speed is greater than the maximum rate a single memory controller can handle. It is therefore imperative that the user carefully analyze the application to be performed and determine the maximum rate at which the DR780 will be operating. In many instances, a second memory controller will be required so that the system can take advantage of interleaving memory to achieve the transfer rate desired.

Table 19-2 will enable the user to do a first pass configuration/throughput analysis of a VAX-11/780 system incorporating a DR780. The assumptions used to develop this analysis are as follows:

- The user must identify which SBI devices (MBAs and UBAs) are transferring **concurrently** with the DR780. Note that this is very different than the number of physical adapters attached to the system.
- The 0 CPU case serves to document the upper limit of DR780 performance. Knowledgeable users can attain the 0 CPU performance with proper programming. For this case, the transfer rates shown require a user-supplied, external clock.
- Generally, users are interested only in the performance of the DR780 with continuous (as opposed to stall) transfers on the DDI. Therefore, only the continuous performance numbers are given.

- Use of small buffers (less than 16 Kb) in command chaining and data chaining will result in some loss of performance.
- UBAs and MBAs have roughly the same effect on DR780 performance, with MBAs with RM03s being the worst case. The configuration rules were obtained by using MBA with RM03 simulations realizing that this is a worst case representation of a mixture of UBAs and MBAs.
- The MA780 multiport memory option is not included in the system configurations. The interaction of these devices is very application dependent and general rules cannot be applied.
- While it is architecturally possible to connect more than one DR780 to a system, properly configuring such a system is very complex since the system's behavior depends highly upon the user application. Consult with your Sales Representative if more than one DR780 is needed.
- The memory read and memory write performance is different enough to specify them separately.

Table 19-2 DR780 Performance

Number of Devices (transferring concurrently with the DR780)			**DR780 Performance (MB/second)**	
Memory CPU's[1]	Memory Controllers	UBAs + MBAs + 1 (DR780)[2]	Memory Read	Memory Write
0	1	0	7.0	6.3
1	1	0	4.0	4.0
1	1	1	3.5	3.5
1	1	2	2.75	2.75
0	2	0	7.5	7.5
1	2	0	5.5	7.5
1	2	1	5.0	7.5
1	2	2	4.5	7.0
1	2	3	3.5	4.5
1	2	4	2.5	4.0

1 The CPU was simulated as "worst case" CPU, that is, a CPU doing MOVC5 instructions with fill characters.

2 UBAs + MBAs is the number of adapters transferring **concurrently** with the DR780. The +1 represents the DR780 in the simulated configuration.

Select the appropriate row which corresponds to the desired configuration and read the applicable DR780 maximum transfer rate. For transfers **into** memory use the "memory write" column, for transfers **from** memory use "memory read." For example, if the desired system consists of one CPU, two memory controllers and two I/O adapters (either MBAs or UBAs or one of each) which are transferring **concurrently** with the DR780, then the maximum rates for the DR780 are:

- from memory to the DR780 = 3.5 MB/sec
- from the DR780 into memory = 4.5 MB/sec

MA780 MULTIPORT MEMORY

Features	Benefits
11 MB/second maximum total throughput on memory transfers	Extremely fast interprocessor communications
Two MA780s (2 MB each) per VAX processor	Expands physical address space up to a system total of 12 MB
Up to four VAX-11/780 processors can share each MA780	Allows parallel or sequential processing functions for exceptionally fast application processing
Full VAX/VMS operating system support	Access to shared memory is transparent to the user
Invalidation logic enables caching of shared memory data	Eliminates problem of "stale data"
Memory interlock capability	Prevents shared memory from being read by one processor while being updated by another
Individual memory port on/off line switching	Enables selective CPU shutdown permitting high over-all system uptime

INTRODUCTION

The MA780 multiport memory enables up to four VAX-11/780 processors to share a bank of from 256 KB to 2 MB of memory. This capability allows VAX-11/780 users to develop multicomputer configurations for very high throughput or enhanced availability applications. The maximum throughput rate for the MA780 is 11 MB per second.

The VAX/VMS operating system fully supports MA780 configured systems. An application built around multiple cooperating processes can be reconfigured to run on a multi-CPU system with no program modification. Processes in shared memory can be moved from one processor to another with complete transparency to the programs involved. Specifically, VAX/VMS provides support for MA780 configurations in the areas of interprocessor communications (sharing of data regions, VMS mailboxes, common event flags) and the sharing of code among processors.

VAX/VMS itself does not use the shared memory in its dynamic page pool of available memory and no part of the operating system resides in the shared memory. Each CPU in the multiport system operates independently using its own copy of VMS stored in its local memory.

Figure 19-5 illustrates two VAX-11/780 systems incorporating the MA780.

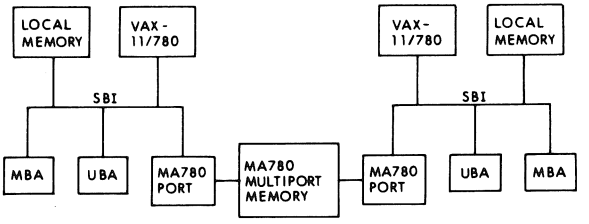


Figure 19-5 VAX-11/780—MA780 Configuration

CAPACITY AND EXPANDABILITY

The MA780 comes with 256 KB of ECC MOS memory as the initial configuration. Additional amounts of memory can be added in 256 KB increments up to a maximum of 2 MB. A VAX-11/780 system will support two MA780 subsystems, thus increasing the total addressable physical memory to 12 MB per system.

Each MA780 option comes equipped with two interface ports to allow communication from one VAX-11/780 processor to a second. Additional interface port options can be added permitting access to a third and fourth CPU. These port options are mounted within the MA780 subsystem.

It is recommended that systems with three or four MA780 I/O ports also incorporate a selective cache invalidate option (see DATA INTEGRITY section).

Physical Layout

Physically, the MA780 is composed of standard VAX-11/780 power supplies, cables and memory array boards, all housed in a standard VAX-11/780 cabinet. Adequate space is allocated within the cabinet to accommodate a second MA780.

Figure 19-6 illustrates the MA780 cabinet layout.

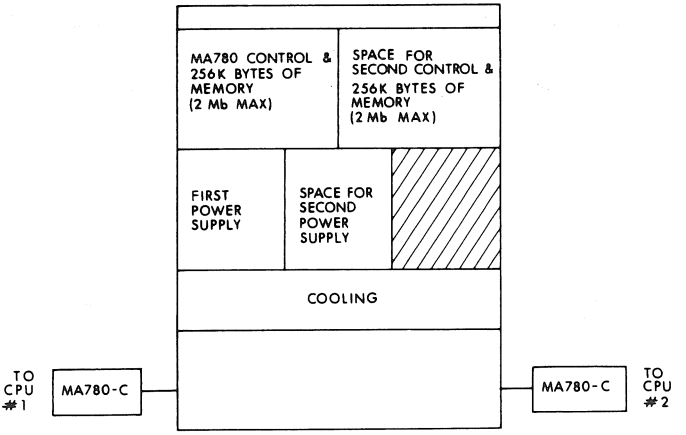


Figure 19-6 MA780 Cabinet Layout

The MA780 is compatible with all VAX-11/780 system and therefore can be readily installed on existing systems.

THROUGHPUT

The MA780 has a maximum throughput rate of 11 MB per second. This rate applies to 64-bit quadword (eight-byte) data transfers. Smaller sized transfers will result in lower throughput. Also, throughput to shared memory as seen by a CPU is very much a function of other factors within that processor, including cache hits, I/O and SBI traffic.

Port Servicing

The high throughput rate is in part due to the fact that each MA780 port has a buffer for commands and data. Further, each port is serviced on a demand basis, that is, first in-first serviced. No time is wasted in polling inactive ports. A servicing algorithm guarantees that no port waits more than three memory cycles to gain access to shared memory.

Parallel and Sequential Processing

The MA780 can enhance overall system throughput by means of one of two configuration types; parallel or sequential (pipeline) processing.

In parallel processing, two or more CPUs divide a task between them. This allows the CPUs to pool their power and complete the job extremely quickly.

Sequential processing can increase total system throughput by allowing instantaneous data exchange between CPUs that are handling sequential parts of an application. Each processor is dedicated to a specific portion of the total application and upon completion of that portion, passes the results on to the next CPU.

Figure 19-7 illustrates parallel and sequential processing configurations.

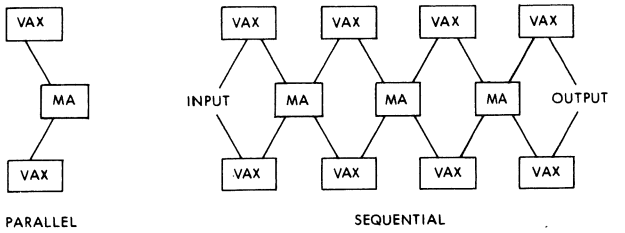


Figure 19-7 Parallel and Sequential Processing Configurations

DATA INTEGRITY

Since the MA780 utilizes the same memory array cards as those of the VAX-11/780 main memory subsystem, the shared memory also has the built-in error correcting code (ECC) to correct all single-bit errors and detect all double-bit errors on memory reads and writes. Parity error checking is also present on all MA780 internal buses.

Interlock Capability

A problem associated with shared memory is that while one processor is reading a memory location, a second processor can simultaneously be trying to update that location. The MA780 eliminates this problem by locking out the second processor until the first has completed its transaction.

Cache Invalidation Logic

The MA780 has been designed to virtually eliminate the problem of incorrect or stale data in the CPU caches. After a write to a shared memory location, the MA780 sends an invalidate signal to all CPUs. This signal transmits an address to each CPU which then checks to see if its cache contains that address. If so, it marks the contents as invalid and on subsequent reads by the CPU, this invalidation forces the CPU to seek the new information from shared memory. The VAX-11/780 cache therefore retains its transparency to software.

Selective Cache Invalidation

For multicomputer systems where there is considerable traffic on a system's SBI, a selective cache invalidation option is available. It is called an invalidation map and is useful primarily in systems with three or four CPUs. It causes the MA780 to send invalidate signals only to those CPUs that have accessed a given location in shared memory. Its operation is quite straightforward: the invalidation map has one entry for each 64-bit quadword in shared memory. This entry contains four bits, one for each CPU. Whenever a CPU reads a 64-bit quadword from shared memory, the MA780 controller sets the bit in the correct invalidation map entry. The next time a CPU writes into that 64-bit quadword, the MA780 controller notifies only those CPUs whose bits have been set in the invalidation map. Once that has been done, all the bits are cleared except that of the CPU performing the write operation. In this way, the overall traffic on all SBIs and cache activity is kept to a minimum while data integrity is still maintained.

Battery Backup

Battery backup support is optionally available for the shared memory in system configurations sensitive to power loss and fluctuation. The backup unit resides within the MA780 cabinet and is capable of supporting 2 MB of memory for a minimum of ten minutes. Smaller amounts of memory are supported for longer periods of time. This level of support is sufficient to protect the memory contents from the vast majority of power interruptions.

FAILSOFT CAPABILITY

Failsoft capability is the capability to maintain a high degree of system uptime by means of a design which incorporates extensive diagnostics to detect faults and allows system functioning to continue even though certain elements of the system may be down for service.

The MA780 supports this failsoft capability through several features. System diagnostics can access the MA780 through any one of its memory access ports and any connected VAX-11/780 system is

allowed to read the multiport status register. Each port can be separately switched off-line and the appropriate processor powered off. This permits on-line overall system service even though the switched-off port may be connected to a system element not presently operational.

The MA780 itself generates interrupts to notify ports of power failures and error conditions as soon as they are detected. The VMS operating system logs these detected errors and consequently minimizes the MA780's mean time to repair.

Multiprocessor systems maintain their high degree of in-service time through one other important feature. Although each processor is working on the same or related tasks, each processor maintains its own VMS operating system. Thus an operating system failure in one system does not bring down the entire multi-CPU system. The user application, however, must have been programmed to deal with this occurrence.

USING SHARED MEMORY

Before an application using multiport memory can execute under VAX/VMS, the system manager must activate the VAX/VMS operating system in processors connected to the multiport memory unit and initialize that memory. The **VAX/VMS System Manager's Guide** explains the system management responsibilities associated with a multiport memory unit.

Preparing Multiport Memory for Use

First, the system manager activates the VAX/VMS operating system in a VAX-11/780 and intializes the multiport memory unit. These actions cause the following to occur:

- The uninitialized shared memory is connected to the VAX/VMS system running in the processor.
- A name is defined that all processes running in all processors can use to refer to the shared memory.
- Limits are set for the following resources in this multiport memory unit:
  - Common event flag clusters: the total number that can be created, by processes running on this processor.
  - Mailboxes: the total number that can be created, and the number that can be created by processes running on this processor.
  - Global sections: the total number that can be created, and the number that can be created by processes running on this processor.

The system manager then activates the VAX/VMS operating system in other processors connected to the multiport memory unit and connects the initialized shared memory to the VAX/VMS system running in each of these processors and sets limits for the number of common event flag clusters, global sections, and mailboxes that processes on each processor can create in the multiport memory.

The system manager can also install global sections in shared memory just as they are installed in local memory. The INSTALL utility can be used to create shared memory global sections for known files. Once the global sections are installed, a process running in any processor connected to the MA780 can map to the section, if the process has the appropriate privilege. The process can gain access to the global section either by using a logical name defined by the system manager or by using the section name specified when the global section was created. In the latter case, the section name must be unique on this processor.

To use facilities in memory shared by multiple processors, all of the user privileges required to use the equivalent facility in local memory must be present. For example, to create a permanent global section, the user must have the PRMGBL privilege, and to create a temporary or permanent mailbox, the user must have the TMPMBX or PRMMBX privilege.

Assigning Logical Names and Logical Name Translation

The user can define a logical name for a shared memory facility with the DEFINE or ASSIGN command or the Create Logical Namme ($CRELOG) system service. Application programs can then refer to the facility using the logical name; for example, a process can invoke the Create Mailbox and Assign Channel ($CREMBX) system service specifying the logical name for an existing mailbox to which a channel is to be assigned.

When translating a logical name for a shared memory facility, the VAX/VMS operating system uses a slightly different approach from that used for other logical names. The purpose of this approach is to allow programmers to specify either the complete name (memory name and facility name) or a logical name that the system will translate to the complete name. If logical names are defined properly, a program that uses a given facility in local memory can be run without change to use the facility in shared memory.

Whenever VAX/VMS encounters the name of a common event flag cluster, mailbox, or global section, it performs the following special logical name translation sequence:

1. Inserts one of the following prefixes to the name (or to the part of the name before the colon if a colon is present):

   CEF$ for common event flag clusters
   MBX$ for mailboxes
   LIB$ for global sections
2. Subjects the resultant string to logical name translation. If translation does not succeed (that is, the original name did not use a logical name), passes the original name string to the system service. If translation does succeed, goes to step 3.
3. Appends the part of the original string after the colon (if any) to the translated name.
4. Repeats steps 1 to 3 (up to nine more times, if necessary) until logical name translation fails. When translation fails, passes the string to the system service.

How VAX/VMS Finds Facilities in Shared Memory

After the VAX/VMS system performs the logical name translation, the final equivalence name must be the name of a facility in either the processor's local memory or in shared memory. If the equivalence name specifies the name of a shared memory (that is, the name is in the format name:facility-name), VAX/VMS searches for the facility in the appropriate data base of the specified memory.

If the equivalence name specifies a common event flag cluster or mailbox and does not specify a memory name, VAX/VMS searches through the common event flag cluster data base or the mailbox data base until it locates the specified cluster or mailbox. Absence of a memory name as part of a common event flag cluster name or mailbox name indicates that the facility is located in local memory.

If the equivalence name specifies a global section and does not specify a memory name, VAX/VMS looks for the section as follows:

1. First, it searches the global section tables for sections in the processor's local memory.
2. Then, it searches the global section tables for each initialized shared memory connected to the processor in the order in which they were connected and recognized by the processor.

The result of searching in this order is that global sections in the processor's local memory take precedence over those in shared memories. Thus, absence of a memory name as part of a global section name is not used as an indication of where the global section is located.

Using Common Event Flags in Shared Memory

Under VAX/VMS, any process can associate with up to two common event flag clusters (event flag numbers 64 through 95 and 96 through 127). These clusters can be located in shared memory or in local memory. To create and associate with a common event flag cluster in shared memory and manipulate flags in the cluster, use the same steps as would be used to associate with a common event flag cluster in local memory:

1. Issue the Associate Common Event Flag Cluster ($ASCEFC) system service to create the cluster or to associate with an existing cluster.
2. Issue any of the services that set, clear, and wait for designated event flags, as appropriate.

As with local memory clusters, the first process among cooperating processes to issue the Associate Common Event Flag cluster ($ASCEFC) system service causes the cluster to be created. Any other process calling this service and specifying the same cluster associates with that cluster. VAX/VMS implicitly qualifies cluster names with the group number of the creator's UIC; therefore, other cooperating processes must belong to the same group.

All of the event flag system services, with the exception of Associate Common Event Flag Cluster and Disassociate Common Event Flag Cluster, function identically regardless of whether they are used with local or shared memory clusters. The only difference with the Associate and Disassociate system services is that to specify a cluster in shared memory, the user must provide the memory name as well as the cluster name. That is, after VAX/VMMS performs logical name translation of the name argument, the cluster name must have the following format:

```
memory-name:cluster-name
```

Using Mailboxes in Shared Memory

The first process on each processor to refer to a shared memory mailbox must use the Create Mailbox and Assign Channel ($CREMBX) system service to create the mailbox and assign a channel to it. Any $CREMBX system service call referring to a shared memory mailbox must specify a mailbox name that has or translates to the following format:

```
memory-name:mailbox-name
```

When the mailbox is created, the $CREMBX system service also creates the mailbox-name portion of the name string as a logical name with an equivalence name in the format MBn. For example, if the

complete name string is SHMEM:MAILBOX, the system service will create MAILBOX as a logical name with an equivalence name of, for example, MBB005.

The Assign I/O Channel ($ASSIGN) and Deassign I/O Channel ($DASSGN) system services require that you specify only the mailbox-name portion of a shared memory mailbox name string. Likewise, any high-level language program statements that open, close, read from, or write to a shared memory mailbox must specify only the mailbox-name portion.

A mailbox in shared memory cannot be used as a process termination mailbox.

Using Global Sections in Shared Memory

Under VAX/VMS, processes can map global sections located in local memory or in shared memory. A global section in shared memory can be mapped to an image file or a data file, just like a global section in local memory. To create a global section in shared memory, the same steps are perfomed as to create a global section in local memory:

1. Using VAX-11 RMS, open the file to be mapped.
2. Issue the Create and Map Section ($CRMPSC) system service.

The file to be mapped must reside on a disk device attached to the local processor. Once the section is created, however, processes on all processors attached to the shared memory can map the section.

To map an existing global section in shared memory, the user issues a Map Global Section ($MGBLSC) system service specifying the name of the section. Once the section is mapped, processes gain access to shared memory global sections in the same manner as they do to local memory sections. VAX/VMS thus makes use of the shared memory unit transparent to the process.

VAX/VMS treats the pages of a global section in shared memory differently from pages in local memory. When a process creates a shared-memory global section, VAX/VMS brings all of the pages of the mapped image or data file into memory. Any process mapped to that global section can gain access to those pages without incurring a page fault because the pages are already in physical memory. Unlike process pages in local memory, global section pages in shared memory are not included in the working sets of the processes that map the section.

Because no paging occurs, VAX/VMS never writes the contents of shared memory global section pages back to their disk file. For read/write global sections in which the user wants to maintain an

updated file while the application executes, an Update Section File Disk ($UPDSEC) system service must be issued. The process issuing the update request must execute on the same processor as the process that created the global section. The disk file can be updated periodically during execution of the application as a checkpoint precaution. The disk file is automatically updated when the section is deleted.

Each process that has mapped a global section in shared memory can unmap the section in either of the following ways:

- Issue a Delete Virtual Address Space ($DELTVA) system service to delete the process's virtual address space that maps the section.
- Terminate the current image, thereby causing VAX/VMS to unmap the process from the section automatically.

Deleting a global section in shared memory requires an explicit deletion request, because all global sections in shared memory must be permanent sections. The deletion request can be either a Delete Global Section ($DGBLSC) system service issued by the application, or a deletion request issued by the system manager. In either case, VAX/VMS does not perform the actual deletion until all processes that have mapped the section unmap it.

VAX-11/782 ATTACHED PROCESSOR SYSTEM

Features	Benefits
Full VAX/VMS operating system support	Access to shared memory is transparent to the user
Improved Dynamic load leveling	Optimal use of system resources
Users can transfer applications from a single to an attached processor configuration	No need for additional development or debugging efforts
Both processors share the same operating system code and data structures	A system manager only needs to maintain one copy of the VAX/VMS operating system

INTRODUCTION

The VAX-11/782 Attached Processor System is classified as a tightly coupled, asymmetric multiprocessor system. It is based on the MA780 multiport memory subsystem discussed in the previous section of this chapter.

In a tightly coupled system, the CPUs execute the same copy of the operating system code and share the same data structures. Asymmetric CPUs cannot execute the entire operating system code at the same time. In this system, all kernel mode and interrupt code is executed by the primary processor. All I/O operations are also conducted by the primary processor. The primary processor schedules all work for the attached processor before scheduling itself. Figure 19-8 illustrates system software relationships.

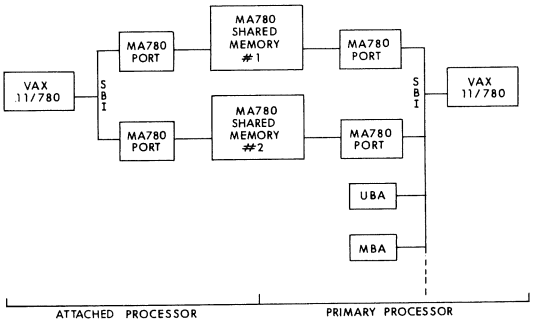


Figure 19-8 System Block Diagram (Software)

Applications

Both commercial and technical users will find the high performance of the VAX-11/782 advantageous. Experimental data reduction, structural analysis, electronic and mechanical design with interactive graphics, high-energy physics and quantum mechanics research are all appropriate technical applications, especially where a multiuser community needs access to large-scale data. The VAX-11/782 is also an excellent choice for econometric modeling, forecasts and business simulation, and statistical processing.

Large compute-intensive programs would typically execute almost entirely in the attached processor, leaving the primary processor free for I/O. This happens because the VAX/VMS scheduler dynamically allocates compute-intensive functions to the attached processor and functions that require I/O transfers to the primary processor.

HARDWARE

The VAX-11/782 attached processor system is available in three complete packaged systems. Each VAX-11/782 system includes two CPUs each with local memory (for diagnostics), an LA120 console terminal,

the VAX/VMS operating system, different amounts of MA780 shared memory, a choice of mass storage subsystems, and an 8-line communication multiplexer. The attached processor does not support I/O peripherals. All I/O devices and peripherals are connected to the primary processor. Figure 19-9 illustrates a VAX-11/782 attached processor system.

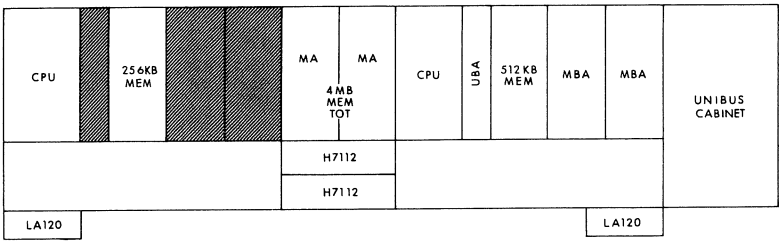


Figure 19-9 VAX-11/782 Attached Processor System

The VAX-11/782 attached processor is also available as an upgrade option to a single processor VAX-11/780 system. Components of the attached processor system upgrade include a CPU, an LA120 console terminal, 1 MB of MA780 shared memory (maximum of 8 MB) with battery backup, and the MA780 cache invalidate option. The CPU cabinet also contains local memory for diagnostics.

Capacity and Expandability

Each CPU has local memory for running diagnostics. The attached processor upgrade is delivered with 1 MB of MA780 shared memory expandable up to 8 MB. Packaged systems are available with different memory sizes and are also expandable to 8 MB.

Cache Invalidation Logic

The MA780 has been designed to virtually eliminate the problem of incorrect or stale data in the CPU caches. After a write to a multiport memory location, the MA780 sends an invalidate signal to each CPU. This signal transmits an address to each CPU, which then checks to see if its cache contains that address. If so, it marks the contents as invalid, and on subsequent reads by the CPU this invalidation forces the CPU to seek the new information from shared memory. The VAX-11/782 cache therefore retains its transparency to software.

Selective Cache Invalidation

For the attached processor systems, the Selective Cache Invalidation option has been included. This allows multicomputer systems that have considerable traffic on a system's SBI to send invalidate signals only to the CPU that has accessed a given location in MA780 memory. This option for shared memory is called an invalidation map and its operation is quite straightforward. It contains one entry for each 64-bit quadword in shared memory. This entry contains four bits, one for each CPU. Whenever a CPU reads a 64-bit quadword from shared memory, the MA780 controller sets the bit in the correct invalidation map entry. The next time a CPU writes into that 64-bit quadword, the MA780 controller notifies only those CPUs whose bits have been set in the invalidation map. Once that has been done, all the bits are cleared except for the bit of the CPU performing the write operation. In this way, the overall traffic on all SBIs and the cache activity are kept to a minimum while data integrity is still maintained.

Battery Backup

Battery backup support is included for the attached processor system configurations. Two backup units reside within the MA780 cabinet and are capable of supporting 4 MB of memory for a minimum of ten minutes. Smaller amounts of memory are supported for longer periods of time. This level of support is sufficient to protect the memory contents from the vast majority of power interruptions.

Throughput

The MA780 has a maximum throughput rate of 11 MB/second. This rate applies to 64-bit quadword (eight-byte) data transfers. Smaller sized transfers will result in lower throughput. Also, throughput to shared memory as seen by a CPU is very much a function of other factors within that processor, including cache hits, I/O and SBI traffic.

Configuring Restrictions for VAX-11/780 Attached Processor

Both VAX-11/780s must be at the same revision level and run the same version of microcode. The Writable Control Store (WCS) option, the DR780, and the 2.2 MB/sec RP07 drive are not supported. If a CPU option, such as the floating point accelerator, is installed in the primary processor then the attached processor must also have the same option. Figure 19-10 illustrates upgrade system cabinet layout.

SOFTWARE

The VAX/VMS multiprocessing-specific code is minimal. The code occupies approximately 8 KB of non-paged pool. All kernel mode and interrupt code is executed by the primary processor. All I/O operations are also conducted by the primary processor.

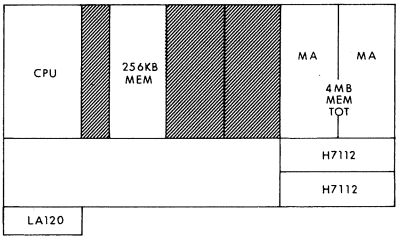


Figure 19-10 VAX-11/782 Upgrade Attached Processor System

Structure

The VAX/VMS operating system fully supports attached processor systems. Most applications built around multiple cooperating processes can be reconfigured to run on a multi-CPU system with no program modification. Processes running on a VAX-11/782 attached processor system will execute on one processor and then the other with complete transparency to the programs involved.

Scheduling

Scheduling is implemented in the same way as on a single VAX processor, i.e., round-robin, with the highest priority job executed first. Two differences exist in the VAX-11/782: first the attached processor will not be pre-empted by a higher priority process, and second it does not execute in kernel mode. The primary processor does all the scheduling for the system. The attached processor is scheduled first and then the primary processor schedules itself.

When there is no process capable of running on the attached processor, the primary will execute a process and force an AST delivery interrupt to occur when the process leaves kernel mode. The AST interrupt is then treated as a rescheduling interrupt and the primary reschedules that process to run on the attached processor.

Initialization

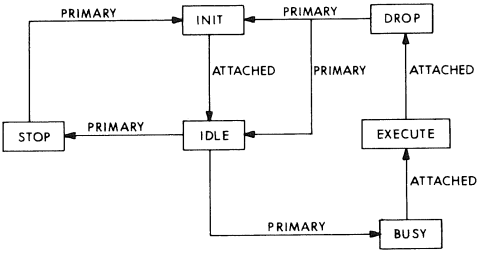
A standard boot procedure is done on the primary processor. After the primary processor is booted, a DCL command is executed to load the multiprocessing specific code into non-paged pool. This command will usually be incorporated into the site-specific startup command file by the system manager. A new system control block (SCB) is initialized for the attached processor and the primary SCB is modified to handle the multiprocessing scheduling code and MA780 interrupt communication.

Next the attached processor is booted. After a minimal amount of initialization it interrupts the primary to request a process to execute. Both processors in the attached system are now running, the primary responsible for scheduling both processors.

Attached Processor States

A state variable is maintained for access by both processors. The transition out of each state is "owned" by a particular processor. Only the owner has the ability to alter the state variable. This prevents various race conditions throughout the multiprocessing code. The primary processor uses the state of the attached processor to determine the availability of the attached processor for work scheduling.

The attached processor is set to the INITIALIZE state by the primary processor when the multi-processing code is loaded. After the attached processor finishes executing the multiprocessing initialization code, it will move into the IDLE state. When the next rescheduling operation is performed by the primary processor, it will initiate scheduling for the attached processor. When the work is assigned to the attached, it is set to the BUSY state. The attached processor checks the state variable and performs a load-process-context and sets the state to EXECUTE. The attached processor will continue to execute its current process until the process either receives its quantum of CPU time or requests some action in kernel mode. When these conditions are met, the attached processor will do a save-process-context and set its state to DROP. The attached processor then notifies the primary processor that it is available for rescheduling. The primary processor, after taking the process back from the attached processor, sets the state to IDLE. The other state available is STOP, which is used to request the attached processor to turn itself off. This state is initiated by the system manager with a DCL command or by the primary processor during operations such as bugcheck. The attached processor can be restarted by another DCL command or by rebooting. Figure 19-11 illustrates the state transitions of the attached processor.



Multiprocessing Interrupt Communication
The MA780 inter-processor interrupt is used extensively by the multiprocessing code.

The primary processor interrupts the attached processor for the following reasons:

-Request for an invalidate of a system space address
-An AST has arrived for the process currently executing in the attached processor
-To request a bugcheck

The attached processor interrupts the primary for the following reasons:
-To request a reschedule event
-To log an error
-To request a bugcheck

Fault Handling
Many VAX/VMS features have been extended for use on multiprocessing systems, including powerfail, bugcheck, machine check, automatic restart, and error logging.
Powerfail has been implemented such that in the event the attached processor loses power, it will transfer the process that it is currently executing to the primary processor and the primary will continue to run without the loss of any data. If the primary powerfails, the attached processor will wait for it to restart.

A bugcheck will stop execution on both processors and can be initiated by either processor. The processors will synchronize during the bugcheck and both can be set to automatically reboot.

The multiprocessing system conducts machine checks and error logging in a manner similar to a single processor system. The error log entries, however, now include the system ID, allowing recognition of the processor that made the entry.


VAX 11/780

CHAPTER 20

VAX-11/780 PRIVILEGED REGISTERS

INTRODUCTION

The processor register space provides access to many types of CPU control and status registers such as the memory management base registers, the PSL, and the multiple stack pointers. These registers are explicitly accessible only by the Move to Processor Register (MTPR) and Move from Processor Register (MFPR) instructions which require kernel mode privileges. This chapter describes those privileged processor registers not found elsewhere in this handbook.

Appendix D contains a description of the ID Bus registers of which the privileged processor registers are a subset. Therefore, those registers containing a processor address are privileged and can be accessed via the MTPR and MFPR instructions only. Chapter 15, Console Subsystem, contains a description of the ID Bus.

SYSTEM IDENTIFICATION REGISTER (SID)

The system identification register is a read-only constant register that specifies the processor type. The entire SID register is included in the error log and the type field may be used by software to distinguish processor types. Figure 20-1 illustrates the system identification register.

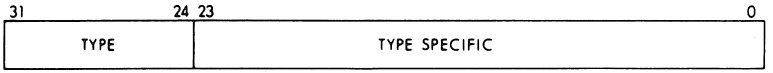
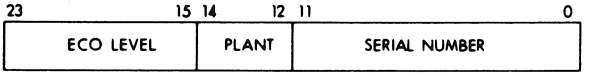


Figure 20-1 System Identification Register

Type	A unique number assigned by engineering to identify a specific processor:
0	Reserved to DIGITAL
1	VAX-11/780
2	VAX-11/750
3	VAX-11/730
4 through 127	Reserved to DIGITAL
128 through 255	Reserved to CSS and customers
Type-specific	Format and content are a function of the value in type. They are intended to include such information as serial number and revision level.

For the VAX-11/780, the type-specific format is:



CONSOLE TERMINAL REGISTERS

The console terminal is accessed through four internal registers. Two are associated with receiving from the terminal and two with writing to the terminal. In each direction there is a control/status register and a data buffer register. Figure 20-2 illustrates the console receive control/status register.

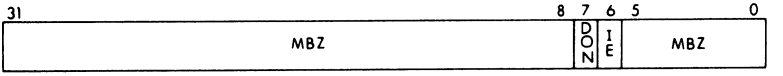


Figure 20-2 Console Receive Control/Status Register (RXCS)

Figure 20-3 illustrates the read-only console receive data buffer register.

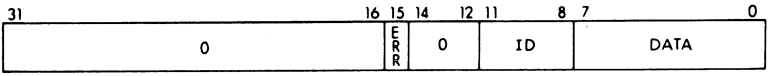


Figure 20-3 Console Receive Data Buffer Register (RXDB)

At bootstrap time, RXCS is initialized to 0. Whenever a datum is received, the read-only bit DONE is set by the console. If IE (interrupt enable) is set by the software, then an interrupt is generated at interrupt priority level (IPL) 20. Similarly, if DONE is already set and the software sets IE, an interrupt is generated (i.e., an interrupt is generated whenever the function (IE AND DONE) changes from 0 to 1). If the received data contained an error such as overrun or loss of connection, then ERR is set in RXDB. The received data appear in DATA. When an MFPR #RXDB,dst is executed, DONE is cleared as is any interrupt request. If ID is 0, then the data are from the console terminal. If ID is not 0, then the entire register is implementation-dependent. In the case of the VAX-11/780, if ID = 1, data are from the floppy disk.

At bootstrap time, TXCS is initialized with just the RDY bit set (ready). Whenever the console transmitter is not busy, it sets the read-only bit RDY. If IE (interrupt enable) is set by the software, then an interrupt is generated at IPL 20. Similarly, if RDY is already set and the software sets IE, an interrupt is generated (i.e., an interrupt is generated when-

ever the function (IE AND RDY) changes from 0 to 1). The software can send a datum by writing it to DATA. When an MTPR src,#TXDB is executed, RDY is cleared as is any interrupt request. If ID is written 0, then the datum is sent to the console terminal. If ID is non-zero, then the entire register is implementation-dependent. In the case of VAX-11/780, if ID = 1, data are sent to the floppy disk. Figure 20-4 illustrates the console transmit control/status register.

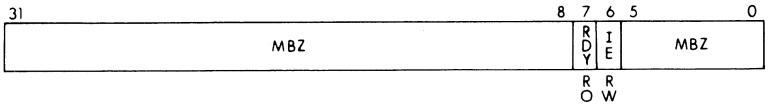


Figure 20-4 Console Transmit Control/Status Register (TXCS)

Figure 20-5 illustrates the read-only console transmit data buffer register.

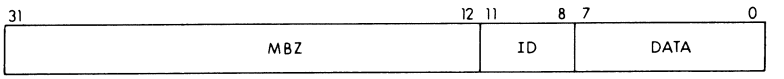


Figure 20-5 Console Transmit Data Buffer Register (TXDB)

CLOCK REGISTERS

The clocks consist of an optional time-of-year clock and a mandatory interval clock. The time-of-year clock is used to measure the duration of power failures and is required by the operating system for unattended restart after a power failure. The interval clock is used for accounting, for time-dependent events, and to maintain the software date and time.

Time-of-Year Clock

The time-of-year clock consists of one longword register. The register forms an unsigned 32-bit binary counter that is driven by a precision clock source with at least .0025% accuracy (approximately 65 seconds per month). The counter has a battery back-up power supply sufficient for at least 100 hours of operation, and the clock does not gain or lose any ticks during transition to or from stand-by power. The battery is recharged automatically. The least significant bit of the counter represents a resolution of 10 milliseconds. Thus, the counter cycles to 0 after approximately 497 days.

If the battery has failed, so that time is not accurate, then the register is cleared upon power up. It then starts counting from 0. Thus, if software initializes this clock to a value corresponding to a large time (e.g., a

month), it can check for loss of time after a power restore by checking the clock value. The time-of-year clock register is illustrated in Figure 20-6.

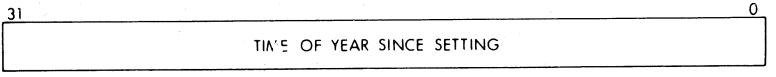


Figure 20-6 Time-of-Year Clock Register (TODR)

If the clock is not installed, then the clock always reads out as 0 and ignores writes.

Interval Clock

The interval clock provides an interrupt at IPL 24 at programmed intervals. The counter is incremented at 1 μs intervals, with at least .01% accuracy (8.64 seconds per day). The clock interface consists of three registers in the privileged register space: the read-only interval count register, the write-only next interval count register, and the interval clock control/status register.

Figure 20-7 illustrates the interval count register.

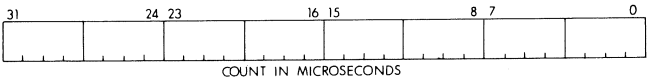


Figure 20-7 Interval Count Register (ICR)

Figure 20-8 illustrates the next interval count register.

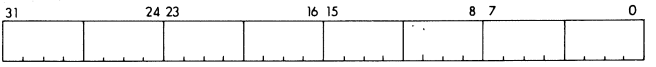


Figure 20-8 Next Interval Count Register (NICR)

Figure 20-9 illustrates the interval clock control/status register.

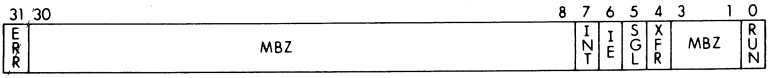


Figure 20-9 Interval Clock Control/Status Register (ICCS)

Interval Count Register
The interval register is a read-only register incremented once every microsecond. It is automatically loaded from NICR upon a carry out from bit 31 (overflow) which also interrupts at IPL 24 if the interrupt is enabled.

Next Interval Count Register
The reload register is a write-only register that holds the value to be loaded into ICR when it overflows. The value is retained when ICR is loaded. NICR is capable of being loaded regardless of the current values of ICR and ICCS.

Interval Clock Control/Status Register (ICCS)
The ICCS register contains control and status information for the interval clock.

Bit: 31 **Name:** ERR
Function: Whenever ICR overflows, if INT is already set, then ERR is set. Thus, ERR indicates a missed clock tick. Attempts to set this bit via MTPR clears ERR.

Bit: 30:8 **Name:** MBZ
Function: Must be zero.

Bit: 7 **Name:** INT
Function: Set by hardware every time ICR overflows. If IE is set, then an interrupt is also generated. An attempt to set this bit via MTPR clears INT, thereby re-enabling the clock tick interrupt (if IE is set).

Bit: 6 **Name:** IE
Function: When set, an interrupt request at IPL 24 is generated every time ICR overflows (INT is set). When clear, no interrupt is requested. Similarly, if INT is already set and the software sets IE, an interrupt is generated (i.e., an interrupt is generated whenever the function (IE AND INT) changes from 0 to 1).

Bit: 5 **Name:** SGL
Function: A write-only bit. If RUN is clear, each time this bit is set, ICR is incremented by one.

Bit: 4 **Name:** XFR
Function: A write-only bit. Each time this bit is set, NICR is transferred to ICR.

Bit: 3:1 **Name:** MBZ
Function: Must be zero.

Bit: 0 **Name:** RUN
Function: When set, ICR increments each microsecond. When clear,

ICR does not increment automatically. At bootstrap time, RUN is cleared.

Thus, to set up the interval clock, load the negative of the desired interval into NICR. Then an MTPR #↑X51,#ICCS will enable interrupts, reload ICR with the NICR interval and set run. Every "interval count" microseconds will cause INT to be set and an interrupt to be requested. The interrupt routing should execute an MTPR #↑XC1,#ICCS to clear the interrupt. If INT has not been cleared (i.e., if the interrupt has not been handled) by the time of the next ICR overflow, the ERR bit will be set.

At bootstrap time, bits <6> and <0> of ICCS are cleared. The rest of ICCS and the contents of NICR and ICR are UNPREDICTABLE.

VAX-11/780 FLOATING POINT ACCELERATOR

The VAX-11/780 processor has an optional accelerator for a subset of the instructions. Two internal registers control the accelerator: ACCS and ACCR.

ACCS is the accelerator control/status register. It indicates whether an accelerator exists, controls whether it is enabled, identifies its type and reports errors and status. At bootstrap time, the type and enable are set; the errors are cleared. Figure 20-10 illustrates the accelerator control/status register.

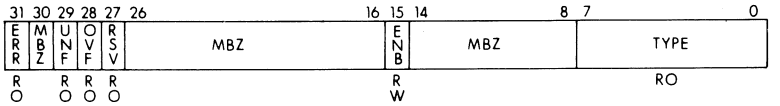


Figure 20-10 Accelerator Control/Status Register (ACCS)

Bit: 31 **Name:** ERR
Function: Read-only bit specifying that at least one of bits RSV, OVF, and UNF is set. Note that bits <31:27> are normally cleared by the main processor microcode before starting the next macro instruction.

Bit: 30 **Name:** MBZ
Function: Must be zero.

Bit: 29 **Name:** UNF
Function: Read-only bit specifying that the last operation had an underflow.

Bit: 28 **Name:** OVF
Function: Read-only bit specifying that the last operation had an overflow.

Bit: 27 **Name:** RSV
Function: Read-only bit specifying that the last operation had a reserved operand.

Bit: 26:16 **Name:** MBZ
Function: Must be zero.

Bit: 15 **Name:** ENB
Function: Read/write field specifying whether the accelerator is enabled. At bootstrap time, this is set if the accelerator is installed and functioning. An attempt to set this is ignored if no accelerator is installed.

Bit: 7:0 **Name:** TYPE
Function: Read-only field specifying the accelerator type as follows:

0 = No accelerator
1 = Floating point accelerator

Numbers in the range 2 through 127 are reserved to DIGITAL. Numbers in the range 128 through 255 are reserved to CSS/customers.

The accelerator maintenance register (ACCR) controls the accelerator's microprogram counter. At bootstrap time its contents are UNPREDICTABLE. Figure 20-11 illustrates the accelerator maintenance register.

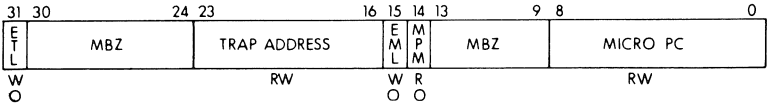


Figure 20-11 Accelerator Maintenance Register (ACCR)

Bit: 31 **Name:** ETL
Function: Enable Trap Address Load. A write-only bit that when set causes <23:16> to be loaded into the accelerator's trap address register. Subsequently, the main processor's microcode can force the accelerator to trap to this location by asserting an internal signal.

Bit: 30:24 **Name:** MBZ
Function: Must Be zero.

Bit: 23:16 **Name:** TRAP
Function: Trap Address. A read/write field used by the main processor to force the accelerator to a specified micro location.

Bit: 15 **Name:** EML
Function: Enable Micro PC Match Load. A write-only bit that when

set causes <8:0> to be loaded into the accelerator's micro PC match register.

Bit: 14 **Name:** MPM
Function: Micro PC Match. A read-only bit that is set whenever the accelerator's micro PC matches the micro PC match register. This is useful primarily as a scope sync signal.

Bit: 13:9 **Name:** MBZ
Function: Must be zero.

Bit: 8:0 **Name:** PC
Function: Next Micro PC on read. This contains the next micro address to be executed.

Match Micro PC on write. If EML is also set, then this updates the micro PC match register.

VAX-11/780 MICRO CONTROL STORE

The VAX-11/780 processor has three registers for microcode control/status. Two are used for writing into any writable control store (WCS) and one is used to control micro breakpoints. Figure 20-12 illustrates the writable control store address register.

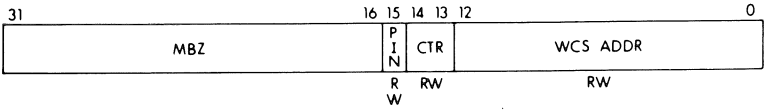


Figure 20-12 User Control Store Address Register (WCSA)

Figure 20-13 illustrates the writable control store data register.

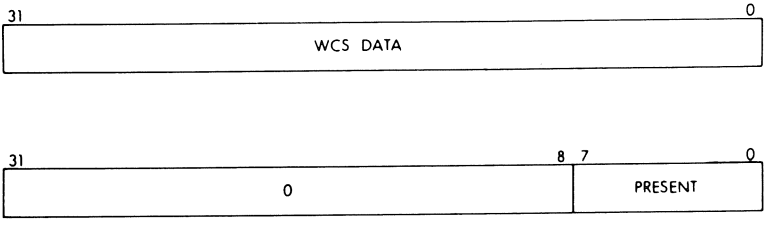


Figure 20-13 Writable Control Store Data Register (WCSD)

Reading WCSD indicates which control store addresses are writable. If WCSD<n> is set, then addresses n*1024 through zn*1024+1023 are writable (i.e., that WCSA<12:10> EQLU n corresponds to writable

control store). n=4 corresponds to WCS that is reserved to DIGITAL for diagnostics and engineering change orders. Other fields correspond to blocks of control that can be used to implement customer- or CSS-specific microcode. Each word of control store contains 96 bits plus 3 parity bits. To write one or more words, initialize WCS ADDR to the address and CTR to 0. Then each MTPR to WCSD will write the next 32 bits and automatically increment CTR. When CTR becomes 3, it is automatically cleared and WCS ADDR is incremented. If PIN is set, then any writes to WCSD are done with inverted parity. An attempt to execute a microword with bad parity results in a machine check. At bootstrap time, the contents of WCSA are UNPREDICTABLE. Figure 20-14 illustrates the microprogram breakpoint address register.

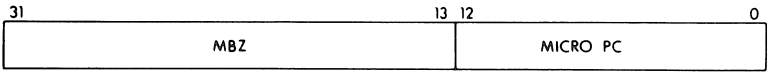
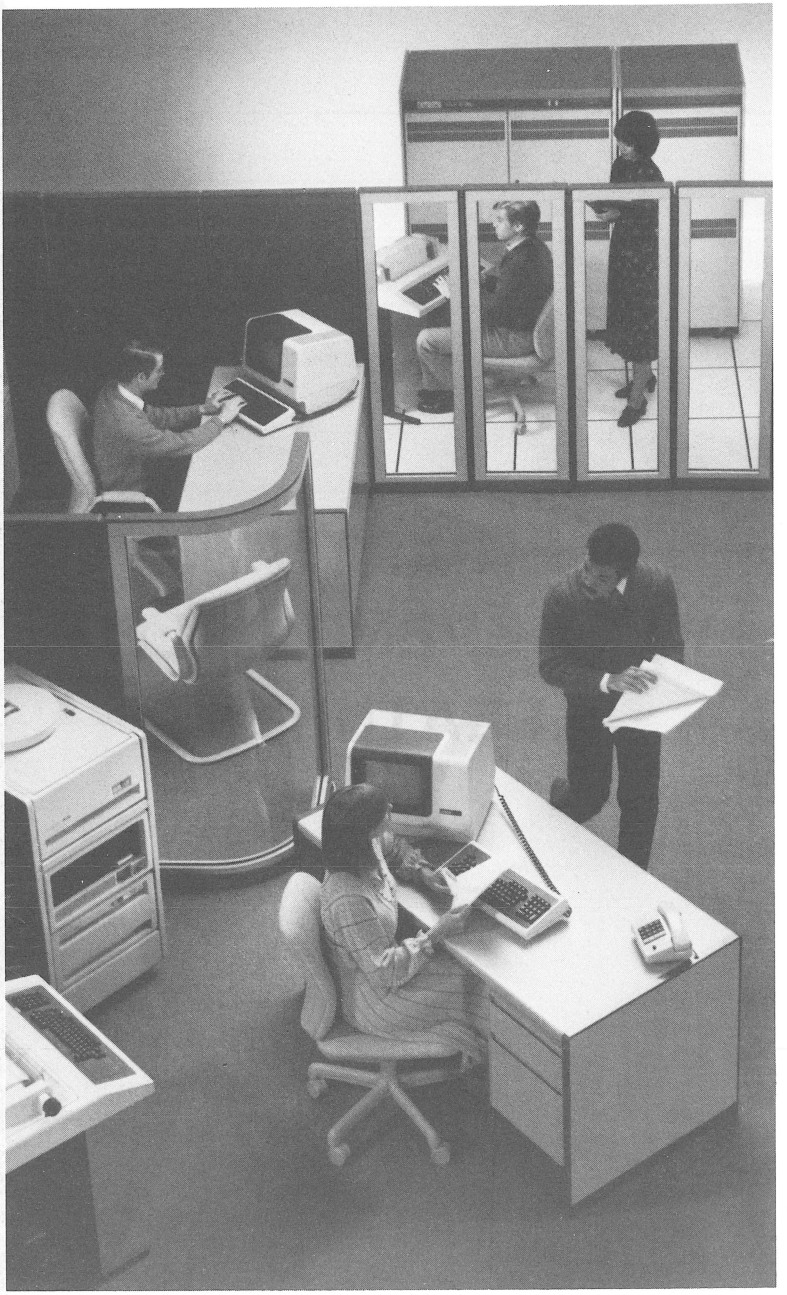


Figure 20-14 Microprogram Breakpoint Address Register (MBRK)

Whenever the microprogram PC matches the contents of MBRK, an external signal is asserted. If the console has enabled stop on microbreak, then the processor clock is stopped when this signal is asserted. If the console has not enabled microbreak, then this signal is available as a diagnostic scope point. Many diagnostics use the NOP instruction to trigger this method of giving a scope point. At bootstrap time, the contents of MBRK are UNPREDICTABLE.

PART V
VAX DEPENDABILITY FEATURES



CHAPTER 21

VAX SYSTEM DEPENDABILITY (RELIABILITY, AVAILABILITY, MAINTAINABILITY)

INTRODUCTION

This chapter describes the extensive reliability, availability, and maintainability features that are an integral part of VAX systems. These features were designed specifically to help prevent frequent system failures and, for certain types of failures, to allow continued system operation. These features were also developed to make VAX systems easier to repair when there is a failure. With better system reliability, optimal system availability, and easier component maintainability, uptime is increased and the overall cost of ownership is reduced.

The first section of this chapter describes the reliability, availability, and maintainability features common to the VAX-11/730, VAX-11/750, and VAX-11/780. The rest of the chapter provides brief descriptions of the system-specific features.

FEATURES COMMON TO VAX SYSTEMS

Error Detection and Reporting Features — Logic in the VAX hardware and software monitors system operation and distinguishes error conditions from normal system operation. The VAX/VMS operating system then records and reports detected error occurrences.

Consistency Checking/Error Checking — Continual consistency and error checking by VAX hardware and software increase data reliability by preventing certain error conditions from propagating through a database or a system. Checks detect abnormal instruction uses, transient and permanent hardware errors, and illegal arithmetic conditions. Some specific checks include:

- *Arithmetic Traps*—Traps occur when overflow, underflow and divide by zero arithmetic conditions are detected. Hardware detection of these error conditions allows checking to be used in high performance software where software checking would be prohibitively slow. Overflow and underflow traps may be enabled or disabled by setting flags in the Processor Status Longword, allowing the arithmetic exception conditions to be ignored if appropriate.
- *Limit Checking Traps*—Decimal string instructions all have length limit checks (0-31 decimal digits) performed on output strings to ensure that instructions do not overwrite adjacent data.

- *Reserved Operand Traps*—"Reserved-to-customer" and "reserved-to-DIGITAL" fields and opcodes ensure that customer extensions to the VAX architecture (e.g., user-defined instructions or data structures) do not conflict with future DIGITAL expansions.
- *Special Instruction Checks*—The CALLx and RETURN instructions have hardware-implemented register save/restore and consistency checking; these instructions provide a standard, identical interface for user routine calls and system calls. The CRC instruction provides block checking error code calculations, important in communications applications.

Additionally, the VAX/VMS operating system uses internal consistency checks to determine when data is not valid. In particular, checks determine the validity of system control information. System malfunctions cause a trap to an exception handling routine, with appropriate information recorded in the error log file.

Exception Handling — When any of the above consistency checks detect a failure, VAX/VMS uses a uniform condition handling facility to manage the hardware or software exception. Because the exception handling facility is uniformly consistent within the VAX system design, operation is more predictable and reliable.

Error Correcting Code — Memory error correcting code (ECC) automatically corrects all single-bit memory errors and detects double-bit memory errors. The code will also detect all greater than double-bit errors if the number of errors is even.

Disk error correcting code detects most errors and corrects errors in a single error burst of up to 11 bits.

ECC provides protection from nonrepeatable errors by automatically correcting data. Detections and corrections are noted in the error log as a preventive maintenance aid.

Environmental Conditions — See "Improved System Packaging."

Mass Storage I/O Verification — I/O verification for mass storage peripherals is supported by the VAX/VMS device drivers. The hardware compares each block for equivalence immediately after the block is read or written. Checking may be performed on all reads or writes to a file or volume, or specified for a single read or write. This capability increases reliability (at the expense of the time required to complete the read or write operation).

System Verification (UETP) — The VAX/VMS operating system contains an automated collection of verification software called the User Environmental Test Package (UETP). The UETP provides a comprehensive and systematic exercising of major peripheral devices and

software components by running most of the VAX/VMS utilities and language translators; calling most of the VAX/VMS System Services and I/O services each with a wide range of parameters; and comparing the results to known answers. The system reports errors to the error log and to the console terminal as execution proceeds.

The UETP is not designed to replace diagnostics, but serves as a means of validating the proper installation and functioning of a VAX system. A user can execute the UETP at any time to check system functions, to perform preliminary diagnosis, or to demonstrate system capability. Thus, in addition to verifying reliability, the package contributes to easier maintainability.

System Exerciser — The System Exerciser diagnostic program performs testing of various subsystems of a VAX system in a user environment (operating under VAX/VMS). It verifies hardware system integrity, or indicates those subsystems that may be failing or whose performance may be deteriorating. This testing is done automatically utilizing online diagnostic programs to perform the actual testing.

The System Exerciser runs as part of an extensive hardware reliability verification procedure, and is part of a total diagnostic hierarchy testing strategy. The System Exerciser is intended to be used as a dedicated process and will use all available system resources to do its job.

Automatic Online Error Logging — Error logging, a software tool used to monitor error occurrences, is an integral part of the VAX/VMS operating system. The error logging process is continual. The operating system accepts signals from the hardware and records CPU, memory, I/O, and software errors in an online log file. At the same time, the error logger notes as much information as possible about the state of the system at the time of an error. If no errors occur over a period of time, the error logger simply notes the time-of-day in the log file to record that the error logging process is running. (In the special case of ECC corrected memory errors, there is a threshold value. If the error rate exceeds this threshold, no more ECC log entries are made for a period of time.) A utility program is available to convert the log file into a meaningful format and summary which can be printed for later study.

Error logging has proved extremely beneficial for efficient maintenance of the hardware. It provides Field Service engineers with a report which can help them diagnose impending or persisting hardware problems; it assists them in detecting trends in fault occurrences; it helps them identify the specific subsystems which should be examined using diagnostics.

Machine Checks — Machine checks are hardware errors detected by

the central processor and reported to VAX/VMS. VAX/VMS categorizes the error and takes appropriate action. In many cases, the instruction in error is retried and operation continues normally (transient error). If the instruction retry fails, VAX/VMS attempts to limit the effects of the error to a single process. If the VAX/VMS executive is executing when the machine check occurs, the system is automatically rebootstrapped. Machine checks are caused by such conditions as cache, translation buffer, or control store parity errors, or by failure of the I/O or memory subsystems to respond to CPU requests.

System Identification (SID) Hardware Register — This register maintains information pertinent to the system processor type and revision number. This information may be examined (during the software error logging process, for example) to determine the engineering status of the processor.

Application Error Detection — VAX/VMS (and the optional software products) provides extensive checking for errors in application software. For example, the parameters passed to the System Services are checked for correctness before the service is attempted, thereby preventing the whole operating system (as well as the application) from being vulnerable to incorrectly coded programs. Additionally, tools such as the MACRO assembler and the Linker provide a high level of error detection so as to minimize the creation of erroneous application software. This latter class of error detection ranges from program syntax errors to detection of corrupted files (system libraries and/or user files) to inconsistency between separately compiled programs.

Deadlock Detection — VAX/VMS provides a general set of services (known as a Lock Manager) for synchronization of multiple processes and for queuing upon the availability of (named) resources they require. One of the more severe problems that can occur in such environments is known as deadlock, where each process is waiting for an event that none can declare to have occurred. The Lock Manager can detect the occurrence of such a situation, preventing it from rendering the application unusable.

Error Analysis and Recovery Features

VAX system hardware and software determines the source of an error and its extent and impact on the user's operation. Often the system can correct the error or mask its effects.

System Dump Analyzer — The System Dump Analyzer (SDA) is a VAX/VMS utility that helps determine the cause of an operating system failure. When an internal error interferes with normal operations, the operating system writes information concerning its status at the time of failure to a predefined system dump file. The SDA examines

and formats the content of this file. With the help of the SDA commands, a user can display parts of the formatted system dump file on a video display terminal, or can create a hardcopy listing. In addition to analyzing the system dump file, the System Dump Analyzer can perform its operations on a running system without interrupting system operation.

Error Log Reporting Program — A VAX/VMS utility program, called the System Error Analyzer, is available to convert the error log file into a meaningful format and summary which can be printed for study.

Instruction Retry — If a hardware-detected error, such as a machine check, interrupts the execution of an instruction, the system in some cases re-executes the instruction. Assuming the error is transient, normal operation continues. If the instruction cannot be retried or if the retry fails, VAX/VMS attempts to limit the failure to the user process currently executing; if the operating system is executing, the system is automatically rebootstrapped.

Nonfatal Bugchecks — When an error condition is detected by hardware or software and VAX/VMS determines that it affects only a single process, that process is removed from the system without affecting continued normal operation of the executive or other processes. The error is also logged in the system error log.

Automatic Stack Expansion — The VAX/VMS operating system automatically extends user stack space as needed.

Unattended Automatic System Restart — Automatic system restart facilities provide the ability to bring the system online, without operator intervention, after a system failure caused by a power interruption or a fatal software error. If the optional memory battery backup was able to preserve the contents of memory during the outage, the time-of-year clock with its own battery backup (up to 100 hours) allows VAX/VMS to recover the correct date and time upon power recovery. A special memory configuration register indicates to the recovery software whether data in memory was lost.

Following a power failure, whenever possible, all I/O in progress before the failure occurred is restarted, which helps guarantee system data integrity. Note that the power fail asynchronous trap facility may be used to initiate image-specific power fail recovery processing. If the battery backup option is not installed, or if the outage lasts longer than the battery backup can handle, the contents of memory are not valid and the system tries to rebootstrap itself.

Following a system failure, any unprocessed error log entries are written to the error log file. In addition, following a system failure, a dump

of physical memory is written to disk for later analysis.

The system operator can control whether restart after a system failure or power failure is automatic or manual by setting a switch on the processor control panel.

Automatic Reconfiguration — The VAX/VMS operating system allows users to continue working even though some of the hardware components have failed. Modification of the system configuration, both manually and automatically at system start-up time, provides a reliable subset of the system, which can be used until maintenance is performed on the failed components. For example:

- If the usual system device is unavailable, the system can be bootstrapped from any disk on which a system disk file can be loaded.
- If memory pages (512 bytes) are defective, memory is configured by VAX/VMS so that defective pages are not referenced (i.e., bad pages are placed in a bad page list). This is done both during bootstrapping and dynamically while the system is running. There is no dependence on any set of physical addresses to bootstrap the system.
- For a VAX-11/750 or VAX-11/780, if the memory cache is down for some reason, it may be disabled (at a performance reduction) because the cache is transparent to operation. This is done dynamically by VAX/VMS if excessive cache errors are detected.
- The VAX/VMS operating system automatically determines the presence of peripherals on the processor at bootstrap time and flags nonresponding devices (including memory units) as unavailable.
- Software spooling allows output to be generated even if the usual output device is not available. The system operator can reroute data to an alternate device using system operating commands.

Diagnostics — See "System Exerciser," "Online Functional Diagnostics," and "Fault-Isolation Diagnostics."

Dynamic Bad Block Handling — Bad blocks may occur when a disk surface becomes worn, or as a result of a failure in the disk drive that performed the data transfer. When the hardware detects a bad block during an I/O operation, the VAX/VMS operating system marks the header of the file in which the error occurred. When the file is eventually deallocated, the system checks the file header to see if any bad blocks exist in the file. If so, they are designated "permanently in use" and are not allocated for use by other files.

Bad Memory Page Replacement — When bad pages are detected, they are placed on a bad page list, both during bootstrapping and

during normal system operation.

Mass Storage Error Recovery — The operating system always attempts recovery from nonfatal disk and tape errors. If an error occurs during an I/O operation, the error is signalled by the hardware and the operation retried by VAX/VMS using all the available hardware recovery mechanisms.

Redundant Recording of Critical Disk Information — Critical information, such as the home block and index file header disk information, is redundantly recorded to allow its reliable recovery in the event of accidental destruction.

Selective Hardware Disabling — Several hardware elements—memory management, cache, translation buffer, and optional floating point accelerator—can be disabled by diagnostics to aid in isolating hardware problems. With the exception of memory management and the floating point accelerator, these elements are also dynamically disabled by VAX/VMS to allow continued operation at a reduced level of performance.

Data Integrity

The VAX/VMS operating system prevents processes from interfering with one another or with critical system data through hardware access protection and software enforced privileges.

Memory Management Hardware — The system's memory management hardware defines four hierarchical modes of memory access privilege termed (from most to least privileged) *kernel, executive, supervisor,* and *user*. Read and write access to memory is designated separately for each mode.

The VAX/VMS operating system is designed so that its critical components run in the most privileged access modes (kernel and executive). Thus, the system is well protected against read/write operations by any users not having the privilege to execute programs at the same levels. This "layered" design ensures system protection, as well as improving overall data reliability and integrity.

Quotas and Privileges — VAX/VMS uses a system of quotas to protect the use of shared system resources such as system dynamic memory and page file space. Quotas are assigned on a per-process basis. Thus, a given process cannot stop normal system operation by depleting shared resources.

User processes are prevented from affecting each other or the operating system by hardware access protection and a set of software enforced privileges. For a process which violates the protection rules,

intentionally or unintentionally, notification is given of the error so that appropriate recovery actions can be taken without affecting the rest of the system.

Access Control to Files and Volumes — The VAX/VMS operating system provides protection on a per-file basis. Users can be assigned read/write access to other user's data and files in an individually controlled manner. These controls may also be applied on a volume-wide basis.

Disk Volume Protection — The VAX/VMS file system provides the ability for the system manager to establish how much of a volume may be used by each user. This disk quota mechanism can be used to prevent one application from consuming all of the available data storage.

Each time a volume is mounted, it is automatically checked for consistency of the file structure and (in many cases) is repaired automatically. Such structure repair is often necessary if a failure of the whole system occurred while files were in use.

Another feature of the VAX/VMS file system is known as Mount Verification. This entails checking (after a transient condition such as a power failure of the system or the drive) that the volume that was in use is the one that is to be used afterwards. Thus, if for some reason the previous volume were removed and another substituted, VAX/VMS is able to prevent the accidental corruption of the new volume.

Maintenance Aids

VAX system packaging and diagnostic tools speed system repair operations and increase availability.

Improved System Packaging — The physical packaging of VAX systems has been designed so that all components are highly reliable and easily accessible for servicing.

Easy Access — VAX systems have been packaged so that all components are easily accessible for repair. Cables are fixed in place. CPU modules, CPU options, and memory arrays have no cables connected to them. Since the printed circuit card cage and backplane are fixed mounted into the system cabinet, not on slides, no service loops on internal cables are required.

Environmental Conditions (Power Loss, Temperature, and Air Flow Sensors) — VAX systems use sensors to detect emergency conditions. Indicators signal abnormal operating conditions: short circuit, over voltage, loss of power, regulator or main power board fail-

ure. This helps a Field Service engineer quickly determine the cause of failure.

Online Functional Diagnostics — One of the most impressive features of VAX systems is the ability to run functional diagnostics online under VAX/VMS. This means many problems can be isolated without taking the system down for stand-alone use.

The following specific functional diagnostics are implemented:

- Line printer
- Card reader
- Synchronous link
- Terminal
- Terminal exerciser
- Bus interaction
- Tape reliability
- Disk reliability
- Disk formatter

See also "System Exerciser."

Fault-Isolation Diagnostics — Once functional diagnostics have isolated faults to a particular subsystem or device, fault-isolation diagnostics can be run to pinpoint the problem to the smallest possible element.

Fault-isolation diagnostics run in a stand-alone environment and isolate problems to a field-replaceable unit. This means that Field Service spends less time diagnosing the fault. It also means that the cost of replacement parts is kept to a minimum by isolating the fault to the smallest possible unit. In addition, Field Service can run microdiagnostics (the optional remote diagnostic module (RDM) is needed on the VAX-11/750) even when the processor is completely down. In addition, VAX systems have a bus which allows microdiagnostics to exercise the hardware to pinpoint faults. This greatly reduces repair time, contributing to both maintainability and availability.

Online Software Update and Maintenance — The system operator can perform software update and maintenance activities without bringing the system down for stand-alone use. Software updates are distributed in machine readable form (on floppy diskette or tape cartridge). The operator can update software modules on disk with patches and replacement modules, concurrent with normal system activities. Note: A rebootstrap operation may be necessary to activate the newly installed modules.

The system operator can also perform software maintenance pro-

cedures online. The operator can perform disk backup concurrent with normal activities. Because these activities are performed online, both system availability and maintainability are increased.

Automatic Updates and Patches — VAX/VMS implements a system of automatic updates and patches.(These software updates are performed in such a way that, if there is a power failure, the operation can be continued when the system is restarted.) The executive is under "ECO" control in that each patch automatically checks for required previous patches and updates the current revision number. The patches are always distributed and applied in machine readable form, thus eliminating the possibility of introducing errors during transcription.

High Resolution Interval Clock — An interval clock is used by diagnostics to test time-dependent functions without requiring machine specific timing loops in programs. It is also used by VAX/VMS to time-stamp operations.

Selective Hardware Disabling — See previous description.

Maintenance Registers — These registers contain bus-specific maintenance information and can be examined at the time of an error to help determine the cause.

VAX-11/730-SPECIFIC FEATURES

Customer Runnable Diagnostics — Customer Runnable Diagnostics (CRDs) allow a system user the capability of easily verifying proper hardware operation. They also allow the quick isolation of system failures to the subsystem or device level. Any error information provided from a CRD session can then be provided to the DIGITAL Field Service office dispatcher. In this way, a specialist with the proper spare parts and tools for a specific failure is dispatched to the customer site. The advantage of this method is that most failures are corrected within a single service visit.

The CRDs have been designed to operate in three modes. These three modes provide a test strategy that will provide error information regardless of the severity of the failure. The following modes of operation are provided:

- Autotest—Autotest provides an efficient means to verify the operation of the CPU, the VMS system disk, and the diagnostic load disk. As this mode makes no assumptions about the status of the machine, it is especially useful in determining the cause of a non-bootable system. Autotest is invoked by typing "TEST" at the con-

sole prompt. Informational messages are displayed at regular intervals to indicate test progress. The successful completion of Autotest initiates the off-line menu mode.

- Off-line Menu—The off-line menu system has been designed to allow a user to verify the proper operation of any device on the system. The concise menu format is designed to provide a simple interface for users that are unfamiliar with VAX diagnostics. All standard VAX-11/730 options and devices are supported. When necessary, the user is prompted to supply necessary device preparation.
- On-line Menu—The on-line menu system allows the user the capability of verifying the proper operation of a device while operating under VMS. Thus, device testing is performed without disturbing other users or processes on the system. Terminals, lineprinters, and some other peripheral devices that in general never interfere with the overall operation of VMS can only be tested using the on-line menu mode. In addition, devices that have both on-line and off-line support should be checked with both menu modes to effect total device verification.

Remote Support — Remote Support provides the DIGITAL service engineer on site with a further level of technical support that can be used when the need arises. It provides customers with maximum protection against extended downtime. For DECSERVICE customers, it has the added benefit of a service feature called Remote Hardware Monitoring (RHM). Under RHM, DIGITAL Field Service will set up a schedule with the customer for running periodic remote hardware checks of the system. This program allows DIGITAL to identify potential problems and schedule maintenance before a costly downtime situation occurs.

Option Replacement — The VAX-11/730's memory arrays, CPU modules, the optional floating point accelerator (FP730), the integrated disk controller, and the optional DMF32 communication board can all be removed easily for quick replacement.

Modular Power Supply — The power system for the VAX-11/730 consists of three modules: a power controller and two power regulators. Any of the three modules can be replaced in less than 20 minutes, meaning increased maintainability and availability. The power supply indicators are visible on the top of the power supply. In addition, a universal power supply is used.

VAX-11/730 Microverify Routines — (Bootstrapping Selfdiagnosis) The VAX-11/730 executes microverify routines automatically on a power-up sequence, when the VAX system is bootstrapped, or when the console front panel RESET switch is pushed. The routines test the

data paths. Successful execution of the microverify routines indicates that the system should bootstrap predictably.

Dual TU58 Tape Drives — The redundant TU58 means that the VAX-11/730 can still operate if one of the TU58s is inoperative. This provides higher system availability.

Parity Checking — The VAX-11/730 performs parity checking on any words read out of the WCS before execution.

Air Flow Sensors — These sensors detect environmental change and blockage of airflow through the module card cage.

VAX-11/750-SPECIFIC FEATURES

VAX-11/750 Microverify Routines — (Bootstrapping Selfdiagnosis) The VAX-11/750 executes microverify routines automatically on a power-up sequence, when the VAX system is bootstrapped, or when the console front panel RESET switch is pushed. The routines test the data paths, the 16 general-purpose registers, most internal CPU registers, the instruction prefetch buffer, the parity logic of the cache, the translation buffer, and all of the cache memory. Successful execution of the microverify routines indicates that the system should bootstrap predictably.

Option Replacement — The VAX-11/750's MASSBUS adapter, memory arrays, user control store, and remote diagnosis module are each single printed circuit board options which can be removed or replaced in a few minutes.

Modular Power Supply — The power system for the VAX-11/750 consists of three assemblies, any of which can be replaced in less than 20 minutes. Since power supply indicators are on the outside rear of the cabinet, a user can determine when a power supply has failed and quickly notify Field Service.

Remote Diagnosis — Remote diagnosis (RD), an option available to customers in North America and parts of Europe, can lower a customer's maintenance cost, as well as increase overall system availability. With the RD option, a customer experiencing hardware problems may take advantage of the resources at a DIGITAL Diagnostic Center where experienced staff are available seven days a week, 24 hours a day. Each Center has a host diagnostic computer that aids in pinpointing system faults.

After receiving a problem call from a user, a technician at the Center dials up the customer's computer and logs onto the VAX system. Unauthorized access by the Center is impossible because a custom-

er's system will not answer the RD line unless the local operator has performed specific procedures. Once the connection is established, the response technician can run tests and diagnostic procedures (some of which can execute without requiring a stand-alone environment), initiate diagnostics at all levels, and examine memory locations and the error log file. If servicing is necessary, repair time is often substantially reduced because problems are known in advance and the engineer who actually makes the service call will be carrying the right parts.

Preventive maintenance diagnostic sessions are also offered through remote diagnosis. These sessions can be scheduled for off-peak periods so as not to interfere with system availability.

Diagnostic Console — The diagnostic console, part of the Remote Diagnosis Module option on the VAX-11/750, can access the central processor's major buses and key control points through a special internal diagnostic bus. The console allows operator diagnostic operations through simple keyboard commands. (The diagnostic console can also serve as an operator console and as a user terminal.) The diagnostic console includes a cartridge tape drive and hardcopy terminal.

Parity and Protocol Checks — VAX systems perform parity checks on MASSBUS data, control and address translation; UNIBUS address translation; memory cache data and address; address translation buffer transactions; microcode and user control store.

Improved Air Flow — The VAX-11/750 uses one blower to pull air in at the top of the cabinet, down through the modules, then through the power supplies, and exhaust air out at the bottom rear of the cabinet. By moving air through the cabinet in this fashion, the system receives air that is not contaminated with dirt particles from the floor.

Blowers operate well below maximum ratings, which extends their operating life significantly. Additionally, the cooling air flow is adequate to operate even when all cabinet doors are open, allowing online servicing of modules.

VAX-11/780-SPECIFIC FEATURES

Remote Diagnosis — Remote Diagnosis features for the VAX-11/780 are the same as for the VAX-11/750. For a detailed description, see "Remote Diagnosis" under the VAX-11/750-Specific Features section of this chapter.

Diagnostic Console — The diagnostic console, standard on the VAX-

11/780, can access the central processor's major buses and key control points through a special internal diagnostic bus. The console allows operator diagnostic operations through simple keyboard commands. (The diagnostic console can also serve as an operator console and as a user terminal.) It includes an LSI-11 microcomputer, floppy disk, and hardcopy terminal. A watchdog timer in the LSI-11 detects hung machine conditions (such as a failure to fetch instructions).

VAX-11/780 UNIBUS Adapter Recovery — When the UNIBUS adaptor detects certain error conditions on the UNIBUS, the conditions are reported in the error log. If the conditions persist, the UNIBUS is reinitialized and all I/O operations currently in progress on that UNIBUS are restarted.

Option Replacement — VAX-11/780 subassemblies have dedicated backpanels. Their replacement can be done in less than 20 minutes by one person using the tools ordinarily carried in the Field Service repair kit.

Modular Power Supply — The power system for the VAX-11/780 is modular in 500 watt increments. There are spaces within the cabinet for six power supplies. Four are part of the base configuration; the remaining two power supplies are for the optional floating point accelerator and MASSBUS adapters. Replacement of these modules can be done easily by one person.

Powering Down VAX-11/780 UNIBUS Peripherals — The VAX-11/780 UNIBUS adapter or any VAX-11/780 UNIBUS peripheral cabinet can be separately powered on/off during normal system operation. This allows online replacement of devices which have been diagnosed as faulty on the UNIBUS. All operations in progress on other devices are restarted automatically when the UNIBUS is powered on.

Sixteen-Level Silo — The Synchronous Backplane Interconnect on the VAX-11/780 has a 16-level silo which monitors the VAX-11/780's central bus activity and contains a history of the 16 most recent cycles of bus activity. The SBI also includes parity. If an error or predetermined special condition occurs, the silo is latched, meaning the error or condition causes the silo contents to freeze, and can be examined to help determine the cause of the problem.

Clock Margining — The VAX-11/780 uses clock margining to vary the central bus clock rate via console commands. This can aid the Field Service engineer in diagnosing intermittent hardware problems.

Parity and Protocol Checks — VAX systems perform parity checks on MASSBUS data, control and address translation; UNIBUS address translation; memory cache data and address; address translation

buffer transactions; microcode and user control store.

Improved Air Flow — The VAX-11/870 uses three blowers to pull air in at the top of the cabinet, down through the modules, then through the power supplies, and exhaust air out at the bottom rear of the cabinet. By moving air through the cabinet in this fashion, the system receives air that is not contaminated with dirt particles from the floor.

Blowers operate well below maximum ratings, which extends their operating life significantly. Additionally, the cooling air flow is adequate to operate even when all cabinet doors are open, allowing online servicing of modules.

PART VI
APPENDICES AND GLOSSARY

APPENDIX A
COMMONLY USED MNEMONICS

ACP	Ancillary Control Process
ANSI	American National Standard Institute
AP	Argument Pointer
ASCII	American Standard Code for Information Interchange
AST	Asynchronous System Trap
ASTLVL	Asynchronous System Trap Level
CCB	Channel Control Block
CM	Compatibility Mode Bit in the Hardware PSL
CRB	Channel Request Block
CRC	Cyclic Redundancy Check
CRD	Customer Runnable Diagnostics
CSR	Control Status Register
DAP	Data Access Protocol
DDB	Device Data Block
DDC	DIGITAL Diagnostic Center
DDCMP	DIGITAL Data Communications Message Protocol
DDT	Driver Data Table
DMA	Direct Memory Access
DV	Decimal Overflow Trap Enable Bit in the PSW
ECB	Exit Control Block
ECC	Error Correction Code
ESP	Executive Mode Stack Pointer
ESR	Exception Service Routine
F11ACP	Files-11 Ancillary Control Process
FAB	File Access Block
FCA	Fixed Control Area
FCB	Fixed Control Block
FCS	File Control Services
FDT	Function Decision Table
FP	Frame Pointer
FPA	Floating Point Accelerator
FPD	First Part (of an instruction) Done
FU	Floating Underflow Trap Enable Bit in the PSW
GSD	Global Section Descriptor
GST	Global Symbol Table
IDB	Interrupt Dispatch Block
IDC	Integrated Disk Controller
IPL	Interrupt Priority Level
IRP	I/O Request Packet
ISECT	Image Section

ISD	Image Section Descriptor
ISP	Interrupt Stack Pointer
IS	Interupt Stack Bit in PSL
ISR	Interrupt Service Routine
IV	Integer Overflow Trap Enable Bit in the PSW
KSP	Kernel Mode Stack Pointer
MBA	MASSBUS Adaptor
MBZ	Must Be Zero
MCR	Monitor Console Routine
MFD	Master File Directory
MFPR	Move From Process Register Instruction
MME	Memory Mapping Enable
MOS	Metal Oxide Semiconductor
MTPR	Move To Process Register Instruction
MUTEX	Mutual Exclusion Semaphore
NSP	Network Services Protocol
OPCOM	Operator Communication Manager
P0BR	Program Region Base Register
P0LR	Program Region Length Register
P0PT	Program Region Page Table
P1BR	Control Region Base Register
P1LR	Control Region Limit Register
P1PT	Control Region Page Table
PAL	Programmed Array Logic
PC	Program Counter
PCB	Process Control Block
PCBB	Process Control Block Base Register
PFN	Page Frame Number
PID	Process Identification Number
PME	Performance Monitor Enable Bit in PCB
PSECT	Program Section
PSL	Processor Status Longword
PSW	Processor Status Word
PTE	Page Table Entry
QIO	Queue Input/Output Request System Service
RAB	Record Access Block
RAM	Random Access Memory
RD	Remote Diagnosis
RFA	Record's File Address
RMS	Record Management Services
ROM	Read Only Memory
RS	Remote Support
RWED	Read, Write, Execute, Delete
SBR	System Base Register

SCB	System Control Block
SCBB	System Control Block Base Register
SLR	System Length Register
SP	Stack Pointer
SPT	System Page Table
SSP	Supervisor Mode Stack Pointer
SVA	System Virtual Address
TP	Trace Trap Pending Bit in PSL
UBA	Unibus Adapter
UCB	Unit Control Block
UCS	User Control Store
UETP	User Environment Test Package
UFD	User File Directory
UIC	User Identification Code
USP	User mode Stack Pointer
VAX	Virtual Address Extension
VCB	Volume Control Block
VMS	Virtual Memory System
VPN	Virtual Page Number
WCB	Window Control Block

APPENDIX B
INSTRUCTION INDEX
By Mnemonic

MNEMONIC LISTING

MNEMONIC	INSTRUCTION	OPCODE	PAGE
ACBB	Add compare and branch byte	9D	268
ACBD	Add compare and branch D_floating	6F	268
ACBF	Add compare and branch F_floating	4F	268
ACBG	Add compare and branch G_floating	4FFD	268
ACBH	Add compare and branch H_floating	6FFD	268
ACBL	Add compare and branch longword	F1	268
ACBW	Add compare and branch word	3D	268
ADAWI	Add aligned word, interlocked	58	194
ADDB2	Add byte 2 operand	80	191
ADDB3	Add byte 3 operand	81	191
ADDD2	Add D_floating 2 operand	60	191
ADDD3	Add D_floating 3 operand	61	191
ADDF2	Add F_floating 2 operand	40	191
ADDF3	Add F_floating 3 operand	41	191
ADDG2	Add G_floating 2 operand	40FD	191
ADDG3	Add G_floating 3 operand	41FD	191
ADDH2	Add H_floating 2 operand	60FD	191
ADDH3	Add H_floating 3 operand	61FD	191
ADDL2	Add longword 2 operand	C0	191
ADDL3	Add longword 3 operand	C1	191
ADDP4	Add packed 4 operand	20	314
ADDP6	Add packed 6 operand	21	314
ADDW2	Add word 2 operand	A0	191
ADDW3	Add word 3 operand	A1	191
ADWC	Add with carry	D8	193
AOBLEQ	Add one and branch on less or equal	F3	270
AOBLSS	Add one and branch on less	F2	270
ASHL	Arithmetic shift longword	78	211
ASHP	Arithmetic shift and round packed	F8	330
ASHQ	Arithmetic shift quadword	79	211

MNEMONIC	INSTRUCTION	OPCODE	PAGE
BBC	Branch on bit clear	E1	264
BBCC	Branch on bit clear and clear	E5	265
BBCCI	Branch on bit clear and clear interlocked	E7	266
BBCS	Branch on bit clear and set	E3	265
BBS	Branch on bit set	E0	264
BBSC	Branch on bit set and clear	E4	265
BBSS	Branch on bit set and set	E2	265
BBSSI	Branch on bit set and set, interlocked	E6	266
BCC	Branch on carry clear	1E	261
BCS	Branch on carry set	1F	261
BEQL	Branch on equal (signed)	13	261
BEQLU	Branch on equal unsigned	13	261
BGEQ	Branch on greater or equal	18	261
BGEQU	Branch on greater or equal unsigned	1E	
BGTR	Branch on greater	14	261
BGTRU	Branch on greater unsigned	1A	261
BICB2	Bit clear byte 2 operand	8A	209
BICB3	Bit clear byte 3 operand	8B	209
BICL2	Bit clear longword 2 operand	CA	209
BICL3	Bit clear longword 3 operand	CB	209
BICPSW	Bit clear program status word	B9	223
BICW2	Bit clear word 2 operand	AA	209
BICW3	Bit clear word 3 operand	AB	209
BISB2	Bit set byte 2 operand	88	208
BISB3	Bit set byte 3 operand	89	208
BISL2	Bit set long 2 operand	C8	208
BISL3	Bit set long 3 operand	C9	208
BISPSW	Bit set program status word	B8	223
BISW2	Bit set word 2 operand	A8	208
BISW3	Bit set word 3 operand	A9	208
BITB	Bit test byte	93	207
BITL	Bit test longword	D3	207
BITW	Bit test word	B3	207
BLBC	Branch on low bit clear	E9	267
BLBS	Branch on low bit set	E8	267

MNEMONIC	INSTRUCTION	OPCODE	PAGE
BLEQ	Branch on less or equal	15	261
BLEQU	Branch on less or equal unsigned	1B	261
BLSS	Branch on less	19	261
BLSSU	Branch on less unsigned	1F	261
BNEQ	Branch on not equal	12	261
BNEQU	Branch on not equal unsigned	12	261
BPT	Break point fault	03	169
BRB	Branch with byte displacement	11	263
BRW	Branch with word displacement	31	263
BSBB	Branch to subroutine with byte displacement	10	275
BSBW	Branch to subroutine with word displacement	30	275
BUGL	Bugcheck longword	FDFF	170
BUGW	Bugcheck word	FEFF	170
BVC	Branch on overflow clear	1C	261
BVS	Branch on overflow set	1D	261
CALLG	Call with general argument list	FA	280
CALLS	Call with stack	FB	282
CASEB	Case byte	8F	273
CASEL	Case longword	CF	273
CASEW	Case word	AF	273
CHME	Change mode to executive	BD	158
CHMK	Change mode to kernel	BC	158
CHMS	Change mode to supervisor	BE	158
CHMU	Change mode to user	BF	158
CLRB	Clear byte	94	181
CLRD	Clear D_floating	7C	181
CLRF	Clear F_floating	D4	181
CLRG	Clear G_floating	7C	181
CLRH	Clear H_floating	7CFD	181
CLRL	Clear longword	D4	181
CLRO	Clear octaword	7CFD	181
CLRQ	Clear quadword	7C	181
CLRW	Clear word	B4	181
CMPB	Compare byte	91	188
CMPC3	Compare character 3 operand	29	294

MNEMONIC	INSTRUCTION	OPCODE	PAGE
CMPC5	Compare character 5 operand	2D	294
CMPD	Compare D_floating	71	188
CMPF	Compare F_floating	51	188
CMPG	Compare G_floating	51FD	188
CMPH	Compare H_floating	71FD	188
CMPL	Compare longword	D1	188
CMPP3	Compare packed 3 operand	35	313
CMPP4	Compare packed 4 operand	37	313
CMPV	Compare field	EC	255
CMPW	Compare word	B1	188
CMPZV	Compare zero-extended field	ED	255
CRC	Calculate cyclic redundancy check	0B	304
CVTBD	Convert byte to D_floating	6C	184
CVTBF	Convert byte to F_floating	4C	184
CVTBG	Convert byte to G_floating	4CFD	184
CVTBH	Convert byte to H_floating	6CFD	184
CVTBL	Convert byte to longword	98	184
CVTBW	Convert byte to word	99	184
CVTDB	Convert D_floating to byte	68	184
CVTDF	Convert D_floating to F_floating	76	184
CVTDH	Convert D_floating to H_floating	32FD	184
CVTDL	Convert D_floating to longword	6A	184
CVTDW	Convert D_floating to word	69	184
CVTFB	Convert F_floating to byte	48	184
CVTFD	Convert F_floating to D_floating	56	184
CVTFG	Convert F_floating to G_floating	99FD	184
CVTFH	Convert F_floating to H_floating	98FD	184
CVTFL	Convert F_floating to longword	4A	184
CVTFW	Convert F_floating to word	49	184
CVTGB	Convert G_floating to byte	48FD	184
CVTGF	Convert G_floating to F_floating	33FD	184
CVTGH	Convert G_floating to H_floating	56FD	184
CVTGL	Convert G_floating to longword	4AFD	184
CVTGW	Convert G_floating to word	49FD	184
CVTHB	Convert H_floating to byte	68FD	184
CVTHD	Convert H_floating to D_floating	F7FD	184
CVTHF	Convert H_floating to F_floating	F6FD	184
CVTHG	Convert H_floating to G_floating	76FD	184
CVTHL	Convert H_floating to longword	6AFD	184
CVTHW	Convert H_floating to word	69FD	184

MNEMONIC	INSTRUCTION	OPCODE	PAGE
CVTLB	Convert longword to byte	F6	184
CVTLD	Convert longword to D_floating	6E	184
CVTLF	Convert longword to F_floating	4E	184
CVTLG	Convert longword to G_floating	4EFD	184
CVTLH	Convert longword to H_floating	6EFD	184
CVTLP	Convert longword to packed	F9	321
CVTLW	Convert longword to word	F7	184
CVTPL	Convert packed to longword	36	322
CVTTP	Convert trailing numeric to packed	26	325
CVTPT	Convert packed to trailing numeric	24	323
CVTPS	Convert packed to leading separate numeric	08	327
CVTRDL	Convert rounded D_floating to longword	6B	184
CVTRFL	Convert rounded F_floating to longword	4B	184
CVTRGL	Convert rounded G_floating to longword	4BFD	184
CVTRHL	Convert rounded H_floating to longword	6BFD	184
CVTSP	Convert leading separate numeric to packed	09	329
CVTWB	Convert word to byte	33	184
CVTWD	Convert word to D_floating	6D	184
CVTWF	Convert word to F_floating	4D	184
CVTWG	Convert word to G_floating	4DFD	184
CVTWH	Convert word to H_floating	6DFD	184
CVTWL	Convert word to longword	32	184
DECB	Decrement byte	97	197
DECL	Decrement longword	D7	197
DECW	Decrement word	B7	197
DIVB2	Divide byte 2 operand	86	204
DIVB3	Divide byte 3 operand	87	204
DIVD2	Divide D_floating 2 operand	66	204
DIVD3	Divide D_floating 3 operand	67	204
DIVF2	Divide F_floating 2 operand	46	204

MNEMONIC	INSTRUCTION	OPCODE	PAGE
DIVF3	Divide F_floating 3 operand	47	204
DIVG2	Divide G_floating 2 operand	46FD	204
DIVG3	Divide G_floating 3 operand	47FD	204
DIVH2	Divide H_floating 2 operand	66FD	204
DIVH3	Divide H_floating 3 operand	67FD	204
DIVL2	Divide longword 2 operand	C6	204
DIVL3	Divide longword 3 operand	C7	204
DIVP	Divide packed	27	319
DIVW2	Divide word 2 operand	A6	204
DIVW3	Divide word 3 operand	A7	204
EDITPC	Edit packed to character	38	335
EDIV	Extended divide	7B	206
EMODD	Extended modulus D_floating	74	202
EMODF	Extended modulus F_floating	54	202
EMODG	Extended modulus G_floating	54FD	202
EMODH	Extended modulus H_floating	74FD	202
EMUL	Extended multiply	7A	201
EXTV	Extract field	EE	253
EXTZV	Extract zero-extended field	EF	253
FFC	Find first clear bit	EB	251
FFS	Find first set bit	EA	251
HALT	Halt	00	171
INCB	Increment byte	96	189
INCL	Increment longword	D6	189
INCW	Increment word	B6	189
INDEX	Compute index	0A	226
INSQHI	Insert into queue head, interlocked	5C	240
INSQTI	Insert into queue tail, interlocked	5D	240
INSQUE	Insert into queue	0E	232
INSV	Insert field	F0	257
JMP	Jump	17	263
JSB	Jump to subroutine	16	275
LDPCTX	Load process context	06	163
LOCC	Locate character	3A	299

MNEMONIC	INSTRUCTION	OPCODE	PAGE
MATCHC	Match characters	39	301
MCOMB	Move complemented byte	92	183
MCOML	Move complemented long	D2	183
MCOMW	Move complemented word	B2	183
MFPR	Move from privilege register	DB	165
MNEGB	Move negated byte	8E	187
MNEGD	Move negated D_floating	72	182
MNEGF	Move negated F_floating	52	182
MNEGG	Move Negated G_floating	52FD	182
MNEGH	Move Negated H_floating	72FD	182
MNEGL	Move negated longword	CE	182
MNEGW	Move negated word	AE	182
MOVAB	Move address of byte	9E	224
MOVAD	Move address of D_floating	7E	224
MOVAF	Move address of F_floating	DE	224
MOVAG	Move Address of G_floating	7E	224
MOVAH	Move Address of H_floating	7EFD	224
MOVAL	Move address of longword	DE	224
MOVAO	Move Address of octaword	7EFD	224
MOVAQ	Move address of quadword	7E	224
MOVAW	Move address of word	3E	224
MOVB	Move byte	90	179
MOVC3	Move character 3 operand	28	289
MOVC5	Move character 5 operand	2C	289
MOVD	Move D_floating	70	179
MOVF	Move F_floating	50	179
MOVG	Move G_floating	50FD	179
MOVH	Move H_floating	70FD	179
MOVL	Move longword	D0	179
MOVO	Move octaword	7DFD	179
MOVP	Move packed	34	312
MOVPSL	Move processor status longword	DC	222
MOVQ	Move quadword	7D	179
MOVTC	Move translated characters	2E	290
MOVTUC	Move translated until character	2F	292
MOVW	Move word	B0	179
MOVZBL	Move zero-extended byte to longword	9A	187

MNEMONIC	INSTRUCTION	OPCODE	PAGE
MOVZBW	Move zero-extended byte to word	9B	187
MOVZWL	Move zero-extended word to longword	3C	187
MTPR	Move to privilege register	DA	165
MULB2	Multiply byte 2 operand	84	199
MULB3	Multiply byte 3 operand	85	199
MULD2	Multiply D_floating 2 operand	64	199
MULD3	Multiply D_floating 3 operand	65	199
MULF2	Multiply F_floating 2 operand	44	199
MULF3	Multiply F_floating 3 operand	45	199
MULG2	Multiply G_floating 2 operand	44FD	199
MULG3	Multiply G_floating 3 operand	45FD	199
MULH2	Multiply H_floating 2 operand	64FD	199
MULH3	Multiply H_floating 3 operand	65FD	199
MULL2	Multiply longword 2 operand	C4	199
MULL3	Multiply longword 3 operand	C5	199
MULP	Multiply packed	25	318
MULW2	Multiply word 2 operand	A4	199
MULW3	Multiply word 3 operand	A5	199
NOP	No operation	01	
POLYD	Evaluate polynomial D_floating	75	214
POLYF	Evaluate polynomial F_floating	55	214
POLYG	Evaluate polynomial G_floating	55FD	214
POLYH	Evaluate polynomial H_floating	75FD	214
POPR	Pop registers	BA	221
PROBER	Probe read access	0C	160
PROBEW	Probe write access	0D	160
PUSHAB	Push address byte	9F	224
PUSHAD	Push address of D_floating	7F	224
PUSHAF	Push address of F_floating	DF	224
PUSHAG	Push Address of G_floating	7F	224
PUSHAH	Push Address of H_floating	7FFD	224
PUSHAL	Push address of longword	DF	224
PUSHAO	Push address of octaword	7FFD	224
PUSHAQ	Push address of quadword	7F	224
PUSHAW	Push address of word	3F	224
PUSHL	Push longword	DD	180
PUSHR	Push registers	BB	220

MNEMONIC	INSTRUCTION	OPCODE	PAGE
REI	Return from exception or interrupt	02	161
REMQHI	Remove from queue head, interlocked	5E	245
REMQTI	Remove from queue tail, interlocked	5F	248
REMQUE	Remove from queue	0F	234
RET	Return from called procedure	04	284
ROTL	Rotate longword	9C	212
RSB	Return from subroutine	05	276
SBWC	Subtract with carry	D9	198
SCANC	Scan for character	2A	297
SKPC	Skip character	3B	299
SOBGEQ	Subtract one and branch on greater or equal	F4	271
SOBGTR	Subtract one and branch on greater	F5	271
SPANC	Span characters	2B	297
SUBB2	Subtract byte 2 operand	82	195
SUBB3	Subtract byte 3 operand	83	195
SUBD2	Subtract D_floating 2 operand	62	195
SUBD3	Subtract D_floating 3 operand	63	195
SUBF2	Subtract F_floating 2 operand	42	195
SUBF3	Subtract F_floating 3 operand	43	195
SUBG2	Subtract G_floating 2 operand	42FD	195
SUBG3	Subtract G_floating 3 operand	43FD	195
SUBH2	Subtract H_floating 2 operand	62FD	195
SUBH3	Subtract H_floating 3 operand	63FD	195
SUBL2	Subtract longword 2 operand	C2	195
SUBL3	Subtract longword 3 operand	C3	195
SUBP4	Subtract packed 4 operand	22	316
SUBP6	Subtract packed 6 operand	23	316
SUBW2	Subtract word 2 operand	A2	195
SUBW3	Subtract word 3 operand	A3	195
SVPCTX	Save process context	07	163
TSTB	Test byte	95	190
TSTD	Test D_floating	73	190
TSTF	Test F_floating	53	190
TSTG	Test G_floating	53FD	190
TSTH	Test H_floating	73FD	190
TSTL	Test long	D5	190
TSTW	Test word	B5	190

MNEMONIC	INSTRUCTION	OPCODE	PAGE
XFC	Extended function call	FC	168
XORB2	Exclusive OR byte 2 operand	8C	210
XORB3	Exclusive OR byte 3 operand	8D	210
XORL2	Exclusive OR longword 2 operand	CC	210
XORL3	Exclusive OR longword 3 operand	CD	210
XORW2	Exclusive OR word 2 operand	TC	210
XORW3	Exclusive OR word 3 operand	AD	210
ESCD	Reserved to DIGITAL	FD	
ESCE	Reserved to DIGITAL	FE	
ESCF	Reserved to DIGITAL	FF	
Reserved to DIGITAL		57;59;5A;5B;77; 00FD to 31FD; 34FD to 3FFD; 57FD, 58FD, ...5FFD; 77FD,78FD, ...7FFD; 80FD to 97FD; 9AFD to F5FD; F8FD to FCFF.	

APPENDIX C
ADDRESS VALIDATION RULES

The memory management system described in Chapter 4 separates validation from the access of arguments. It is necessary to adopt certain coding conventions to prohibit unauthorized user access to sensitive data. Specifically it must not be possible for a user to call an inner access mode in such a way that will corrupt system integrity (e.g., cause supervisory code to write over itself) or incorrectly allow access to data that would otherwise have been inaccessible (e.g., the reading of a password table).

The following discussion sets forth operating system requirements that must be adhered to when accessing arguments from an inner access mode to avoid a breach of security.

The following requirements are made concerning operating system software:

1. Operating system software (kernel and executive mode) is trustworthy and does not maliciously attempt to break down the protection mechanisms (e.g., change the mapping or protection of pages at arbitrary times).
2. The protection of a shared page may not be changed unless the share count (a software construct) is one and the process attempting the change is that sharer. Share count=a software maintained record of the number of processes sharing a page.
3. The protection of a page with a nonzero I/O pending count (a software construct) may not be changed until the count goes to zero.
4. Operating system software will not deliver ASTs to outer access modes while the process is executing in an inner access mode.
5. Arguments passed to an inner access mode can be maliciously destroyed asynchronously by another process (e.g., shared data) or by an I/O transfer, but not by a less privileged mode of the executing process itself.
6. Kernel and executive stacks are never allocated in shared memory or accessible to other than their respective access modes.

The following summarizes related aspects of the VAX hardware:

1. Four access modes are provided and there is a stack per-process per-access mode.
2. Protection is hierarchical with the innermost access mode being the least restricted and the outermost the most restricted.

3. Four instructions are provided to change the processor mode to the four access modes (CHMU, CHMS, CHME, and CHMK); furthermore, when a process is executing a change mode instruction the access mode can only be decreased (changed to a more privileged mode) or left the same.
4. Two instructions are provided to validate the accessibility of arguments: Probe Read (PROBER) and Probe Write (PROBEW). These instructions validate the accessibility of arguments using the maximization of the Previous Mode field of PSL and a specified access mode. Thus only current and more restricted access modes can be probed.
5. The Return from Interrupt instruction (REI) insures that the current mode field of the restored PSL is greater than or equal to the current mode field of the current PSL and that the previous mode field of the restored PSL is greater than or equal to the current mode field of the restored PSL.

Given the previous operating system requirements, the following rules guarantee that less privileged modes cannot pass erroneous addresses to more privileged modes.

1. All addresses (including indirect addresses) passed as arguments to an inner access mode must be copied (preferably to a register, but in any case to an area of memory that is not modifiable by less privileged modes) before the accessibility of the actual argument is validated. In some programs such an address will later be used to asynchronously post information back to an outer access mode. In such cases, the least privileged access mode that can perform the specified read or write operation must be copied from the corresponding page table entry and stored with the argument address.

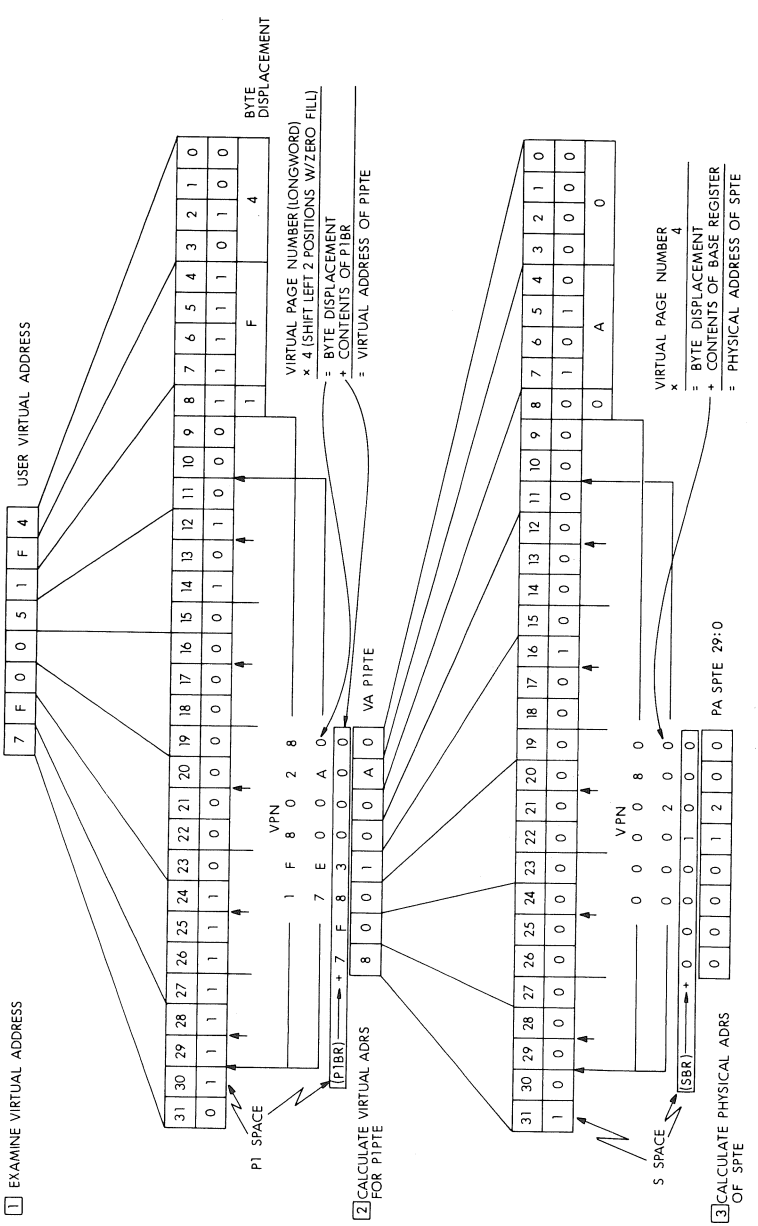
NOTE

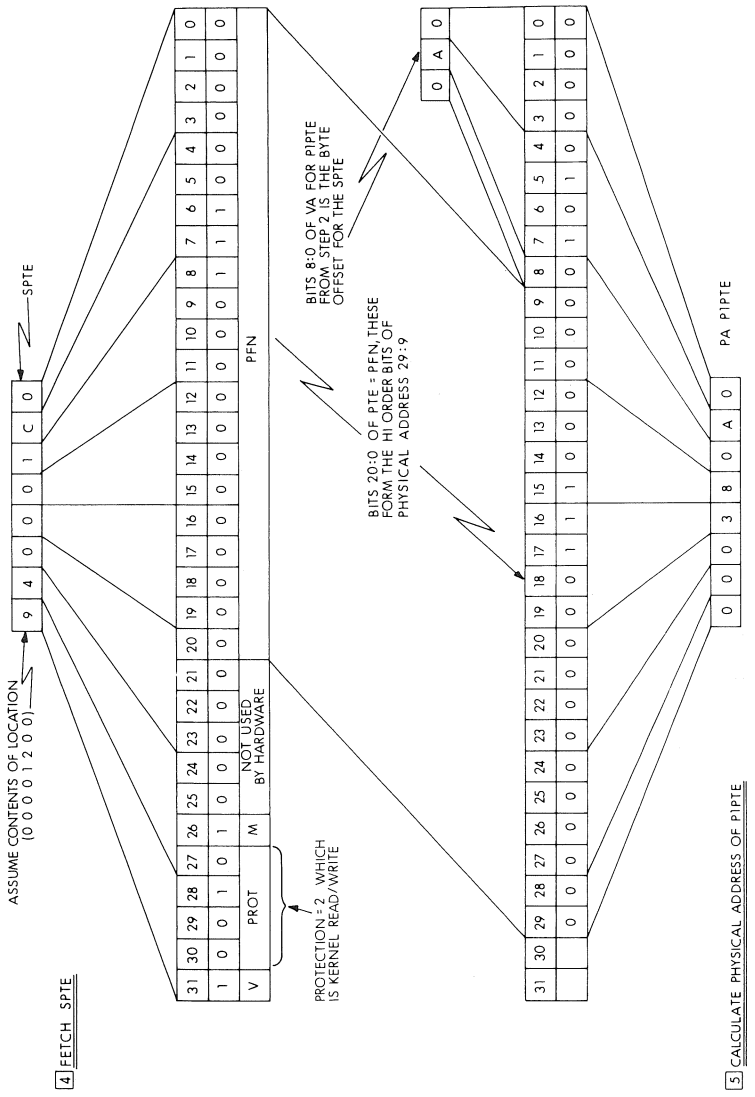
Using least privileged does not work properly when the data structure resides in pages with different protection and the first page has a lesser protection value than the others. When checking the accessibility of such a structure in the context of the serial execution of the process, the check will succeed, but later when the accessibility is checked again during the asynchronous posting of information, the check will fail. This situation is considered to be an operating system bug (may cause the generation of a bug check) and merely causes no information to be posted.

2. The synchronous validation of argument addresses (i.e., as the result of serial program execution) must be explicitly coded using Probe instructions specifying an access mode of zero (i.e., cause maximization to previous access mode).
3. The asynchronous validation of argument addresses (i.e., as the result of software interrupts) must be explicitly coded using Probe instructions specifying the least privileged access mode stored when the argument address was saved (see 1) and with a previous access mode field equal to or greater than that of the current mode field of PSL (i.e., cause maximization to least privileged access mode).
4. All arguments to be written must be PROBEWed before they are written (otherwise there would be a potential protection violation).
5. All arguments to be read must be PROBERed before they are read to defend against arguments mapped to I/O space and thereby causing an I/O side effect.
6. All addresses passed from an outer access mode to an inner access mode must be copied and validated before being passed as arguments in a call to a more inner access mode. This insures the integrity of intermediate modes.

This discussion is centered on the validation of argument addresses. There are other arguments that also deserve similar handling. Such arguments are typically address modifiers (e.g., a buffer length) and in most cases must also be copied to insure system integrity.

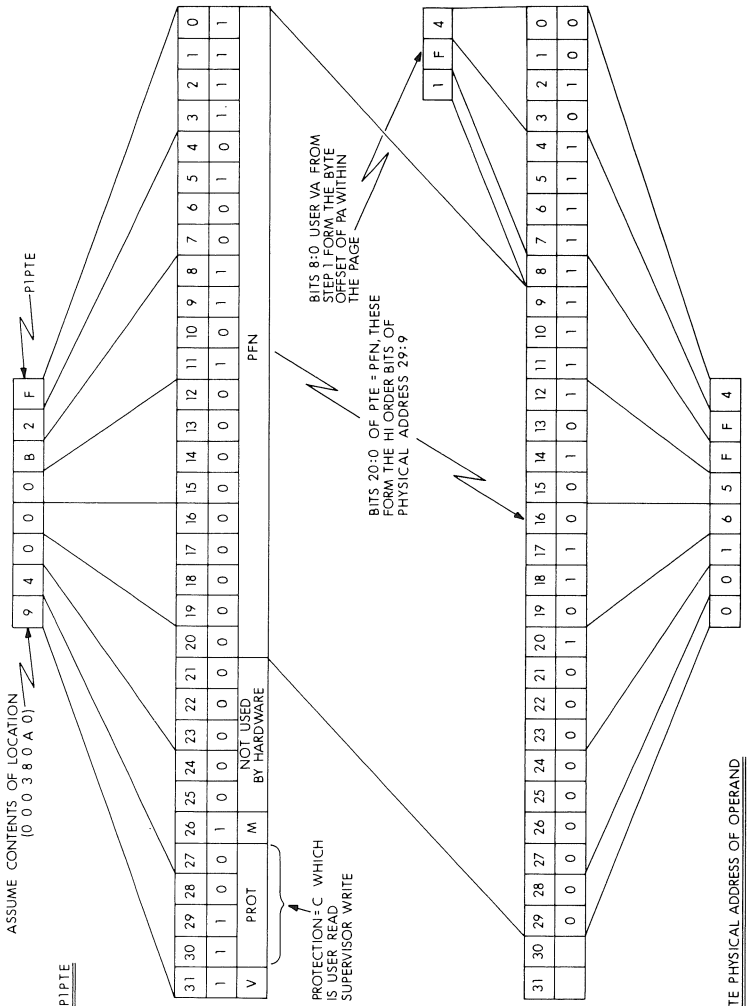
APPENDIX D
VIRTUAL TO PHYSICAL ADDRESS TRANSLATION





NOTE

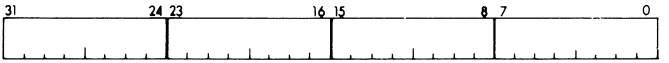
The address translation example shown determines a 30-bit physical address on a VAX-11/780 system. Address translation on the VAX-11/750 follows an identical pattern, except that the translation results in a 24-bit physical address.


ASSUME CONTENTS OF LOCATION
(0 0 0 3 8 0 A 0)
PTE
6 FETCH PTE
V
PROT
M
NOT USED
BY HARDWARE
PFN
PROTECTION=C WHICH
IS USER READ
SUPERVISOR WRITE
BITS 8:0 USER VA FROM
STEP 1 FORM THE BYTE
OFFSET OF PA WITHIN
THE PAGE
BITS 20:0 OF PTE = PFN, THESE
FORM THE HI ORDER BITS OF
PHYSICAL ADDRESS 29:9
7 CALCULATE PHYSICAL ADDRESS OF OPERAND

APPENDIX E
VAX-11/730 INTERNAL PROCESSOR REGISTERS

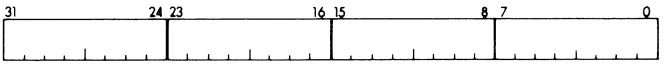
Kernel Stack Pointer Register (KSP)

Processor Address 00



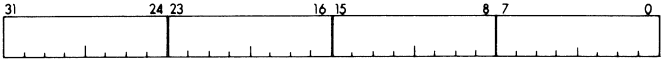
Executive Stack Pointer Register (ESP)

Processor Address 01



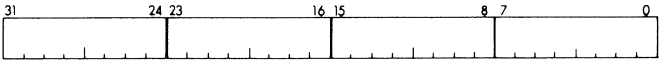
Supervisor Stack Pointer Register (SSP)

Processor Address 02



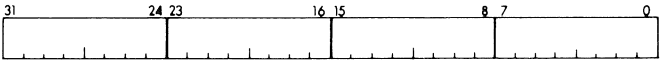
User Stack Pointer Register (USP)

Processor Address 03



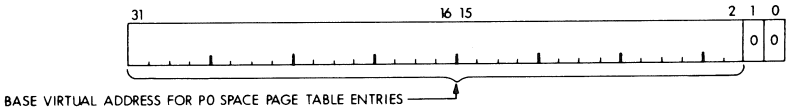
Interrupt Stack Pointer Register (ISP)

Processor Address 04



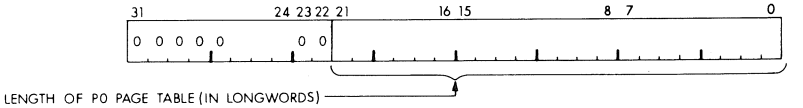
P0 Base Register (P0BR)

Processor Address 08



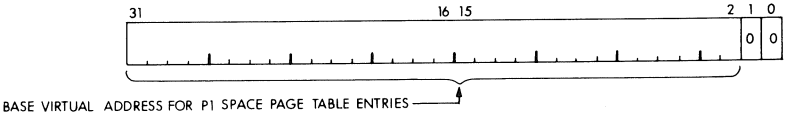
P0 Length Register (P0LR)

Processor Address 09



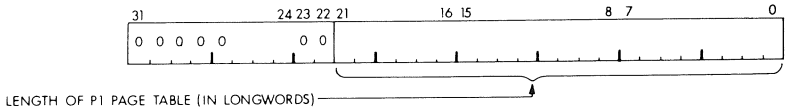
P1 Base Register (P1BR)

Processor Address 0A



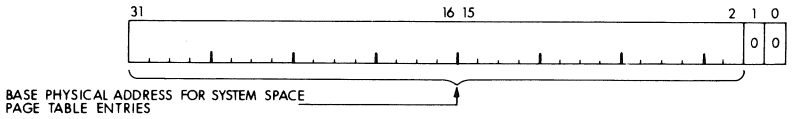
P1 Length Register (P1LR)

Processor Address 0B



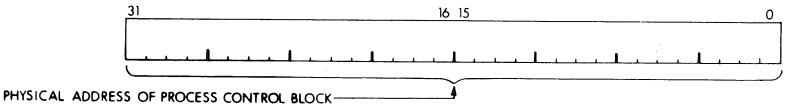
System Base Register (SBR)

Processor Address 0C



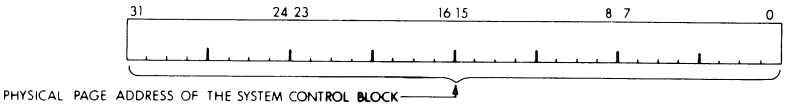
Process Control Block Base Register (PCBB)

Processor Address 10



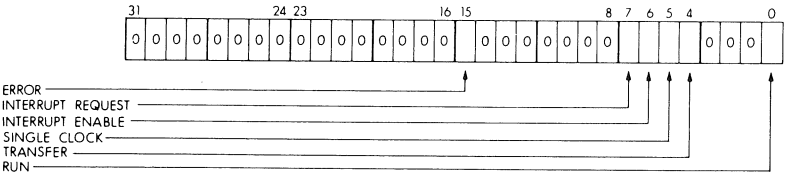
System Control Block Base Register (SCBB)

Processor Address 11



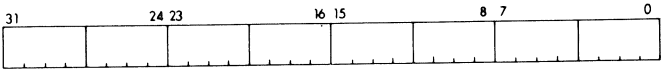
Interval Clock Control/Status Register (ICCS)

Processor Address 18



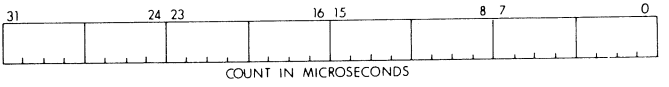
Next Interval Count Register (NICR)

Processor Address 19



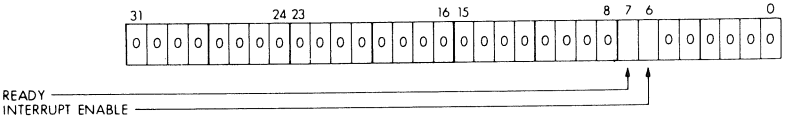
Interval Counter Register (ICR)

Processor Address 1A



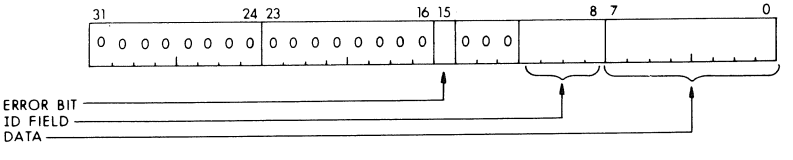
Console Receive Control/Status Register (RXCS)

Processor Address 20



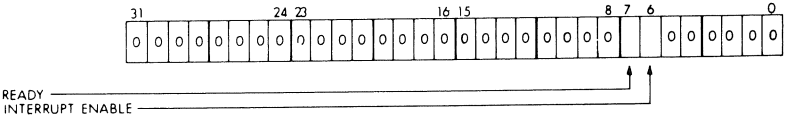
Console Receive Data Buffer Register (RXDB)

Processor Address 21



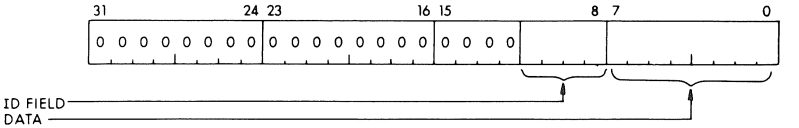
Console Transmit Control/Status Register (TXCS)

Processor Address 22



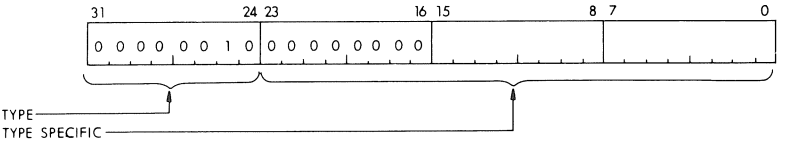
Console Transmit Data Buffer Register (TXDB)

Processor Address 23



System Identification Register (SID)

Processor Address 3E

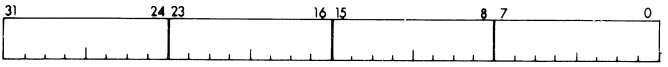


APPENDIX F

VAX-11/750 INTERNAL PROCESSOR REGISTERS

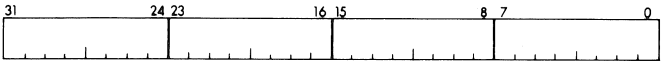
Kernel Stack Pointer Register (KSP)

Processor Address 00



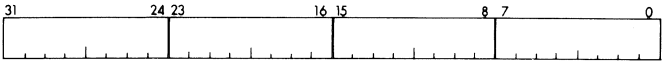
Executive Stack Pointer Register (ESP)

Processor Address 01



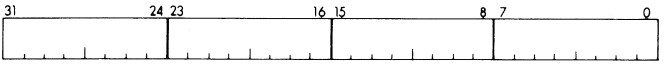
Supervisor Stack Pointer Register (SSP)

Processor Address 02



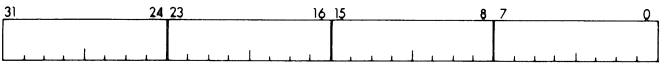
User Stack Pointer Register (USP)

Processor Address 03



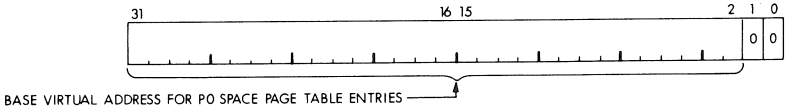
Interrupt Stack Pointer Register (ISP)

Processor Address 04



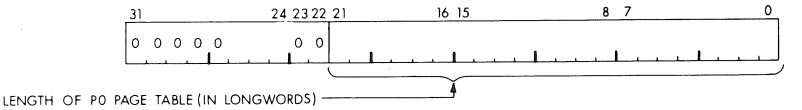
P0 Base Register (P0BR)

Processor Address 08



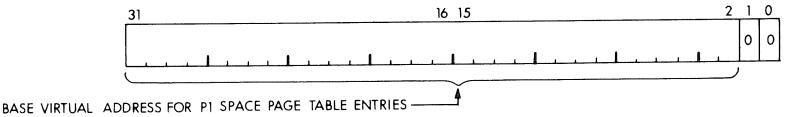
P0 Length Register (P0LR)

Processor Address 09



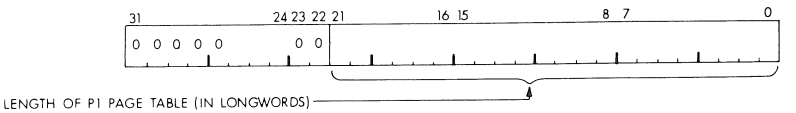
P1 Base Register (P1BR)

Processor Address 0A



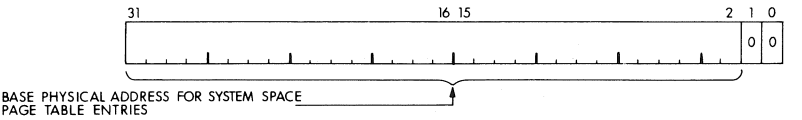
P1 Length Register (P1LR)

Processor Address 0B



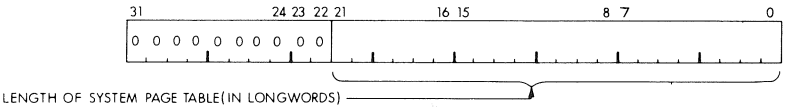
System Base Register (SBR)

Processor Address 0C



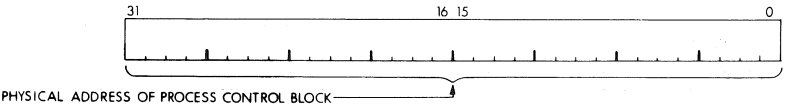
System Length Register (SLR)

Processor Address 0D



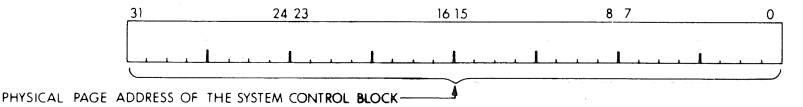
Process Control Block Base Register (PCBB)

Processor Address 10



System Control Block Base Register (SCBB)

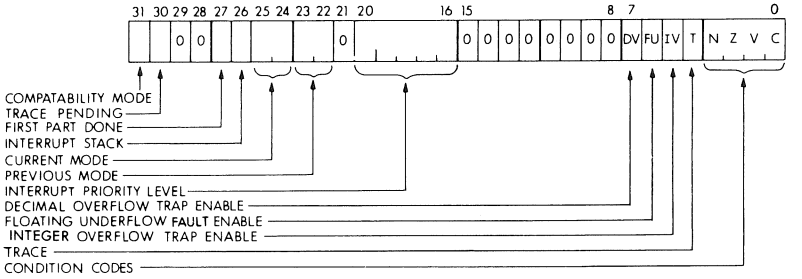
Processor Address 11



Processor Status Longword Register (PSL)

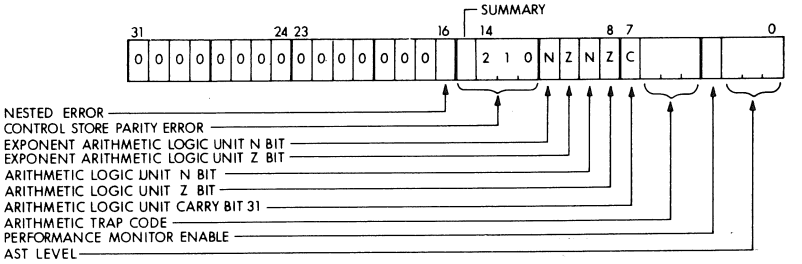
Interrupt Priority Level (IPL) Bits <16:20>

Processor Address 12



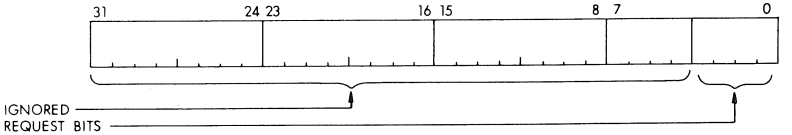
Processor Address 13 (Accesses AST level bits <2:0>) (ASTR)

Processor Address 3D (Accesses Performance Monitor Enable bit <3>) (PMR)



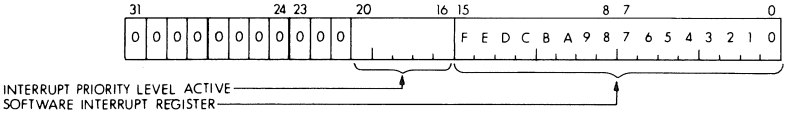
Software Interrupt Request Register (SIRR)

Processor Address 14



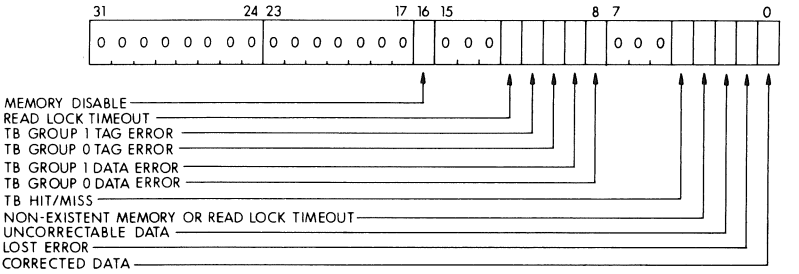
Software Interrupt Register (SISR)

Processor Address 15



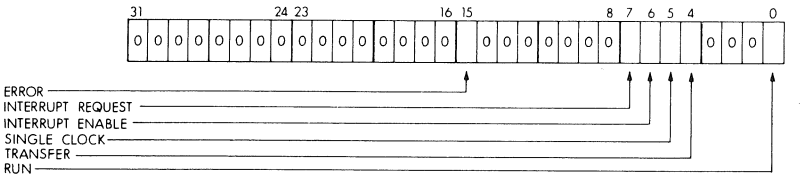
Machine Check Status Register (MCSR)

Processor Address 17



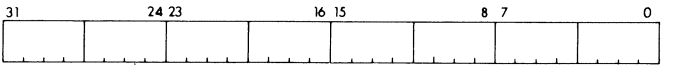
Interval Clock Control/Status Register (ICCS)

Processor Address 18



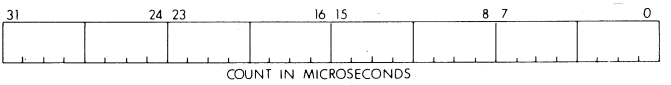
Next Interval Count Register (NICR)

Processor Address 19



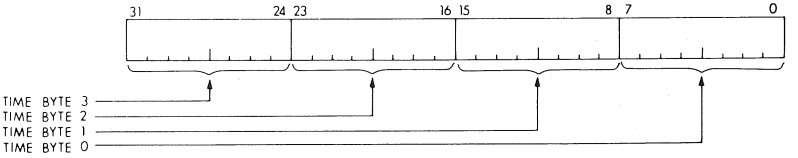
Interval Counter Register (ICR)

Processor Address 1A



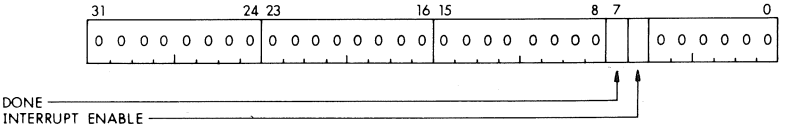
Time of Day Register (TODR)

Processor Address 1B



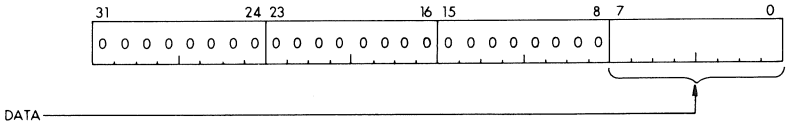
Console Storage Receive Status Register (CSRS)

Processor Address 1C



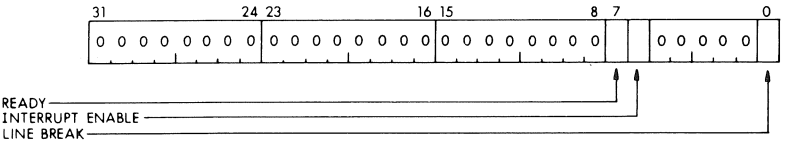
Console Storage Receive Data Register (CSRD)

Processor Address 1D



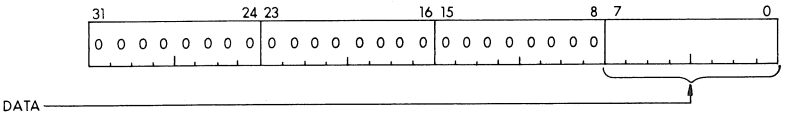
Console Storage Transmit Status Register (CSTS)

Processor Address 1E



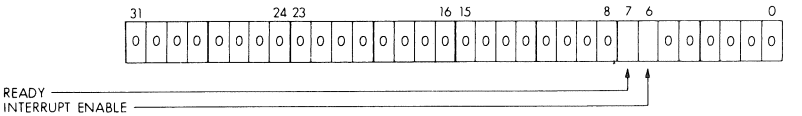
Console Storage Transit Data Register (CSTD)

Processor Address 1F



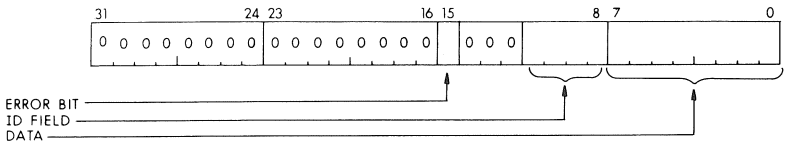
Console Receive Control/Status Register (RXCS)

Processor Address 20



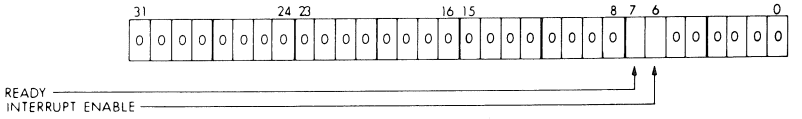
Console Receive Data Buffer Register (RXDB)

Processor Address 21



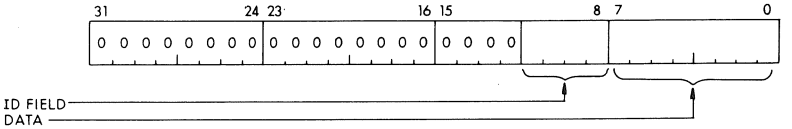
Console Transmit Control/Status Register (TXCS)

Processor Address 22



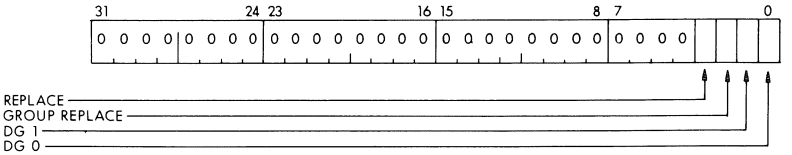
Console Transmit Data Buffer Register (TXDB)

Processor Address 23



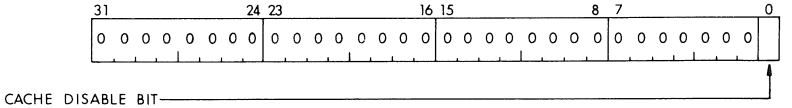
Translation Buffer Disable Register (TBDR)

Processor Address 24



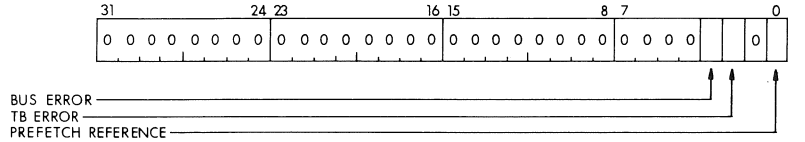
Cache Disable Register (CADR)

Processor Address 25



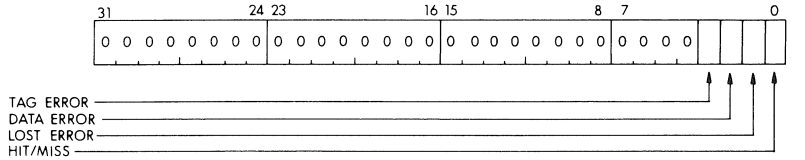
Machine Check Error Summary Register (MCESR)

Processor Address 26



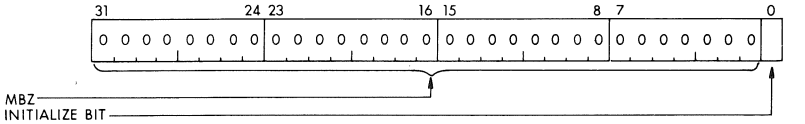
Cache Error Register (CAER)

Processor Address 27



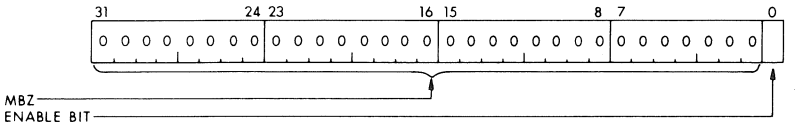
Initialize UNIBUS Register

Processor Address 37



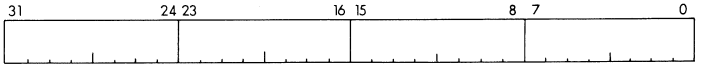
Memory Management Enable Register (MME)

Processor Address 38



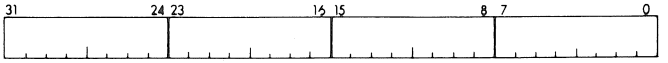
Translation Buffer Invalidate All Register (TBIA)

Processor Address 39



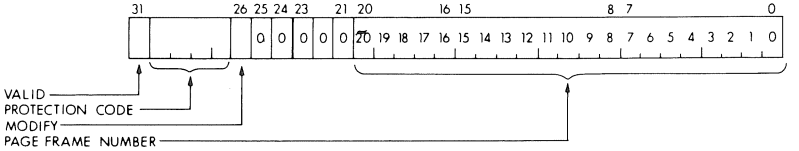
Translation Buffer Invalidate Single Register (TBIS)

Processor Address 3A



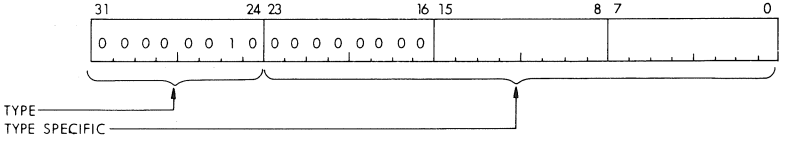
Translation Buffer Register (TB)

Processor Address 3B



System Identification Register (SID)

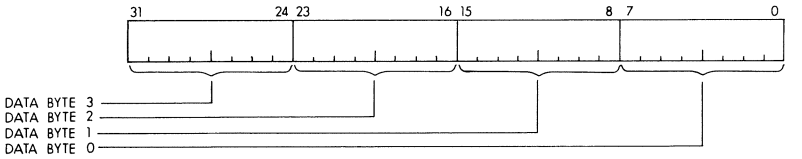
Processor Address 3E



APPENDIX G
VAX-11/780 INTERNAL DATA (ID) BUS REGISTERS

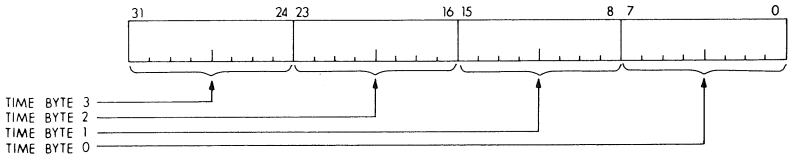
Instruction Buffer Register

ID Address 00
Processor Address —



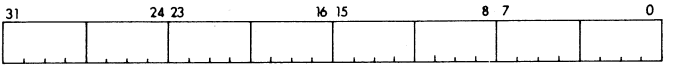
Time of Day Register (TODR)

ID Address 01
Processor Address 1B



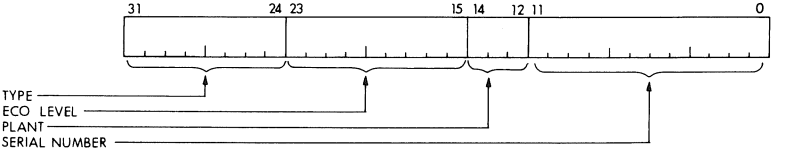
Reserved Register

ID Address 02
Processor Address —



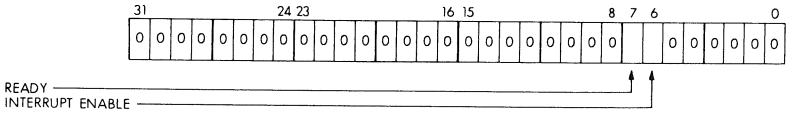
System Identification Register (SID)

ID Address 03
Processor Address 3E



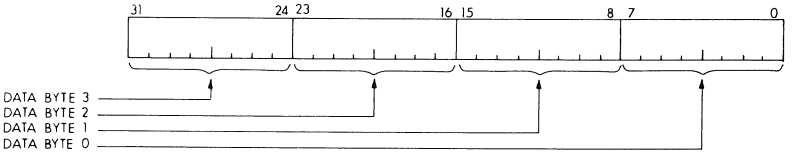
Console Receive Control/Status Register (RXCS)

ID Address 04
Processor Address 20



Console Receive Data Buffer Register (RXDB)

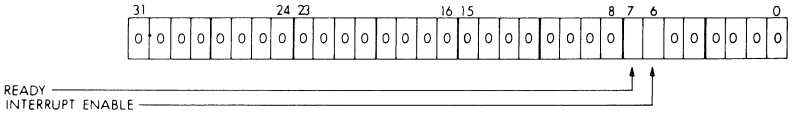
ID Address 05
Processor Address 21



As defined in the software, bits <7:0> define the data field, bits <11:8> define the ID field and bit <15> is the error bit. However, the hardware is not restricted to this convention.

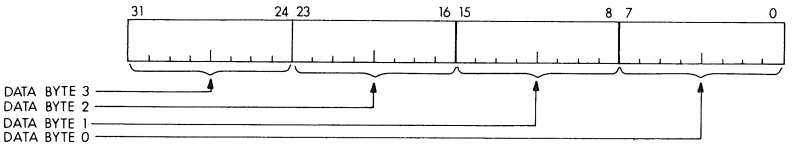
Console Transmit Control/Status Register (TXCS)

ID Address 06
Processor Address 22



Console Transmit Data Buffer Register (TXDB)

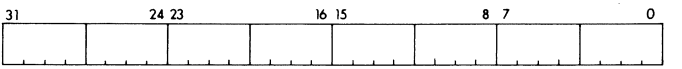
ID Address 07
Processor Address 23



As defined in the software, bits <7:0> define the data field, and bits <11:8> define the ID field. However, the hardware is not restricted to this convention.

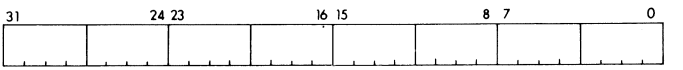
DQ Register

ID Address 08
Processor Address —



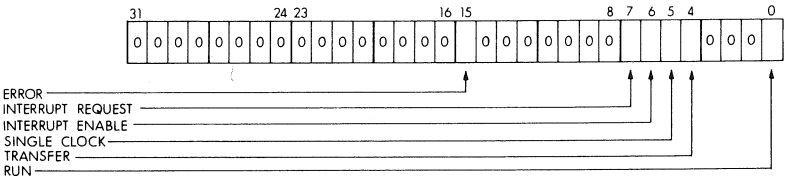
Next Interval Count Register (NICR)

ID Address 09
Processor Address 19



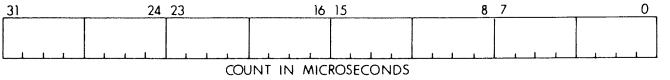
Interval Clock Control/Status Register (ICCS)

ID Address 0A
Processor Address 18



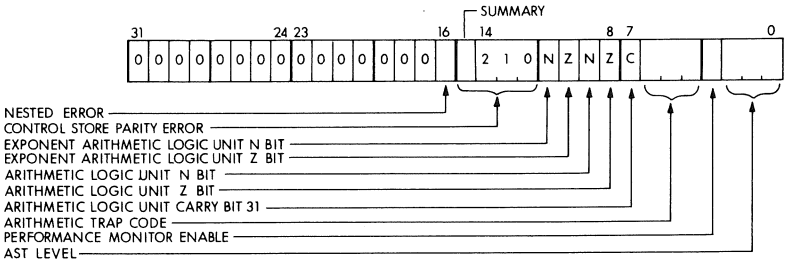
Interval Counter Register (ICR)

ID Address 0B
Processor Address 1A



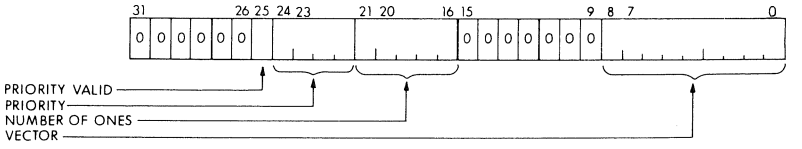
ID Address 0C
Processor Address 13 (Accesses AST level bits<2:0>(ASTR)

Processor Address 3D (Accesses performance monitor Enable bit <3>) (PMR)



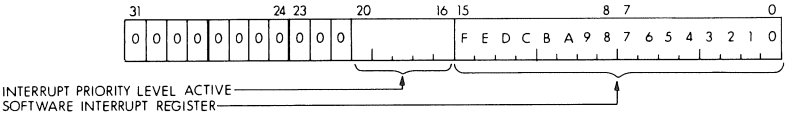
Vector Register

ID Address 0D
Processor Address —



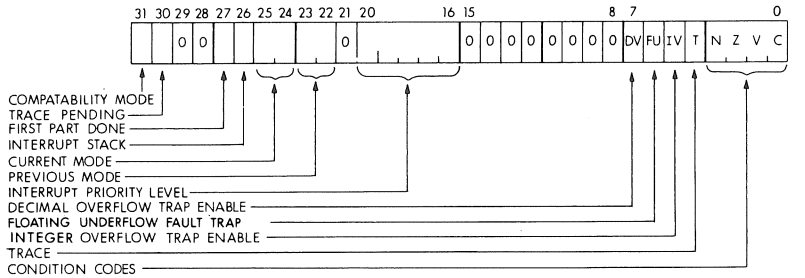
Software Interrupt Summary Register (SISR)

ID Address 0E
Processor Address 15



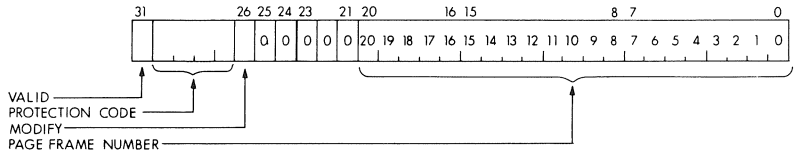
Processor Status Longword Register (PSL)

ID Address 0F
Processor Address 12



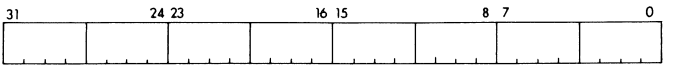
Translation Buffer Data Register (TB)

ID Address 10
Processor Address —



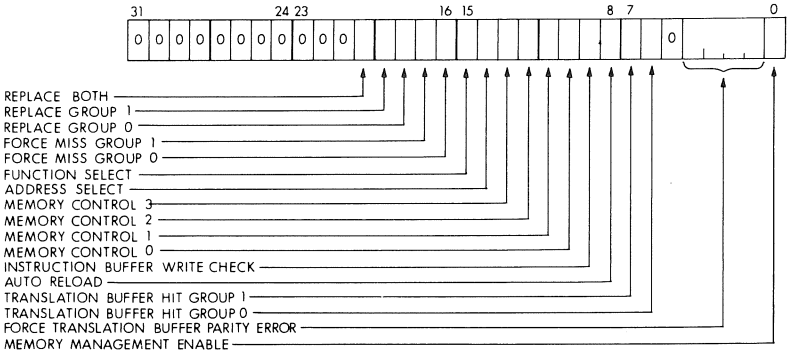
Reserved Register

ID Address 11
Processor Address —



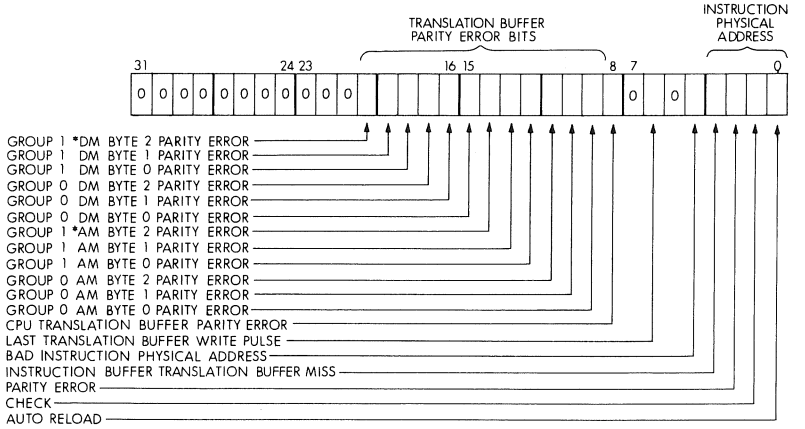
Translation Buffer Control Register 0

ID Address 12
Processor Address —



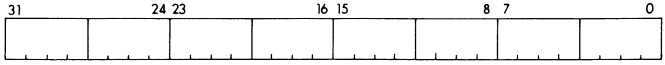
Translation Buffer Control Register 1

ID Address 13
Processor Address —



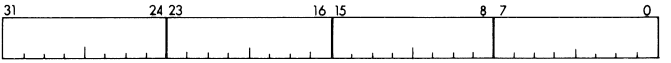
Accelerator Control Register 0

ID Address 14
Processor Address —



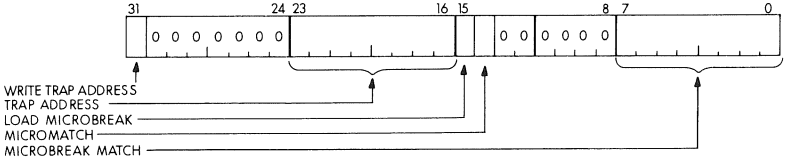
Accelerator Control Register 1

ID Address 15
Processor Address —



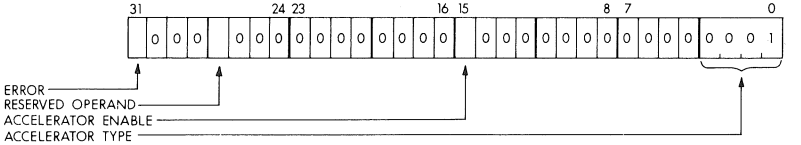
Accelerator Maintenance Register

ID Address 16
Processor Address —



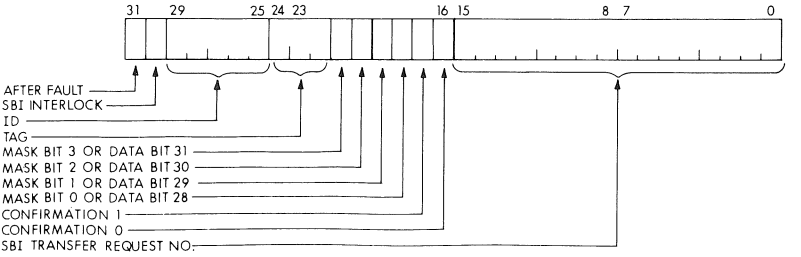
Accelerator Control/Status Register (ACCS)

ID Address 17
Processor Address 28



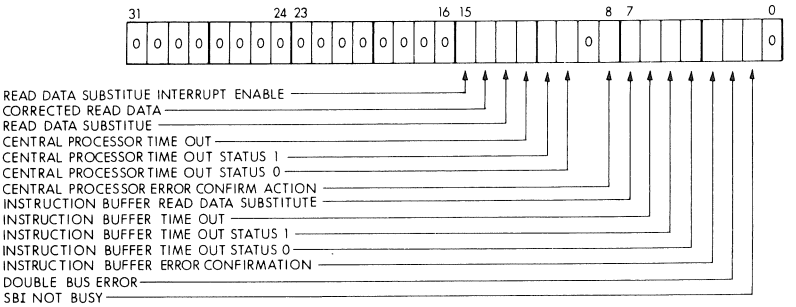
SBI Silo Register

ID Address 18
Processor Address 31



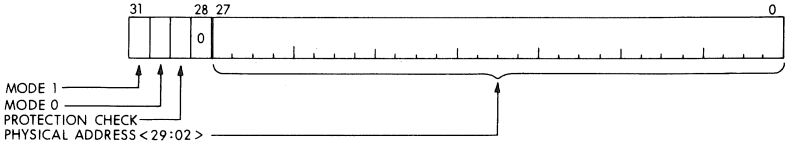
SBI Silo Register

ID Address 19
Processor Address 34



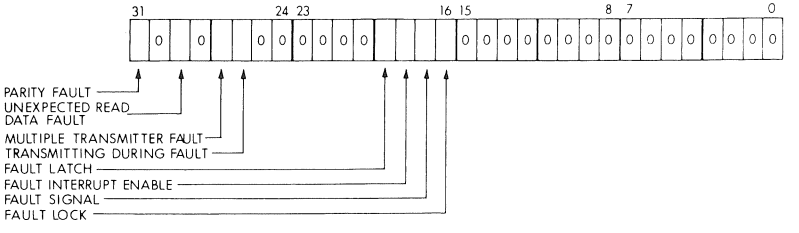
SBI Time Out Address Register

ID Address 1A
Processor Address 35



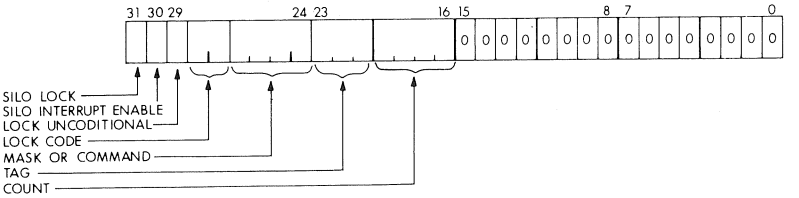
SBI Fault Signal Register

ID Address 1B
Processor Address 30



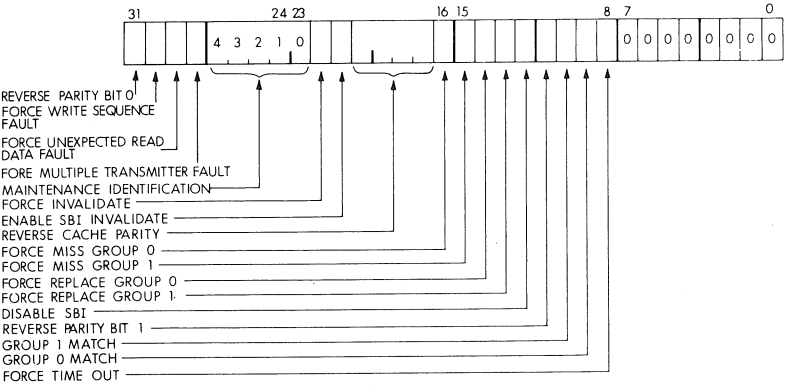
SBI Silo Comparator Register

ID Address 1C
Processor Address 32



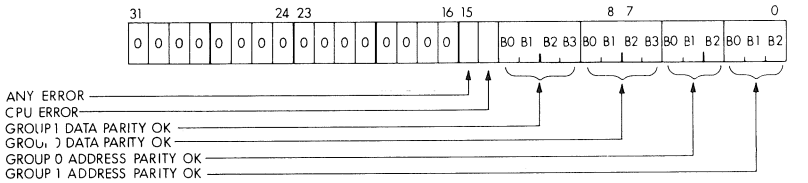
SBI/Cache Maintenance Register

ID Address 1D
Processor Address 33



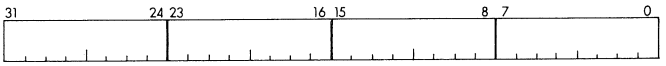
Cache Parity Register

ID Address 1E
Processor Address —



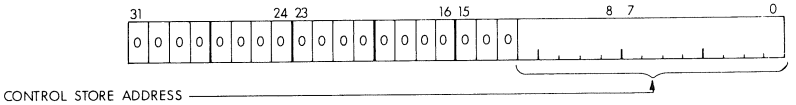
Reserved Register

ID Address 1F
Processor Address —



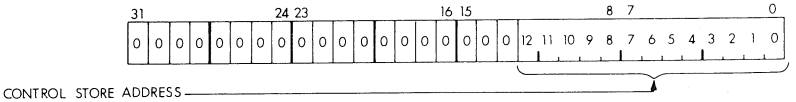
Micro Stack Register

ID Address 20
Processor Address —



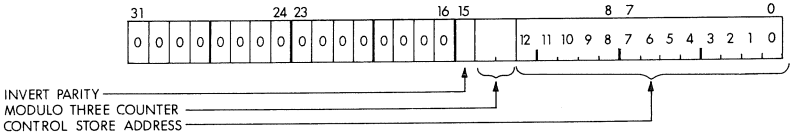
Micro Match Register

ID Address 21
Processor Address 3C



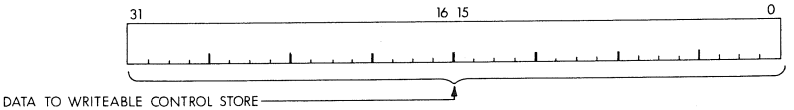
Writable Control Store Address Register

ID Address 22
Processor Address 2C



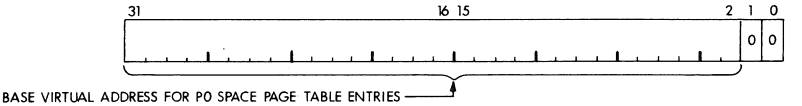
Writable Control Store Data Register

ID Address 23
Processor Address 2D



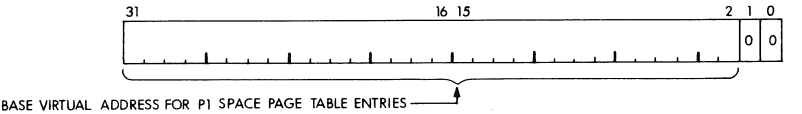
P0 Base Register (P0BR)

ID Address 24
Processor Address 08



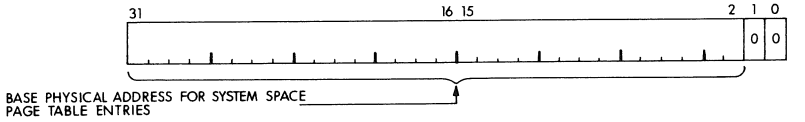
P1 Base Register (P1BR)

ID Address 25
Processor Address 0A



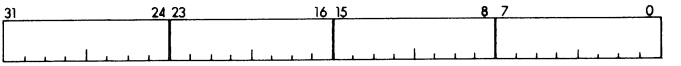
System Base Register (SBR)

ID Address 26
Processor Address 0C



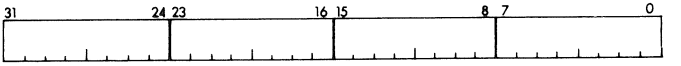
Reserved Register

ID Address 27
Processor Address —



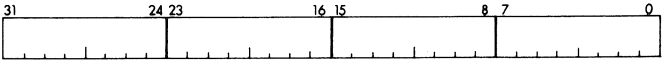
Kernal Stack Pointer Register (KSP)

ID Address 28
Processor Address 00



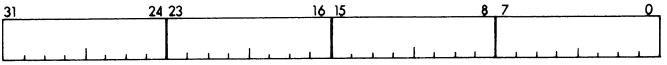
Executive Stack Pointer Register (ESP)

ID Address 29
Processor Address 01



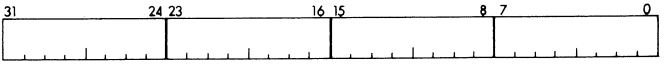
Supervisor Stack Pointer Register (SSP)

ID Address 2A
Processor Address 02



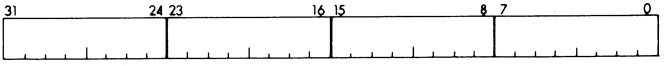
User Stack Pointer Register (USP)

ID Address 2B
Processor Address 03



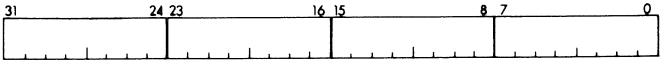
Interrupt Stack Pointer Register (ISP)

ID Address 2C
Processor Address 04



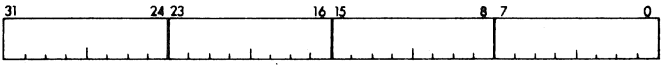
First Part Done Address Register

ID Address 2D
Processor Address —



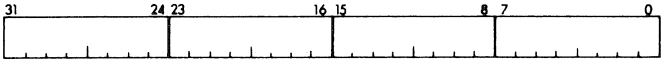
D Save Register

ID Address 2E
Processor Address —



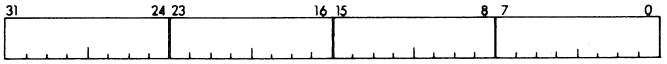
Q Save Register

ID Address 2F
Processor Address —



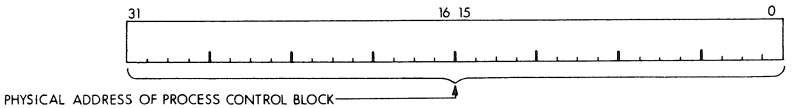
Temp 0 to Temp 9 Registers

ID Address (30 to 39)
Processor Address —



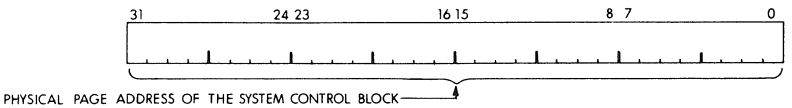
Process Control Block Base Register (PCBB)

ID Address 3A
Processor Address 10



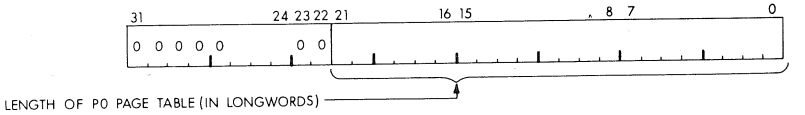
System Control Block Base Register (SCBB)

ID Address 3B
Processor Address 11



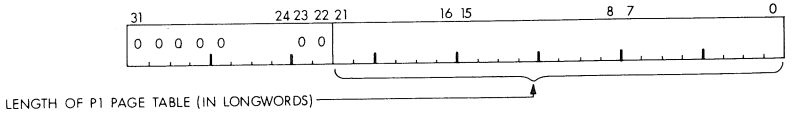
P0 Length Register (P0LR)

ID Address 3C
Processor Address 09



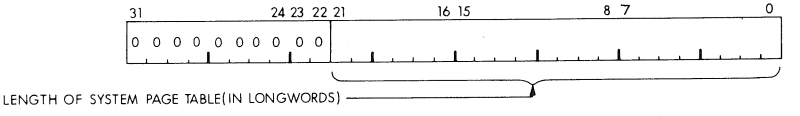
P1 Length Register (P1LR)

ID Address 3D
Processor Address 0B



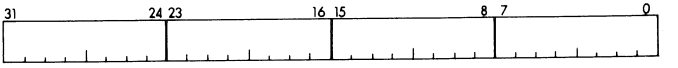
System Length Register (SLR)

ID Address 3E
Processor Address 0D



Reserved Register

ID Address 3F
Processor Address —



APPENDIX H
OPERAND SPECIFIER NOTATION

OPERAND SPECIFIERS

Operand specifiers are described in the following way:

<name><access type><data type>

where:

Name is a suggestive name for the operand in the context of the instruction. The name is often abbreviated.

Access type is a letter denoting the operand specifier access type:

a	Calculate the effective address of the specified operand. Address is returned in a longword which is the actual instruction operand. Context of address calculation is given by <data type>.
b	No operand reference. Operand specifier is a branch displacement. Size of branch displacement is given by <data type>.
m	Operand is read, potentially modified and written. Note that this is NOT an indivisible memory operation. Also note that if the operand is not actually modified, it may not be written back. However, modify type operands are always checked for both read and write accessibility.
r	Operand is read-only.
v	Calculate the effective address of the specified operand. If the effective address is in memory, the address is returned in a longword which is the actual instruction operand. Context of address calculation is given by <data type>. If the effective address is Rn, then the operand actually appears in R[n], or in R[n+1]'R[n].
w	Operand is written only.

Data type is a letter denoting the data type of the operand:

b	byte
d	D_floating
f	F_floating

g	G_floating
h	H_floating
l	longword
q	quadword
w	word
x	first data type specified by instruction
y	second data type specified by instruction

OPERATION DESCRIPTION NOTATION

The operation of each instruction is given as a sequence of control and assigment statements in an ALGOL-like syntax. No attempt is made to define the syntax formally; it is assumed to be familiar to the reader.

+	addition
−	subtraction, unary minus
*	multiplication
/	division (quotient only)
**	exponentiation
'	concatenation
←	is replaced by
=	is defined as
Rn or R[n]	contents of register Rn
PC, SP, FP, or AP	the contents of register R15, R14, R13, or R12 respectively
PSW	the contents of the processor status word
PSL	the contents of the processor status longword
(x)	contents of memory location whose address is x
(x)+	contents of memory location whose address is x; x incremented by the size of operand referenced at x
−(x)	x decremented by size of operand to be referenced at x; contents of memory location whose address is x
<x:y>	a modifier which delimits an extent from bit position x to bit position y inclusive
<x1,x2,...,xn>	a modifier which enumerates bits x1,x2...,xn
x...y	x through y inclusive

{ }	arithmetic parentheses used to indicate precedence
AND	logical AND
OR	logical OR
XOR	logical XOR
NOT	logical (1's) complement
LSS	less than signed
LSSU	less than unsigned
LEQ	less than or equal signed
LEQU	less than or equal unsigned
EQL	equal signed
EQLU	equal unsigned
NEQ	not equal signed
NEQU	not equal unsigned
GEQ	greater than or equal signed
GEQU	greater than or equal unsigned
GTR	greater than signed
GTRU	greater than unsigned
SEXT (x)	is sign-extended to size of operand needed
ZEXT (x)	is zero-extended to size of operand needed
REM (x, y)	remainder of x divided by y
MINU (x, y)	minimum unsigned of x and y

The following conventions are used:

- Other than that caused by ( ) +, or −( ), and the advancement of PC, only operands or portions of operands appearing on the left side of assignment statements are affected.
- No operator precedence is assumed, other than that replacement (←) has the lowest precedence. Precedence is indicated explicitly by { }.
- All arithmetic, logical, and relational operators are defined in the context of their operand. For example "+" applied to floating operands means a floating add while "+" applied to byte operands is an integer byte add. Similarly, "LSS" is a floating comparison when applied to floating operands while "LSS" is an integer byte comparison when applied to byte operands.

- Instruction operands are evaluated according to the operand specifier conventions. The order in which operands appear in the instruction description has no effect on the order of evaluation.
- Condition codes are in general affected on the value of actual stored results, not on "true" results (which might be generated internally to greater precision). Thus, for example, two positive integers can be added together and the sum stored, because of overflow, as a negative value. The condition codes will indicate a negative value even though the "true" result is clearly positive.

APPENDIX I
I/O SPACE RESTRICTIONS

A subset of native mode instructions is not used to reference I/O space. The reasons are:

1. String instructions are restartable via PSL<FPD>.
2. The PC, SP, or PCBB cannot point to I/O space.
3. I/O space does not support operand types of quad, F_floating, D_floating, G_floating, H_floating, field, or queue; nor can the position, size, length, or base of them be from I/O space.
4. The instruction may be interruptable because it is potentially a slow instruction in some implementations.
5. Only instructions with a maximum of one modify or write destination can be used. The destination must be the last operand.

For any memory reference to I/O space, the programmer must use an instruction from the following lists and must ensure that no interrupts or faults will occur, including page faults, after the first I/O space reference. To ensure no interrupts, the programmer must avoid operand specifier addressing modes 9, 11, 13, and 15, and these modes indexed. (Symbolically, these are @(Rn)+, @B↑D(Rn), @W↑D(Rn), and @L↑D(Rn), and these indexed.) The hardware may allow interrupts for these modes in order to minimize interrupt latency. For the instructions in the following lists, the hardware ensures that no other interrupts will occur after the first I/O space access.

Since these instructions are not interruptable after I/O space accesses (except for the addressing modes above), their execution will extend the interrupt latency. The programmer should make some effort to keep them short by minimizing the number of memory references. Use R0 through R13 instead, for example.

Instructions for which any explicit operands can be in I/O space are:

MOV{B,W,L}
CLR{B,W,L}
MCOM{B,W,L}
CVT{BW,BL,WB,WL,LB,LW}
TST{B,W,L}
ADD{B,W,L}3
INC{B,W,L}
SUB{B,W,L}2
DEC{B,W,L}
BIT{B,W,L}
PUSHL
MNEG{B,W,L}
MOVZ{BW,BL,WL}
CMP{B,W,L}
ADD{B,W,L}2
ADAWI
ADWC
SUB{B,W,L}3
SBWC
BIS{B,W,L}2

BIS{B,W,L}3
BIC{B,W,L}3
XOR{B,W,L}3
MOVAQ
PUSHAQ
MOVPSL
BICPSW
PROBE{R,W}
BIC{B,W,L}2
XOR{B,W,L}2
MOVA{B,W,L}
PUSHA{B,W,L}
CASE{B,W,L}
BISPSW
CHM{K,E,S,U}
MTPR, MFPR

Instructions for which all operands except the branch displacement can be in I/O space are:

BLB {S,C}

Instructions for which some operands can be in I/O space are:

XFC (depending on implementation)
REMQUE addr (destination)

In spite of the above rules, it is possible for a specific hardware implementation to execute macro code from the I/O space and/or to allow the stack or PCB to be in I/O space. This might be used as part of the bootstrap process, for example. If this is done, then it is valid for software to transfer to this code.

APPENDIX J
TECHNICAL SPECIFICATIONS FOR VAX-11/730 PROCESSOR

Processor Type	Microprogrammed 24-bit control store word
Micro-control store instruction time	270 nanoseconds
Control store size	16 K words (24-bit words), read-only memory
Internal data path	32 bits
Instruction buffer size	1-longword lookahead

CPU Address Translation Buffer

Size	128 address translations

CPU Clocks

Programmable Interval Timer
Time-of-Year Clock

VAX Instruction Set

16 32-bit registers
304 basic operations
56 optional instructions implemented with the FP730
32 priority interrupt levels
PDP-11 compatibility mode instructions

Multiple data types	Integer, floating point, packed decimal, character string, variable bit fields, numeric strings, queues
Addressing modes	9

Other Standard Features

- Power-fail automatic restart
- Hardware bootstrap load
- Three serial line ASCII console interfaces
- DMF32 communications board (except on box product)
- Dual TU58 cartridge tape drives
- Microverify diagnostic run automatically on power-up
- Extensive reliability and maintainability features
- Virtual console commands from console terminal
- Integrated Disk Controller (IDC)

Main Memory

Virtual address space	4 billion bytes
Physical address space	16 megabytes
Address lines	24 bits
Physical expansion	5 MB in 1 MB increments
Parity	7-bit error correcting code (ECC) per 32-bit longword
Technology	Bit-slice and Programmed Array Logic (PAL)
Cycle times	810 nanoseconds R/W

I/O UNIBUS Adapter

Max. UNIBUS I/O rate	1.5 MB/s
Interrupts	Directly vectored

Mechanical

	Box	Dual RL02	R80/RL02
Weight (max.)	45.4 kg (100 lbs)	227.0 kg (500 lbs)	249.7 kg (550 lbs)
Height	26.6 cm (10.5 in)	106.2 cm (41.8 in)	106.2 cm (41.8 in)
Width	47.0 cm (18.5 in)	54.1 cm (21.3 in)	54.1 cm (21.3 in)
Depth	66 cm (26.6 in)	80.0 cm (31.5 in)	80.0 cm (31.5 in)

Electrical Power Requirements

NOTE

Specifications for 60 Hz systems are listed first; 50 Hz systems are in parenthesis.

	Dual RL02 System	RL02/R80 System
Maximum		
AC line voltage tolerance	90-128V (180-256V)	90-128V (180-256V)
Frequency tolerance	47-63Hz (47-63Hz)	60 Hz ±1 Hz (50 Hz ± 1 Hz)
Phases	1 (1)	1 (1)

	Box	Dual RL02	RL02/R80
Steady state current, 90 Vrms	8	10	15
Steady state current, 180 Vrms	4	5	7

Surge current, 90 Vrms	15	20	32
Surge current, 180 Vrms	7.5	10	12
Plug Type	5-15P (6-15P)	5-20P (6-15P)	L5-30P (6-15P)
AC cable length	3.05 m (10 ft)	4.57 m (15 ft)	4.57 m (15 ft)
Maximum heat dissipation (est.)	403.2 kcal/hr (1603 Btu/hr)	756.0 kcal/hr (2694 Btu/hr)	1260.0 kcal/hr (4109 Btu/hr)
Maximum AC power consumption	470 W	790 W	1205 W
Nominal voltage:	120 (220-240)	120 (220-240)	120 (220-240)

Environment

Operating:

Temperature	10° to 40° C (50° to 104° F)
Relative humidity	10 to 90%
Maximum altitude	2.4 km (8000 ft.)

Storage:

Temperature	−40° to 66° C (−40° to 151° F)
Relative humidity	10 to 95%

Specification for TU58 in VAX-11/730 Console

Capacity per cartridge	256 KB (512 records by 512 bytes)
Capacity per track	128 KB

Tape length	140 feet
Data transfer rate	19.2 KB/s
Average access time	9.3 seconds
Maximum access time	28 seconds
Read/write tape speed	30 in/s
Search tape speed	60 in/s
Bit density	800 bpi
Number of passes per cartridge to read/write cartridge	4 (2 tracks interleaved)
Best case read/write time for entire cartridge	10 minutes

APPENDIX K
TECHNICAL SPECIFICATIONS FOR VAX-11/750 PROCESSOR

NOTE

Specifications with an * are for the VAX-11/750 processor only.

Processor Type	Microprogrammed 80-bit control store word
Micro-control store instruction time	320 nanoseconds
Control store size	6K words (80-bit words), read-only memory
Internal data path	32 bits
Maximum system I/O rate	5 MB/s
Instruction buffer size	8-byte lookahead

CPU Cache Memory

Size	4 KB direct mapped
Effective main memory cycle time	400 nanoseconds/32 bits
Typical hit ratio	90%
Typical cache cycle time	320 nanoseconds

CPU Address Translation Buffer

Size	512 address translations
Typical hit ratio	98-99%

CPU Clocks

Realtime clock	Crystal controlled, 01% accuracy 1 microsecond resolution
Time-of-year clock	Includes recharging battery backup for over 100 hours

VAX Instruction Set

16 32-bit registers
248 basic operations
56 optional instructions
32 priority interrupt levels
PDP-11 compatibility mode instructions

Multiple data types	Integer, floating point, packed decimal, character string, variable bit fields, numeric strings, queues
Addressing modes	9

Other Standard Features

- Power-fail automatic restart
- Hardware bootstrap load for up to four different devices
- Single serial line ASCII console interface
- Eight-line communications multiplexer (DZ11-A)
- TU58 cartridge tape unit
- Microverify diagnostic run automatically on power-up
- Extensive reliability and maintainability features
- Virtual console commands from DECwriter terminal

Main Memory

Virtual address space	4 billion bytes
Physical address lines	16 megabytes (24 bits)

Physical expansion	8 MB in 1 MB increments
Parity	7-bit error correcting code (ECC) per 32-bit longword
Technology	64 K-bit MOS RAMs
Cycle times	800 nanoseconds per 32-bit read 640 nanoseconds per 32-bit write
Power failure protection	Optional battery backup

I/O UNIBUS Adapter

Max. UNIBUS I/O rate	1.5 MB/s through buffered data paths
Buffered data paths	3 total, 4-byte buffer in each
Interrupts	Directly vectored

VAX-11/750 OPTIONS

MASSBUS Adapters (up to 3 per VAX-11/750)

Maximum bandwidth	2.0 MB/s per MBA (to a system maximum of 5.0 MB/s)
Devices	Up to eight high speed disks or tapes
Buffer size	32 bytes per MBA

UNIBUS Adapter (1 with 2 or less optional MASSBUS adapters)

See UNIBUS adapter specifications on previous page.

User Control Store

Size	1 K word (80-bit words)

Technology	Writeable memory (RAM), 70 nanosecond access time

Memory Battery Backup

Minimum backup time	10 minutes with 8 MB of memory

Mechanical (VAX-11/750-BA/BB cabinet as installed)*

Weight	182 kg (400 lbs.) Max.
Height	106.2 cm (41.8 in.)
Width	73.7 cm (29.0 in.)
Depth	76.3 cm (30.0 in.)

Electrical Power Requirements*

Maximum	**VAX-11/750-BA**	**VAX-11/750-BB**
AC line voltage tolerance	90-128 V	180-256 V
Frequency tolerance	47-63 Hz	47-63 Hz
Phases	1	1
Steady state current	30 @ 90 Vrms	15 @ 180 Vrms
Surge current	100A	100A
Surge duration (exponential decay)	8 cycles	8 cycles
Plug Type	L5-30P	(Varies by country)
AC cable length	3 m (9.84 ft.)	
Maximum heat dissipation	1460 kcal/hr (5800 BTU/hr)	
Maximum AC power consumption	1700 watts	
Nominal voltage:	120 volts at 18 amps	

Typical AC Utilization by Option

NOTE

UNIBUS peripheral controllers plugged into the remaining eight slots in the VAX-11/750 DD11-DK backplane must also be included for an accurate total.

Module Description	AC Utilization (Watts)	Heat Dissipation (BTU/Hour)
Basic CPU, 1 MB memory, TU58, DZ11-A	505	1723
Each additional 1 MB memory card	14	47
Memory battery backup	40	137
Remote diagnosis option	96	328
Second UNIBUS adapter	50	170
Each MASSBUS adapter	125	407

Environment

NOTE

Environmental and power figures are given for the VAX-11/750-AA/AB only. Consult DIGITAL Field Service for the recommended value for an entire system.

Operating:

Temperature 10° to 40° C (50° to 104° F)

Relative humidity	10 to 90%
Maximum altitude	2.4 km (8000 ft.)

Storage:

Temperature	−40° to 66° C (−40° to 151° F)
Relative humidity	10 to 95%

Expansion Slots

UNIBUS	2 quad slots 6 hex slots (in addition to DZ11-A) Second distribution box for DZ11 lines
CPU Options	7 for memory array cards 1 each for User Control Store, Remote Diagnosis Option, and general I/O adaptor slots for up to 3 MASSBUS adapters or 1 UNIBUS and 2 MASSBUS adapters
Max. UNIBUS D.C. Power in VAX-11/750 Cabinet	+15 Volts 2 amps −15 Volts 3.5 amps + 5 Volts 25.0 amps

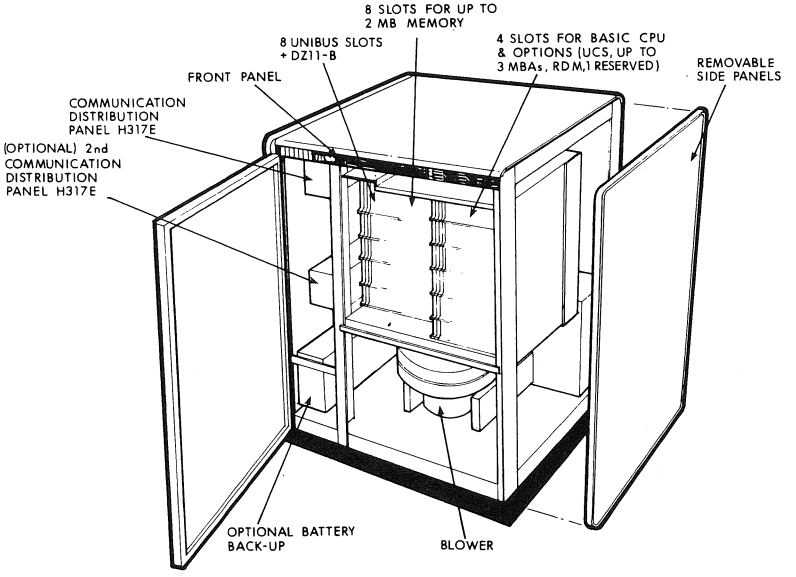
Specification for TU58 in VAX-11/750 Console

Capacity per cartridge	256 KB (512 records by 512 bytes)
Capacity per track	128 KB
Tape length	140 feet
Data transfer rate	19.2 KB/s
Average access time	9.3 seconds
Maximum access time	28 seconds

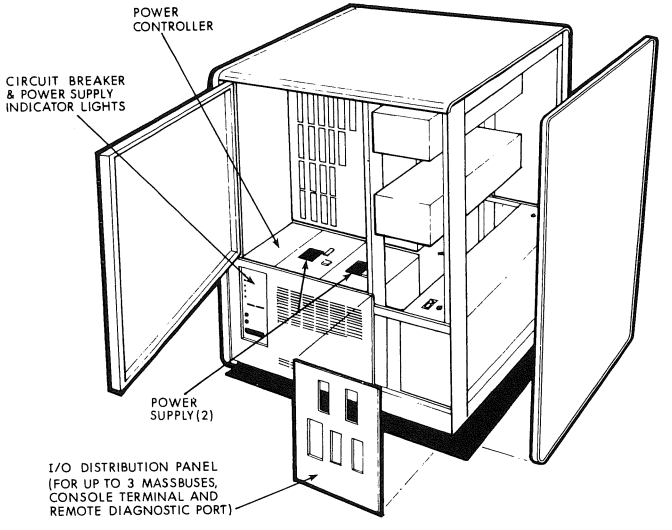
Read/write tape speed	30 in/s
Search tape speed	60 in/s
Bit density	800 bpi
Number of passes per cartridge to read/write cartridge	4 (2 tracks interleaved)
Best case read/write time for entire cartridge	10 minutes

Custom Technology Specifications for the VAX-11/750

Implementation technique	Gate arrays
Circuit technology	Low power bipolar Schottky
Circuit density	Large scale integration (LSI)
Die size	.215 inches × .244 inches
Power utilized per die	2 watts maximum
Package size	144 sq. in. (2.4 × 0.6 inches)
Number of pins/package	48
I/O circuits/die	44 I/O transceiver gates
Logic gates	400 identical 4-input NAND gates
Voltages used	+2.5 volts, +5 volts
Speed per gate	5-10 nanoseconds



Rear View of VAX-11/750 System Cabinet



Front View of VAX-11/750 System Cabinet

APPENDIX L

TECHNICAL SPECIFICATIONS FOR VAX-11/780 PROCESSOR

NOTE

The VAX-11/782 consists of two VAX-11/780 processors, and thus specifications are listed only for the VAX-11/780.

Processor Type	Microprogrammed, 99-bit control store word
	Microcontrol store instruction time—200 nanoseconds
	Control store size—6 K words (99-bit words), 4 K words ROM and 2 K words WDCS
Internal data path	32 bits

CPU Cache Memory

Size	8 KB, 2-way set associative
Effective main memory cycle time	1800 nanoseconds/64 bits
Typical hit ratio	95%
Typical cache cycle time	290 nanoseconds

CPU Address Translation Buffer

Size	128 address translations
Typical hit ratio	97%

CPU Clocks

Realtime clock	Crystal controlled, .01% accuracy 1 μs. resolution
Time-of-year clock	Includes recharging battery backup for over 100 hours

VAX Instruction Set

16 32-bit registers

248 basic operations

32 priority interrupt levels

Multiple data types	Integer, floating point, packed decimal, character string, variable bit fields, and numeric strings PDP-11 compatability mode instructions
Addressing modes	9

Other Standard Features

- Power fail/automatic restart
- Single serial line ASCII console interface
- 8-line communications multiplexer (DZ11-A)
- RX01 floppy disk drive
- Writable diagnostic control store (WDCS)
- Extensive reliability and maintainability features
- Virtual console commands from DECwriter terminal

Main Memory

Virtual address space	4 billion bytes
Physical address lines	1 billion bytes (30 bits)
Physical expansion	8 MB in 256 KB increments

Parity	8-bit error correcting code (ECC) per 64-bit quadword
Technology	16 K-bit dynamic RAMs (200 nanosecond access time)
Cycle times	800 nanoseconds per 64-bit read (1300 nanoseconds with single-bit errors) 1400 nanoseconds per 64-bit write
Power failure protection	Optional battery backup

I/O UNIBUS Adapter (1 standard, up to 4 total)

Maximum UNIBUS I/O rate	1.35 MB/s through buffered data paths
Buffered data paths	15 total, 8-byte buffer in each
Maximum number of bus loads	18 without a repeater
Interrupts	Directly vectored via UNIBUS adapter

VAX-11/780 Options

MASSBUS Adapters (up to 4)	Maximum bandwidth 2.0 MB/s per MBA (to a system maximum of 5.0 MB/s)
Devices	Up to 8 high-speed disks or tapes
Buffer size	32 bytes per MBA

User Control Store

Size	2K word (99-bit words)
Technology	Writable memory (RAM), 70 μs access time

Memory Battery Backup

Minimum backup time	10 minutes with 2 Mb of MS780-D memory

Floating Point Accelerator

Enhances performance of all floating point instructions (single & double precision) including polynomial evaluation, integer/floating conversions, 8-, 16-, and 32-bit integer multiply.

Mechanical (VAX-11/780 cabinet as installed)

Weight	498 kg (1100 lbs.)
Height	153.7 cm (60.5 in.)
Width	118.1 cm (46.5 in.)
Depth	76.2 cm (30 in.)

Electrical Power Requirements

AC line voltage	120/208V
Frequency tolerance	59-61 Hz
Phases	3 phase phase A: 11.2 A max. continous phase B: 9.9 A max. continous phase C: 13.1 A max. continous neutral: 14.4 A max. continous
Also available	220/380V 50 Hz and 240/415V 50 Hz
Plug type	L5-30P (varies by country)
AC cable length	3 m (9.84 ft.) from back of cabinet
Maximum heat dissipation	5350 kcal/hr (21,230 BTU/hr)
Maximum ac power consumption	6225 watts

Typical Power Requirements and Thermal Dissipation

Option	Watts	kcal/hr (BTU/hr)
Fully-configured CPU cabinet	6,225 Watts	5,350 kcal/hr (21,230 BTU/hr)
Fully-configured CPU expansion cabinet, *H9602-HA(HB)*	2,000 Watts	1,720 kcal/hr (6,820 BTU/hr)
Fully-configured UNIBUS options cabinet *H9602-MF(MH)*	2,000 Watts	1,720 kcal/hr (6,820 BTU/hr)
MBA	150 Watts	130 kcal/hr (512 BTU/hr)
UBA, *DW780-AA(AB)*	300 Watts	260 kcal/hr (1024 BTU/hr)
512 KB memory, control & power supply	350 Watts	302 kcal/hr (1,195 BTU/hr)
Fully configured multiport memory (2 MA780 subsystems, 4 MB Memory, 8 ports)	1800 Watts	1550 kcal/hr (6140 BTU/hr)
Multiport memory with 1 MA780 subsystem, 2 MB memory, 4 ports	1000 Watts	860 kcal/hr (3410 BTU/hr)
DR780 (VAX-11/780 only)	226 Watts	194 kcal/hr (771 BTU/hr)
FP780	300 Watts	260 kcal/hr (1,025 BTU/hr)

KU780-A	100 Watts	86 kcal/hr (341 BTU/hr)
H7112	25 Watts	22 kcal/hr (85 BTU/hr)

NOTE

UNIBUS peripheral controllers plugged into the remaining eight slots in the VAX-11/780 DD11-DK backplane must also be included for an accurate total.

Environment

Operating:

Radiated acoustic noise level (at 1 meter distant, 1.5 meter height)	front: 74 dB rear: 65 dB
Temperature	15° to 32°C (59° to 90°F)
Relative humidity	20% to 80%
Maximum altitude	2.4 km (8000 ft.)

Nonoperating:

Temperature	−40° to 66°C (−40° to 151°F)
Relative humidity	0 to 95%

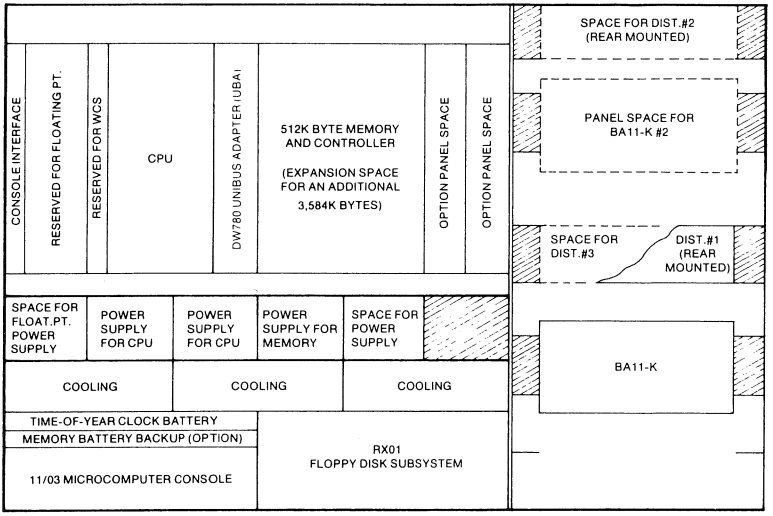


Figure VAX-11/780 Cabinet Space Allocation

* UNIBUS expansion cabinet is included in all standard VAX-11/780 systems.

APPENDIX M

SYSTEM THROUGHPUT CONSIDERATIONS FOR THE VAX-11/780

For the majority of applications, the standard VAX-11/780 packaged systems provide more than ample I/O bus capacity. System throughput is determined by the computing requirements and the time waiting for the disk to seek or spin. This appendix is intended to provide guidance in configuring those remaining VAX-11/780 applications that are characterized by very intensive I/O.

Bus and Memory Bandwidths

The bandwidth or speed capacity of the SBI is 13.3 million bytes per second. That is not to say that 13.3 MB/s will be achieved in any specific configuration or application. In practice, the flow of data is usually much slower because of contention between devices, because of irregularities in the rate of requests, and because the devices may not be able to capitalize on available bandwidth. Often all of these conditions are true. The 13.3 MB/s number could be called the protocol limit to the bus bandwidth.

The focal point of traffic on the bus is the memory subsystem, since most transfers are between a memory and one of the other types of connections, either the CPU or an I/O bus adapter. The memory is 72 bits wide (64 bits of data plus 8 ECC bits). This width contributes to the capacity of a fully configured large system to handle an enormous amount of I/O traffic, inasmuch as a single memory can accept a 64-bit write every 0.8 μs. However, it takes 1.4 μs to handle writes of less than 64 bits since the memory must do an internal update rather than a simple overwrite.

Of the other connections to the SBI, the CPU represents the most intensive traffic load on the memory subsystem unless the configuration includes a DR780. The CPU can and often will request bursts of repeated 32-bit writes to memory every 1.2 μs and less often request short bursts of 64-bit reads from memory. Although the processor in computing will request data much more often than it will generate data, the effect of the very large, write-through cache is to reverse these statistics as seen by the memory subsystem. The cache portion of the processor in typical applications will do 32-bit writes to memory more than ten times as often as it will do 64-bit reads. Frequently, these 32-bit writes will occur in bursts, as for example when the operating system clears a page of memory or when a COBOL application moves data into a buffer. This would mean that despite its low priority,

the processor could easily dominate a single memory controller; if the request buffer of the memory controller were enabled fully, the processor could have a write in process, four writes pending, and another write waiting to be requested. This would result in long delays as seen by the I/O bus adapters, and high-speed disk transfers would be marginal at best. For this reason, the memory request buffer is by design limited to a depth of one command unless interleaving is enabled. Even with the request buffer limited, the processor presents sufficient contention to limit the effective MASSBUS speed capacity in single memory configurations to about 1.3 to 1.5 MB/s, a figure below the MASSBUS protocol bandwidth but sufficient to support disk products.

If there is more than one MASSBUS channel transferring data concurrently, then interchannel contention for a single memory controller can contribute to data-lates and overruns. Except in certain time-critical or data acquisition applications, an occasional data-late is harmless (VAX/VMS will simply retry the operation), but an excessive percentage of data-lates can interfere with productive work. A system based on interleaved memory will almost never experience even a harmless data-late.

MASSBUS Arrangements

The VAX/VMS system offers flexibility in physically configuring the arrangement of disks and tapes over one or more MASSBUS channels. Typically tapes are installed on one channel, and all disks are on another, even if there are several disks. This is not always the best arrangement; sometimes there should be disks on the tape channel; sometimes extra channels should be included even if the total number of disks is below the maximum allowed on one channel. The best arrangement depends on the objectives of the installation and the characteristics of the application.

MASSBUS products support overlapped control operations. Once a control function such as a disk seek or a tape rewind has been initiated, the MASSBUS is free to start another function, control or transfer. This is supported in VAX/VMS, so overlapped seeking, for example, happens whether the disks are distributed over multiple channels or concentrated on one. However, once a data transfer is physically started, then it must complete or abort before another data transfer operation on that *same channel* can be started or a seek completion can be recognized. It is for this reason that tape drives are advisable on a disk channel only if use of the two is essentially mutually exclusive. Except for memory contention, the MASSBUS channels are quite independent, and VAX/VMS supports concurrent operations on multiple buses.

Tape drives connect to a MASSBUS differently than do disks. Disks interface to a MASSBUS directly through interface electronics associated with and replicated in each individual disk. Tape drives are cabled to a common interface called a formatter which in turn connects to the MASSBUS. The formatter is included as one of the components of a master tape drive. Typically all of the tapes on a system share, and contend for, the services of the same formatter. They need not, and this is a consideration for applications with intense tape traffic.

Because VAX/VMS supports concurrent multiple bus traffic, the I/O capacity of a fully configured VAX can be very large, but only if the memory subsystem is matched to the I/O subsystem with consideration for the processor as an essential element. If the memory subsystem is the single controller minimum, then the restriction that transfers on the MASSBUS are mutually exclusive can be a beneficial restriction in that the software cannot initiate a transfer until the previous one is complete. Unproductive contention for the single memory can be partially or completely avoided by putting the application-required number of disks on the physically minimum possible number of buses. If contention would still be a problem, there is no sensible alternative other than a second memory controller. The immediate need is to predict the threat of unproductive memory contention; the second need is to estimate the impact of the resulting data-lates and overruns.

Unproductive memory contention will occur if the aggregate set of DMA buses cannot all together sustain their individual peak instantaneous rates. The emphasis is on the peak rates, not the average rates; the ratio of peak to average determines the frequency of contention and the impact but not the eventual certainty. As a guideline, the maximum allowable peak rates for a single memory controller system are two channels at 1.3 MB/s or three channels at 0.806 MB/s.

Another general guideline to follow is: if transfers are relatively short and infrequent, then memory contention is not likely to be a problem because there will be adequate capacity for a retry. However, if the transfers are long and occur frequently, then unproductive memory contention will be much more bothersome because of the high likelihood that the retrys will also fail.

Two memory controllers interleaved can easily support a full complement of I/O channels at full speed. By including the second memory and extra MASSBUS channels, the application can benefit by spreading the disk traffic over multiple channels. This benefit can be significant if the physical transfers are long and negligible if the transfers are typically only a few blocks. Swapping traffic and some data processing

applications are characterized by transfers of more than half a disk track. Transfers of only a few blocks tend to be dominated by the rotational latency.

Impact on the Processor

Whenever there is sufficiently intense I/O traffic, the effective speed of the processor is slowed by the contention for memory. This is rarely important, but if the success of the application is critically dependent on a prediction of execution time, then this effect should be considered. The major I/O parameter here is the total of the average I/O rates. In predicting the average I/O rate, remember that the relationship between peak and average depends upon the transfer length, the dead time between transfers, and any gaps within the transfer. With two memory controllers the processor will be slowed roughly 4% per averaged megabyte per second of I/O traffic; with only a single memory the impact is about two to four times greater.

For example, consider a small system with RP06 disk drives. The average I/O load consists of 15 transfers per second with each transfer to be 5 KB long. Although the peak transfer rate of an RP06 is 0.806 MB/s, the average rate would be 15 multiplied by 0.005, or 0.075 MB/s. With the previously mentioned 4% per averaged MB/s slowdown factor, the throughput slowdown in the system being considered here is 0.3%, or virtually nil. If, instead, the I/O load were a multibuffered disk-to-disk copy from one channel to another, then ignoring overhead and allowing 10 ms for an adjacent track seek, 7 ms for rotational latency after each seek, and 214,016 bytes to be transferred in the next 19 revolutions, the calculated average rate per disk is 0.64 MB/s. If the seeks and transfers of the other disk were completely overlapped with the first, then the total average I/O rate during the copy would be 1.28 MB/s, corresponding to a slowdown of 5% incorporating the 4% per MB/s factor. This is still within the speed tolerance of the basic processor specification, but the example and methodology shown here should be helpful in applications with really intense I/O traffic.

Even in the absence of I/O traffic, the computing rate of a system with a single memory controller will be as much as 15% slower than an interleaved memory configuration. The slowdown depends on the frequency of consecutive writes to memory. In most cases it is imperceptible.

UNIBUS Configurations

The total throughput of any UNIBUS system is a function of both the bus master and the bus slave. The characteristic that comes into play in determining system throughput is the delay from the time MSYN is

received on the bus until SSYN is asserted for all of the different types of functions that the UNIBUS adapter can perform. The actual measured times on a system can vary as a function of other system activity. If the memory or the SBI is in use when the UNIBUS adapter makes a reference, then the UNIBUS adapter will be stalled until that activity is completed and the MSYN to SSYN time will be lengthened accordingly.

DMC-11 Configurations

The DMC-11 Network Link is a high performance interconnection of UNIBUS structured systems for use where the computers are located within the same facility. The DMC-11 can be configured for operation at up to 1 megabaud (1 million bits per second). The DMC-11 can also be used with non-UNIBUS computers, such as the VAX-11/780, through the UNIBUS adapter. However, certain care must be exercised in the operation.

In the case of the VAX-11/780, the DMC-11 cannot take advantage of the buffered data path, but must use the direct data path, which is slower. The resultant NPR rate for the DMC-11 is not fast enough to support the one megabit per second rate in both transmit and receive lines simultaneously. Therefore, simultaneous transmit and receive at one megabit can occur only as long as the silos in the DMC-11 provide some slack.

Whenever the transmit silo is empty and the receive silo is full, the transmit line will suffer underruns and experience CRC errors. The message rejected by the CRC error would be corrected on an automatic retry unless the line is so busy that the retry also fails.

The performance on the line is therefore dependent on the amount of line usage or the message length, and since the amount of usage is not usually controlled, the message length is the only parameter that can be used to prevent link problems from occurring. A message length of 100 bytes or fewer will not cause problems on a VAX-11/780 processor, but larger message lengths could cause problems when link usage is high. The guideline is therefore to keep message lengths below 100 bytes at the one megabaud rate.

This throughput issue is a major concern when the DMC-11 is configured in VAX-11/780 systems. The use of DECnet software and its inherent protocol costs will allow the DMC-11 to operate at one megabit speeds, although the average output will be much lower.

KMC-11 Configurations

The KMC-11 is an auxiliary processor, complete with memory, that can be used to off-load from the CPU such tasks as data communica-

tions, and analog input/output. The KMC-11 operates in parallel with the main CPU, or in the case of VAX-11/780 systems, in parallel with the UNIBUS adapter.

The KMC-11 interface to the UNIBUS is directly controlled by the microcode in the KMC-11. Through this microcode, it is possible to gain control of the UNIBUS, hold it, and disrupt the normal operation of other units, including the VAX processor. It is important that VAX-11/780 systems incorporating KMC-11s under the direction of user-developed software do not hold the UNIBUS for longer than 20 microseconds.

The KMC-11 can affect UNIBUS throughput in another manner also. As an auxiliary processor, the KMC-11 is continuously asserting itself as bus master to determine if there is work for it to do, such as processing data from a DZ11 multiplexer. If the rate at which the KMC-11 asserts its bus mastership is great enough, UNIBUS throughput can be significantly reduced. Careful use of the KMC-11, or the use of additional UNIBUS adapters, can help avoid this situation.

Three reference sources useful to those interfacing devices to VAX systems through the UNIBUS adapter are the *PDP-11 UNIBUS Design Description* shown in the *PDP-11 UNIBUS Handbook*, Order No. EB-17525-20, *DW780 UNIBUS Adapter Technical Description*, Order No. EK-DW780-TD, and the *I/O User's Guide*, VMS Documentation Set, Volume III.

Impact of the DR780

VAX-11/780 systems which incorporate a DR780 high performance interconnect also have configuration and application dependent throughput considerations. A detailed discussion of these issues and a chart showing typical throughput rates as a function of system configuration and workload can be found in Chapter 19. The *DR780 General Purpose Interface Technical Description*, Order No. EK-DR780-TD and the *DR780 User's Guide*, Order No. EK-DR780-UG are also available reference sources.

Impact of the MA780

VAX-11/780 systems using the MA780 multiport memory option also have unique throughput considerations depending on the total system configuration and application. A description of the MA780 can be found in Chapter 19 of this handbook and a detailed technical description of its operation is available in the *MA780 Multiport Memory Technical Description*, Order No. EK-MA780-TD.

GLOSSARY

abort An exception that occurs in the middle of an instruction and potentially leaves the registers and memory in an indeterminate state, such that the instruction cannot necessarily be restarted.

absolute indexed mode An indexed addressing mode in which the base operand specifier is addressed in absolute mode.

absolute mode In absolute mode addressing, the PC is used as the register in autoincrement deferred mode. The PC contains the address of the location containing the actual operand.

absolute time Time values expressing a specific date (month, day, and year) and time of day. Absolute time values are always expressed in the system as positive numbers.

access mode 1. Any of the four processor access modes in which software executes. Processor access modes are, in order from most to least privileged and protected: kernel (mode 0), executive (mode 1), supervisor (mode 2), and user (mode 3). When the processor is in kernel mode, the executing software has complete control of, and responsibility for, the system. When the processor is in any other mode, the processor is inhibited from executing privileged instructions. The Processor Status Longword contains the current access mode field. The operating system uses access modes to define protection levels for software executing in the context of a process. For example, the Executive runs in kernel and executive mode and is most protected. The command interpreter is less protected and runs in supervisor mode. The debugger runs in user mode and is no more protected than normal user programs. 2. See record access mode.

access type 1. The way in which the processor accesses instruction operands. Access types are: read, write, modify, address, and branch. 2. The way in which a procedure accesses its arguments. 3. See also record access type.

access violation An attempt to reference an address that is not mapped into virtual memory or an attempt to reference an address that is not accessible by the current access mode.

account name A string that identifies a particular account used to accumulate data on a job's resource use. This name is the user's accounting charge number, not the user's UIC.

address A number used by the operating system and user software to identify a storage location. See also virtual address and physical address.

address access type The specified operand of an instruction is not directly accessed by the instruction. The address of the specified operand is the actual instruction operand. The context of the address calculation is given by the data type of the operand.

addressing mode The way in which an operand is specified; for example, the way in which the effective address of an instruction operand is calculated using the general registers. The basic general register addressing modes are called: register, register deferred, autoincrement, autoincrement deferred, autodecrement, displacement, and displacement deferred. In addition, there are six indexed addressing modes using two general registers, and literal mode addressing. The PC addressing modes are called: immediate (for register deferred mode using the PC), absolute (for autoincrement deferred mode using the PC), and branch.

address space The set of all possible addresses available to a process. Virtual address space refers to the set of all possible virtual addresses. Physical address space refers to the set of all possible physical addresses sent out on the SBI.

allocate a device To reserve a particular device unit for exclusive use. A user process can allocate a device only when that device is not allocated by any other process.

alphanumeric character An upper or lower case letter (A-Z, a-z), a dollar sign ($), an underscore (_), or a decimal digit (0-9).

alternate key An optional key within the data records in an indexed file; used by VAX-11 RMS to build an alternate index. See key (indexed files) and primary key.

American Standard Code for Information Interchange (ASCII) A set of 8-bit binary numbers representing the alphabet, punctuation, numerals, and other special symbols used in text representation and communications protocol.

Ancillary Control Process (ACP) A process that acts as an interface between user software and an I/O driver. An ACP provides functions supplemental to those performed in the driver, such as file and directory management. Three examples of ACPs are: the Files-11 ACP (F11ACP), the magnetic tape ACP (MTACP), and the networks ACP (NETACP).

area Areas are VAX-11 RMS maintained regions of an indexed file. They allow a user to specify placement and/or specific bucket sizes for particular portions of a file. An area consists of any number of buckets, and there may be from 1 to 255 areas in a file.

Argument Pointer General register 12 (R12). By convention, AP contains the address of the base of the argument list for procedures initiated using the CALL instructions.

assign a channel To establish the necessary software linkage between a user process and a device unit before a user process can transfer any data to or from that device. A user process requests the system to assign a channel and the system returns a channel number.

asynchronous record operation A mode of record processing in which a user program can continue to execute after issuing a record retrieval or storage request without having to wait for the request to be fulfilled.

Asynchronous System Trap A software-simulated interrupt to a user-defined service routine. ASTs enable a user process to be notified asynchronously with respect to its execution of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes the process at the point where it was interrupted.

Asynchronous System Trap level (ASTLVL) A value kept in an internal processor register that is the highest access mode for which an AST is pending. The AST does not occur until the current access mode drops in priority (raises in numeric value) to a value greater than or equal to ASTLVL. Thus, an AST for an access mode will not be serviced while the processor is executing in a higher priority access mode.

authorization file See user authorization file.

autodecrement indexed mode An indexed addressing mode in which the base operand specifier uses autodecrement mode addressing.

autodecrement mode In autodecrement mode addressing, the contents of the selected register are decremented, and the result is used as the address of the actual operand for the instruction. The contents of the register are decremented according to the data type context of the register: 1 for byte, 2 for word, 4 for longword and floating, 8 for quadword and double floating.

autoincrement deferred indexed mode An indexed addressing mode in which the base operand specifier uses autoincrement deferred mode addressing.

autoincrement deferred mode In autoincrement deferred mode addressing, the specified register contains the address of a longword

which contains the address of the actual operand. The contents of the register are incremented by 4 (the number of bytes in a longword). If the PC is used as the register, this mode is called absolute mode.

autoincrement indexed mode An indexed addressing mode in which the base operand specifier uses autoincrement mode addressing.

autoincrement mode In autoincrement mode addressing, the contents of the specified register are used as the address of the operand, then the contents of the register are incremented by the size of the operand.

balance set The set of all process working sets currently resident in physical memory. The processes whose working sets are in the balance set have memory requirements that balance with available memory. The balance set is maintained by the system swapper process.

base operand address The address of the base of a table or array referenced by index mode addressing.

base operand specifier The register used to calculate the base operand address of a table or array referenced by index mode addressing.

base priority The process priority that the system assigns a process when it is created. The scheduler never schedules a process below its base priority. The base priority can be modified only by the system manager or the process itself.

base register A general register used to contain the address of the first entry in a list, table, array, or other data structure.

binding See linking.

bit string See variable-length bit field.

block 1. The smallest addressable unit of data that the specified device can transfer in an I/O operation (512 contiguous bytes for most disk devices). 2. An arbitrary number of contiguous bytes used to store logically related status, control, or other processing information.

block I/O The set of VAX-11 RMS procedures that allow you direct access to the blocks of a file regardless of file organization.

bootstrap block A block in the index file on a system disk that contains a program that can load the operating system into memory and start its execution.

branch access type An instruction attribute which indicates that the processor does not reference an operand address, but that the oper-

and is a branch displacement. The size of the branch displacement is given by the data type of the operand.

branch mode In branch addressing mode, the instruction operand specifier is a signed byte or word displacement. The displacement is added to the contents of the updated PC (which is the address of the first byte beyond the displacement), and the result is the branch address.

bucket A storage structure, consisting of from 1 to 32 blocks, used for building and processing relative and indexed files. A bucket contains one or more records or record cells. Buckets are the unit of contiguous transfer between VAX-11 RMS buffers and the disk.

buffered I/O See system buffered I/O.

bug check The operating system's internal diagnostic check. The system logs the failure and crashes the system.

byte A byte is eight contiguous bits starting on an addressable byte boundary. Bits are numbered from the right, 0 through 7, with bit 0 the low-order bit. When interpreted arithmetically, a byte is a 2's complement integer with significance increasing from bits 0 through 6. Bit 7 is the sign bit. The value of the signed integer is in the range −128 to 127 decimal. When interpreted as an unsigned integer, significance increases from bits 0 through 7 and the value of the unsigned integer is in the range 0 to 255 decimal. A byte can be used to store one ASCII character.

cache memory A small, high-speed memory placed between slower main memory and the processor. A cache increases effective memory transfer rates and processor speed. It contains copies of data recently used by the processor, and fetches several bytes of data from memory in anticipation that the processor will access the next sequential series of bytes.

call frame See stack frame.

call instructions The processor instructions CALLG (Call Procedure with General Argument List) and CALLS (Call Procedure with Stack Argument List).

call stack The stack, and conventional stack structure, used during a procedure call. Each access mode of each process context has one call stack, and interrupt service context has one call stack.

channel A logical path connecting a user process to a physical device unit. A user process requests the operating system to assign a channel to a device so the process can transfer data to or from that device.

character A symbol represented by an ASCII code. See also alphanumeric character.

character string A contiguous set of bytes. A character string is identified by two attributes: an address and a length. Its address is the address of the byte containing the first character of the string. Subsequent characters are stored in bytes of increasing addresses. The length is the number of characters in the string.

character string descriptor A quadword data structure used for passing character data (strings). The first word of the quadword contains the length of the character string. The second word can contain type information. The remaining longword contains the address of the string.

cluster 1. A set of contiguous blocks that is the basic unit of space allocation on a Files-11 disk volume. 2. A set of pages brought into memory in one paging operation. 3. An event flag cluster.

command An instruction, generally an English word, typed by the user at a terminal or included in a command file which requests the software monitoring a terminal or reading a command file to perform some well-defined activity. For example, typing the COPY command requests the system to copy the contents of one file into another file.

command file A file containing command strings. See also command procedure.

command interpreter Procedure-based system code that executes in supervisor mode in the context of a process to receive, syntax check, and parse commands typed by the user at a terminal or submitted in a command file.

command parameter The positional operand of a command delimited by spaces, such as a file specification, option, or constant.

command procedure A file containing commands and data that the command interpreter can accept in lieu of the user typing the commands individually on a terminal.

command string A line (or set of continued lines), normally terminated by typing the carriage return key, containing a command and, optionally, information modifying the command. A complete command string consists of a command, its qualifiers, if any, and its parameters (file specifications, for example), if any, and their qualifiers, if any.

common A FORTRAN term for a program section that contains only data.

common event flag cluster A set of 32 event flags that enables

cooperating processes to post event notification to each other. Common event flag clusters are created as they are needed. A process can associate with up to two common event flag clusters.

compatibility mode A mode of execution that enables the central processor to execute non-privileged PDP-11 instructions. The operating system supports compatibility mode execution by providing an RSX-11M programming environment for an RSX-11M task image. The operating system compatibility mode procedures reside in the control region of the process executing a compatibility mode image. The procedures intercept calls to the RSX-11M Executive and convert them to the appropriate operating system functions.

condition An exception condition detected and declared by software. For example, see failure exception mode.

condition codes Four bits in the Processor Status Word that indicate the results of previously executed instructions.

condition handler A procedure that a process wants the system to execute when an exception condition occurs. When an exception condition occurs, the operating system searches for a condition handler and, if found, initiates the handler immediately. The condition handler may perform some action to change the situation that caused the exception condition and continue execution for the process that incurred the exception condition. Condition handlers execute in the context of the process at the access mode of the code that incurred the exception condition.

condition value A 32-bit quantity that uniquely identifies an exception condition.

context The environment of an activity. See also process context, hardware context, and software context.

context indexing The ability to index through a data structure automatically because the size of the data type is known and used to determine the offset factor.

context switching Interrupting the activity in progress and switching to another activity. Context switching occurs as one process after another is scheduled for execution. The operating system saves the interrupted process' hardware context in its hardware process control block (PCB) using the Save Process Context instruction, and loads another process' hardware PCB into the hardware context using the Load Process Context instruction, scheduling that process for execution.

continuation character A hyphen at the end of a command line signifying that the command string continues on to the next command line.

console The manual control unit integrated into the central processor. The console includes an LSI-11 microprocessor and a serial line interface connected to a hard copy terminal. It enables the operator to start and stop the system, monitor system operation, and run diagnostics.

console terminal The hard copy terminal connected to the central processor console.

control region The highest-addressed half of per-process space (the P1 region). Control region virtual addresses refer to the process-related information used by the system to control the process, such as: the kernel, executive, and supervisor stacks, the permanent I/O channels, exception vectors, and dynamically used system procedures (such as the command interpreter and RSX-11M programming environment compatibility mode procedures). The user stack is also normally found in the control region, although it can be relocated elsewhere.

Control Region Base Register (P1BR) The processor register, or its equivalent in a hardware process control block, that contains the base virtual address of a process control region page table.

Control Region Length Register (P1LR) The processor register, or its equivalent in a hardware process control block, that contains the number of non-existent page table entries for virtual pages in a process control region.

copy-on-reference A method used in memory management for sharing data until a process accesses it, in which case it is copied before the access. Copy-on-reference allows sharing of the initial values of a global section whose pages have read/write access but contain pre-initialized data available to many processes.

counted string A data structure consisting of a byte-sized length followed by the string. Although a counted string is not used as a procedure argument, it is a convenient representation in memory.

current access mode The processor access mode of the currently executing software. The Current Mode field of the Processor Status Longword indicates the access mode of the currently executing software.

cylinder The tracks at the same radius on all recording surfaces of a disk.

D floating (point) datum A floating point datum consisting of 8 contiguous bytes (64 bits) starting on an arbitrary byte boundary. The value of the D_floating datum is in the approximate range (+ or −) 0.29×10^{-38} to 1.7×10^{38}. The precision is approximately one part in 2^{55} or typically sixteen decimal digits.

data base 1. All the occurrences of data described by a data base management system. 2. A collection of related data structures.

data structure Any table, list, array, queue, or tree whose format and access conventions are well-defined for reference by one or more images.

data type In general, the way in which bits are grouped and interpreted. In reference to the processor instructions, the data type of an operand identifies the size of the operand and the significance of the bits in the operand. Operand data types include: byte, word, longword and quadword integer, floating and double floating, character string, packed decimal string, and variable-length bit field.

deferred echo Refers to the fact that terminal echoing does not occur until a process is ready to accept input entered by type ahead.

delta time A time value expressing an offset from the current date and time. Delta times are always expressed in the system as negative numbers whose absolute value is used as an offset from the current time.

demand zero page A page, typically of an image stack or buffer area, that is initialized to contain all zeros when dynamically created in memory as a result of a page fault. This feature eliminates the waste of disk space that would otherwise be required to store blocks (pages) that contain only zeros.

descriptor A data structure used in calling sequences for passing argument types, addresses and other optional information. See character string descriptor.

detached process A process that has no owner. The parent process of a tree of subprocesses. Detached processes are created by the job controller when a user logs on the system or when a batch job is initiated. The job controller does not own the user processes it creates; these processes are therefore detached.

device The general name for any physical terminus or link connected to the processor that is capable of receiving, storing, or transmitting data. Card readers, line printers, and terminals are examples of record-oriented devices. Magnetic tape devices and disk devices are examples of mass storage devices. Terminal line interfaces and interprocessor links are examples of communications devices.

device interrupt An interrupt received on interrupt priority level 16 through 23. Device interrupts can be requested only by devices, controllers, and memories.

device name The field in a file specification that identifies the device unit on which a file is stored. Device names also include the mnemonics that identify an I/O peripheral device in a data transfer request. A device name consists of a mnemonic followed by a controller identification letter (if applicable), followed by a unit number (if applicable), and ends with a colon (:).

device queue See spool queue.

device register A location in device controller logic used to request device functions (such as I/O transfers) and/or report status.

device unit One drive, and its controlling logic, of a mass storage device system. A mass storage system can have several drives connected to it.

diagnostic A program that tests logic and reports any faults it detects.

direct I/O An I/O operation in which the system locks the pages containing the associated buffer in memory for the duration of the I/O operation. The I/O transfer takes place directly from the process buffer. Contrast with system buffered I/O.

direct mapping cache A cache organization in which only one address comparison is needed to locate any data in the cache because any block of main memory data can be placed in only one possible position in the cache. Contrast with fully associative cache.

directory A file used to locate files on a volume that contains a list of file names (including type and version number) and their unique internal identifications.

directory name The field in a file specification that identifies the directory file in which a file is listed. The directory name begins with a left bracket ([or <) and ends with a right bracket (] or >).

displacement deferred indexed mode An indexed addressing mode in which the base operand specifier uses displacement deferred mode addressing.

displacement deferred mode In displacement deferred mode addressing, the specifier extension is a byte, word, or longword displacement. The displacement is sign extended to 32 bits and added to a base address obtained from the specified register. The result is the address of a longword which contains the address of the actual operand. If the PC is used as the register, the updated contents of the PC

are used as the base address. The base address is the address of the first byte beyond the specifier extension.

displacement indexed mode An indexed addressing mode in which the base operand specifier uses displacement mode addressing.

displacement mode In displacement mode addressing, the specifier extension is a byte, word, or longword displacement. The displacement is sign extended to 32 bits and added to a base address obtained from the specified register. The result is the address of the actual operand. If the PC is used as the register, the updated contents of the PC are used as the base address. The base address is the address of the first byte beyond the specifier extension.

drive The electro-mechanical unit of a mass storage device system on which a recording medium (disk cartridge, disk pack, or magnetic tape reel) is mounted.

driver The set of code that handles physical I/O to a device.

dynamic access A technique in which a program switches from one record access mode to another while processing a file.

echo A terminal handling characteristic in which the characters typed by the terminal user on the keyboard are also displayed on the screen or printer.

effective address The address obtained after indirect or indexing modifications are calculated.

entry mask A word whose bits represent the registers to be saved or restored on a subroutine or procedure call using the call and return instructions.

entry point A location that can be specified as the object of a call. It contains an entry mask and exception enables known as the entry point mask.

equivalence name The string associated with a logical name in a logical name table. An equivalence name can be, for example, a device name, another logical name, or a logical name concatenated with a portion of a file specification.

error logger A system process that empties the error log buffers and writes the error messages into the error file. Errors logged by the system include memory system errors, device errors and timeouts, and interrupts with invalid vector addresses.

escape sequence An escape is a transition from the normal mode of operation to a mode outside the normal mode. An escape character

is the code that indicates the transition from normal to escape mode. An escape sequence refers to the set of character combinations starting with an escape character that the terminal transmits without interpretation to the software set up to handle escape sequences.

event A change in process status or an indication of the occurrence of some activity that concerns an individual process or cooperating processes. An incident reported to the scheduler that affects a process' ability to execute. Events can be synchronous with the process' execution (a wait request), or they can be asynchronous (I/O completion). Some other events include: swapping; wake request; page fault.

event flag A bit in an event flag cluster that can be set or cleared to indicate the occurrence of the event associated with that flag. Event flags are used to synchronize activities in a process or among many processes.

event flag cluster A set of 32 event flags which are used for event posting. Four clusters are defined for each process: two process-local clusters, and two common event flag clusters. Of the process-local flags, eight are reserved for system use.

exception An event detected by the hardware (other than an interrupt or jump, branch, case, or call instruction) that changes the normal flow of instruction execution. An exception is always caused by the execution of an instruction or set of instructions (whereas an interrupt is caused by an activity in the system independent of the current instruction). There are three types of hardware exceptions: traps, faults, and aborts. Examples are: attempts to execute a privileged or reserved instruction, trace faults, compatibility mode faults, breakpoint instruction execution, and arithmetic faults such as floating point overflow, underflow, and divide by zero.

exception condition A hardware- or software-detected event other than an interrupt or jump, branch, case, or call instruction that changes the normal flow of instruction execution.

exception dispatcher An operating system procedure that searches for a condition handler when an exception condition occurs. If no exception handler is found for the exception or condition, the image that incurred the exception is terminated.

exception enables See trap enables.

exception vector See vector.

executable image An image that is capable of being run in a process. When run, an executable image is read from a file for execution in a process.

executive The generic name for the collection of procedures included in the operating system software that provides the basic control and monitoring functions of the operating system.

executive mode The second most privileged processor access mode (mode 1). The record management services (RMS) and many of the operating system's programmed service procedures execute in executive mode.

exit An image exit is a rundown activity that occurs when image execution terminates either normally or abnormally. Image rundown activities include deassigning I/O channels and disassociation of common event flag clusters. Any user- or system-specified exit handlers are called.

exit handler A procedure executed when an image exits. An exit handler enables a procedure that is not on the call stack to gain control and clean up procedure-own data bases before the actual image exit occurs.

extended attribute block (XAB) An RMS user data structure that contains additional file attributes beyond those expressed in the file access block (FAB), such as boundary types (aligned on cylinder, logical block number, virtual block number) and file protection information.

extension The amount of space to allocate at the end of a file each time a sequential write exceeds the allocated length of the file.

extent The contiguous area on a disk containing a file or a portion of a file. Consists of one or more clusters.

F_floating (point) datum A floating point datum consisting of 4 contiguous bytes (32 bits) starting on an arbitrary byte boundary. The value of the F_floating datum is in the approximate range (+ or −) 0.29×10^{-38} to 1.7×10^{38}. The precision is approximately one part in 2^{23} or typically seven decimal digits.

failure exception mode A mode of execution selected by a process indicating that it wants an exception condition declared if an error occurs as the result of a system service call. The normal mode is for the system service to return an error status code for which the process must test.

fault A hardware exception condition that occurs in the middle of an instruction and that leaves the registers and memory in a consistent state, such that elimination of the fault and restarting the instruction will give correct results.

field 1. See variable-length bit field. 2. A set of contiguous bytes in a logical record.

file An organized collection of related items (records) maintained in an accessible storage area, such as disk or tape.

file access block (FAB) An RMS user data structure that represents a request for data operations related to the file as a whole, such as OPEN, CLOSE, or CREATE.

file header A block in the index file describing a file on a Files-11 disk structure. The file header identifies the locations of the file's extents. There is a file header for every file on the disk.

file name The field preceding a file type in a file specification that contains a 1- to 9-character logical name for a file.

filename extension See file type.

file organization The physical arrangement of data in the file. You select the specific organization from those offered by VAX-11 RMS, based on your individual needs for efficient data storage and retrieval. See indexed file organization, relative file organization, and sequential file organization.

Files-11 The name of the on-disk structure used by the RSX-11, IAS and VAX/VMS operating systems. Volumes created under this structure are transportable between these operating systems.

file specification A unique name for a file on a mass storage medium. It identifies the node, the device, the directory name, the file name, the file type, and the version number under which a file is stored.

file structure The way in which the blocks forming a file are distributed on a disk or magnetic tape to provide a physical accessing technique suitable for the way in which the data in the file is processed.

file system A method of recording, cataloging, and accessing files on a volume.

file type The field in a file specification that is preceded by a period or dot (.) and consists of a zero- to three-character type identification. By convention, the type identifies a generic class of files that have the same use or characteristics, such as ASCII text files, binary object files, etc.

fixed control area An area associated with a variable length record available for controlling or assisting record access operations. Typical uses include line numbers and printer format control information.

fixed-length record format Property of a file in which all records are

of the same size. This format provides simplicity in determining the exact location of a record in the file and eliminates the need to prefix a record size field to each record.

floating (point) datum A numeric data type in which the number is represented by a fraction (less than 1 and greater than or equal to ½) multiplied by 2 raised to a power. There are four floating point data types: F_floating (4 bytes), D_floating (8 bytes), G_floating (8 bytes), and H_floating (16 bytes)

foreign volume Any volume other than a Files-11 formatted volume which may or may not be file structured.

fork process A dynamically created system process such as a process that executes device driver code or the timer process. Fork processes have minimal context. Fork processes are scheduled by the hardware rather than by the software. The timer process is dispatched directly by software interrupt. I/O driver processes are dispatched by a fork dispatcher. Fork processes execute at software interrupt levels and are dispatched for execution immediately. Fork processes remain resident until they terminate.

frame pointer General register 13 (R13). By convention, FP contains the base address of the most recent call frame on the stack.

fully associative cache A cache organization in which any block of data from main memory can be placed anywhere in the cache. Address comparision must take place against each block in the cache to find any particular block. Contrast with direct mapping cache.

G_floating (point) datum A floating point datum consisting of 8 contiguous bytes (64 bits) starting on an arbitrary byte boundary. The value of the G_floating datum is in the approximate range (+ or −) 0.56×10^{-308} to 0.9×10^{308}. The precision is approximately one part in 2^{52} or typically fifteen decimal digits.

general register Any of the sixteen 32-bit registers used as the primary operands of the native mode instructions. The general registers include 12 general purpose registers which can be used as accumulators, as counters, and as pointers to locations in main memory, and the Frame Pointer (FP), Argument Pointer (AP), Stack Pointer (SP), and Program Counter (PC) registers.

generic device name A device name that identifies the type of device but not a particular unit; a device name in which the specific controller and/or unit number is omitted.

giga Metric term used to represent the number 1 followed by nine zeros.

global page table The page table containing the master page table entries for global sections.

global section A data structure (e.g., FORTRAN global common) or shareable image section potentially available to all processes in the system. Access is protected by privilege and/or group number of the UIC.

global symbol A symbol defined in a module that is potentially available for reference by another module. The linker resolves (matches references with definitions) global symbols. Contrast with local symbol.

global symbol table (GST) In a library, an index of strongly defined global symbols used to access the modules defining the global symbols. The linker will also put global symbol tables into an image. For example, the linker appends a global symbol table to executable images that are intended to run under the symbolic debugger, and it appends a global symbol table to all shareable images.

group 1. A set of users who have special access privileges to each other's directories and files within those directories (unless protected otherwise), as in the context "system, owner, group, world," where group refers to all members of a particular owner's group. 2. A set of jobs (processes and their subprocesses) who have access privileges to a group's common event flags and logical name tables, and may have mutual process controlling privileges, such as scheduling, hibernation, etc.

group number The first number in a User Identification Code (UIC).

H_floating (point) datum A floating point datum consisting of 16 contiguous bytes (128 bits) starting on an arbitrary byte boundary. The value of the H_floating datum is in the approximate range (+ or −) 0.84×10^{4932} to 0.59×10^{4932}. The precision isapproximately one part in 2^{112} or typically 33 decimal digits.

hardware context The values contained in the following registers while a process is executing: the Program Counter (PC); the Processor Status Longword (PSL); the 14 general registers (R0 through R13); the four processor registers (P0BR, P0LR, P1BR and P1LR) that describe the process virtual address space; the Stack Pointer (SP) for the current access mode in which the processor is executing; plus the contents to be loaded in the Stack Pointer for every access mode other than the current access mode. While a process is executing, its hardware context is continually being updated by the processor. While a process is not executing, its hardware context is stored in its hardware PCB.

hardware process control block (PCB) A data structure known to the processor that contains the hardware context when a process is not executing. A process' hardware PCB resides in its process header.

hibernation A state in which a process is inactive, but known to the system with all of its current status. A hibernating process becomes active again when a wake request is issued. It can schedule a wake request before hibernating, or another process can issue its wake request. A hibernating process also becomes active for the time sufficient to service any AST it may receive while it is hibernating. Contrast with suspension.

home block A block in the index file that contains the volume identification, such as volume label and protection.

image An image consists of procedures and data that have been bound together by the linker. There are three types of images: executable, shareable, and system.

image activator A set of system procedures that prepare an image for execution. The image activator establishes the memory management data structures required both to map the image's virtual pages to physical pages and to perform paging.

image exit See exit.

image I/O segment That portion of the control region that contains the RMS internal file access blocks (IFAB) and I/O buffers for the image currently being executed by a process.

image name The file name of the file in which an image is stored.

image privileges The privileges assigned to an image when it is linked. See process privileges.

image section (isect) A group of program sections (psects) with the same attributes (such as read-only access, read/write access, absolute, relocatable, etc.) that is the unit of virtual memory allocation for an image.

immediate mode In immediate mode addressing, the PC is used as the register in autoincrement mode addressing.

index The structure which allows retrieval of records in an indexed file by key value. See key (indexed files).

index file The file on a Files-11 volume that contains the access information for all files on the volume and enables the operating system to identify and access the volume.

index file bit map A table in the index file of a Files-11 volume that indicates which file headers are in use.

index register A register used to contain an address offset.

indexed addressing mode In indexed mode addressing, two registers are used to determine the actual instruction operand: an index register and a base operand specifier. The contents of the index register are used as an index (offset) into a table or array. The base operand specifier supplies the base address of the array (the base operand address or BOA). The address of the actual operand is calculated by multiplying the contents of the index register by the size (in bytes) of the actual operand and adding the result to the base operand address. The addressing modes resulting from index mode addressing are formed by adding the suffix "indexed" to the addressing mode of the base operand specifier: register deferred indexed, autoincrement indexed, autoincrement deferred indexed (or absolute indexed), autodecrement indexed, displacement indexed, and displacement deferred indexed.

indexed file organization A file organization which allows random retrieval of records by key values and sequential retrieval of records in sorted order by key value. See key (indexed files).

indirect command file See command procedure.

input stream The source of commands and data. One of: the user's terminal, the batch stream, or an indirect command file.

instruction buffer A buffer in the processor used to contain bytes of the instruction currently being decoded and to pre-fetch instructions in the instruction stream. The control logic continously fetches data from memory to keep the buffer full.

interleaving Assigning consecutive physical memory addresses alternately between two memory controllers.

interlocked The property of a read followed by a write to the same datum with no possibility of an intervening reference by a second processor or I/O device. Examples are the Branch on Bit Interlocked and Add Aligned Word Interlocked instructions.

interprocess communication facility A common event flag, mailbox, or global section used to pass information between two or more processes.

interrecord gap A blank space deliberately placed between data records on the recording surface of a magnetic tape.

interrupt An event other than an exception or branch, jump, case, or call instruction that changes the normal flow of instruction execution. Interrupts are generally external to the process executing when the interrupt occurs. See also device interrupt, software interrupt, and urgent interrupt.

interrupt priority level (IPL) The interrupt level at which the processor executes when an interrupt is generated. There are 31 possible interrupt priority levels. IPL 1 is lowest, 31 highest. The levels arbitrate contention for processor service. For example, a device cannot interrupt the processor if the processor is currently executing at an interrupt priority level greater than the interrupt priority level of the device's interrupt service routine.

interrupt service routine The routine executed when a device interrupt occurs.

interrupt stack The system-wide stack used when executing in interrupt service context. At any time, the processor is either in a process context executing in user, supervisor, executive or kernel mode, or in system-wide interrupt service context operating with kernel privileges, as indicated by the interrupt stack and current mode bits in the PSL. The interrupt stack is not context switched.

interrupt stack pointer The stack pointer for the interrupt stack. Unlike the stack pointers for process context stacks, which are stored in the hardware PCB, the interrupt stack pointer is stored in an internal register.

interrupt vector See vector.

I/O driver See driver.

I/O function An I/O operation that is interpreted by the operating system and typically results in one or more physical I/O operations.

I/O function code A 6-bit value specified in a Queue I/O Request system service that describes the particular I/O operation to be performed (e.g., read, write, rewind).

I/O function modifier A 10-bit value specified in a Queue I/O Request system service that modifies an I/O function code (e.g., read terminal input no echo).

I/O lockdown The state of a page such that it cannot be paged or swapped out of memory until any I/O in progress to that page is completed.

I/O rundown An operating system function in which the system cleans up any I/O in progress when an image exits.

I/O space The region of physical address space that contains the configuration registers, and device control/status and data registers.

I/O status block A data structure associated with the Queue I/O Request system service. This service optionally returns a status code, number of bytes transferred, and device- and function-dependent in-

formation in an I/O status block. It is not returned from the service call, but filled in when the I/O request completes.

job 1. A job is the accounting unit equivalent to a process and the collection of all the subprocesses, if any, that it and its subprocesses create. Jobs are classified as batch and interactive. For example, the job controller creates an interactive job to handle a user's requests when the user logs onto the system and it creates a batch job when the symbiont manager passes a command input file to it. 2. A print job.

job controller The system process that establishes a job's process context, starts a process running the LOGIN image for the job, maintains the accounting record for the job, manages symbionts, and terminates a process and its subprocesses.

job queue A list of files that a process has supplied for processing by a specific device, for example, a line printer.

kernel mode The most privileged processor access mode (mode 0). The operating system's most privileged services, such as I/O drivers and the pager, run in kernel mode.

key
indexed files: A character string, a packed decimal number, a 2- or 4-byte unsigned binary number, or a 2- or 4-byte signed integer within each data record in an indexed file. You define the length and location within the records; VAX-11 RMS uses the key to build an index. See primary key, alternate key, and random access by key value.

relative files: The relative record number of each data record in a data file; VAX-11 RMS uses the relative record numbers to identify and access data records in a relative file in random access mode. See relative record number.

lexical function A command language construct that the command interpreter evaluates and substitutes before it performs expression analysis on a command string. Lexical functions return information about the current process, such as UIC or default directory; and about character strings, such as length or substring locations.

librarian A program that allows the user to create, update, modify, list, and maintain object library, image library, and assembler macro library files.

library file A direct access file containing one or more modules of the same module type.

limit The size or number of given items requiring system resources (such as mailboxes, locked pages, I/O requests, open files, etc.) that a job is allowed to have at any one time during execution, as specified

by the system manager in the user authorization file. See also quota.

line number A number used to identify a line of text in a file processed by a text editor.

linker A program that reads one or more object files created by language processors and produces an executable image file, a shareable image file, or a system image file.

linking The resolution of external references between object modules used to create an image, the acquisition of referenced library routines, service entry points, and data for the image, and the assignment of virtual addresses to components of an image.

literal mode In literal mode addressing, the instruction operand is a constant whose value is expressed in a 6-bit field of the instruction. If the operand data type is byte, word, longword, quadword, or octaword, the operand is zero-extended and can express values in the range 0 through 63 (decimal). If the operand data type is F_, D_, G_, or H_floating, the 6-bit field is composed of two 3-bit fields, one for the exponent and the other for the fraction. The operand is extended to F_, D_, G_, or H_floating format.

locality See program locality.

local symbol A symbol meaningful only to the module that defines it. Symbols not identified to a language processor as global symbols are considered to be local symbols. A language processor resolves (matches references with definitions) local symbols. They are not known to the linker and cannot be made available to another object module. They can, however, be passed through the linker to the symbolic debugger. Contrast with global symbol.

locate mode Technique used for a record input operation in which the data records are not copied from the I/O buffer. See move mode.

locking a page in memory Making a page in an image ineligible for either paging or swapping. A page stays locked in memory until it is specifically unlocked.

locking a page in the working set Making a page in an image ineligible for paging out of the working set for the image. The page can be swapped when the process is swapped. A page stays locked in a working set until it is specifically unlocked.

logical block number A number used to identify a block on a mass storage device. The number is a volume-relative address rather than its physical (device-oriented) address or its virtual (file-relative) address. The blocks that constitute the volume are labeled sequentially starting with logical block 0.

logical I/O function A set of I/O operations (e.g., read and write logical block) that allow restricted direct access to device level I/O operations using logical block addresses.

logical name A user-specified name for any portion or all of a file specification. For example, the logical name INPUT can be assigned to a terminal device from which a program reads data entered by a user. Logical name assignments are maintained in logical name tables for each process, each group, and the system. A logical name can be created and assigned a value permanently or dynamically.

logical name table A table that contains a set of logical names and their equivalence names for a particular process, a particular group, or the system.

logical I/O functions A set of I/O functions that allow restricted direct access to device level I/O operations.

logical record A group of related fields treated as a unit.

longword Four contiguous bytes (32 bits) starting on an addressable byte boundary. Bits are numbered from right to left, 0 through 31. The address of the longword is the address of the byte containing bit 0. When interpreted arithmetically, a longword is a 2's complement integer with significance increasing from bit 0 to bit 30. When interpreted as a signed integer, bit 31 is the sign bit. The value of the signed integer is in the range −2,147,483,648 to 2,147,483,647. When interpreted as an unsigned integer, significance increases from bit 0 to bit 31. The value of the unsigned integer is in the range 0 through 4,294,967,295.

macro A statement that requests a language processor to generate a predefined set of instructions.

mailbox A software data structure that is treated as a record-oriented device for general interprocess communication. Communication using a mailbox is similar to other forms of device-independent I/O. Senders perform a write to a mailbox, the receiver performs a read from that mailbox. Some system-wide mailboxes are defined: the error logger and OPCOM read from system-wide mailboxes.

main memory See physical memory.

mapping window A subset of the retrieval information for a file that is used to translate virtual block numbers to logical block numbers.

mass storage device A device capable of reading and writing data on mass storage media such as a disk pack or a magnetic tape reel.

member number The second number in a user identification code that uniquely identifies that code.

memory management The system functions that include the hardware's page mapping and protection and the operating system's image activator and pager.

Memory Mapping Enable (MME) A bit in a processor register that governs address translation.

modify access type The specified operand of an instruction or procedure is read, and is potentially modified and written, during that instruction's or procedure's execution.

module 1. A portion of a program or program library, as in a *source module*, *object module*, or *image module*. 2. A board, usually made of plastic covered with an electrical conductor, on which logic devices (such as transistors, resistors, and memory chips) are mounted, and circuits connecting these devices are etched, as in a *logic module*.

Monitor Console Routine (MCR) The command interpreter in an RSX-11 system.

mount a volume 1. To logically associate a volume with the physical unit on which it is loaded (an activity accomplished by system software at the request of an operator). 2. To load or place a magnetic tape or disk pack on a drive and place the drive online (an activity accomplished by a system operator).

move mode Technique used for a record transfer in which the data records are copied between the I/O buffer and your program buffer for calculations or operations on the record. See locate mode.

mutex A semaphore that is used to control exclusive access to a region of code that can share a data structure or other resource. The mutex (mutual exclusion) semaphore ensures that only one process at a time has access to the region of code.

name block (NAM) An RMS user data structure that contains supplementary information used in parsing file specifications.

native image An image whose instructions are executed in native mode.

native mode The processor's primary execution mode in which the programmed instructions are interpreted as byte-aligned, variable-length instructions that operate on byte, word, longword, quadword, and octaword integer, F_, D_, G_ and H_floating format, character string, packed decimal, and variable-length bit field data. The instruction execution mode other than compatibility mode.

network A collection of interconnected individual computer systems.

nibble The low-order or high-order four bits of a byte.

node An individual computer system in a network.

null process A small system process that is the lowest priority process in the system and takes one entire priority class. One function of the null process is to accumulate idle processor time.

numeric string A contiguous sequence of bytes representing up to 31 decimal digits (one per byte) and possibly a sign. The numeric string is specified by its lowest addressed location, its length, and its sign representation.

object module The binary output of a language processor such as the assembler or a compiler, which is used as input to the linker.

object time system (OTS) See Run Time Procedure Library.

octaword Sixteen contiguous bytes (128 bits) starting on an addressable byte boundary. Bits are numbered from right to left, 0 to 127. An octaword is identified by the address of the byte containing the low-order bit (bit 0).

offset A fixed displacement from the beginning of a data structure. System offsets for items within a data structure normally have an associated symbolic name used instead of the numeric displacement. Where symbols are defined, programmers always reference the symbolic names for items in a data structure instead of using the numeric displacement.

opcode The pattern of bits within an instruction that specify the operation to be performed.

operand specifier The pattern of bits in an instruction that indicate the addressing mode, a register and/or displacement, which, taken together, identify an instruction operand.

operand specifier type The access type and data type of an instruction's operand(s). For example, the test instructions are of read access type, since they only read the value of the operand. The operand can be of byte, word, or longword data type, depending on whether the opcode is for the TSTB (test byte), TSTW (test word), or TSTL (test longword) instruction.

Operator Communication Manager (OPCOM) A system process that is always active. OPCOM receives input from a process that wants to inform an operator of a particular status or condition, passes a message to the operator, and tracks the message.

operator's console Any terminal identified as a terminal attended by a system operator.

owner In the context "system, owner, group, world," an owner is the particular member (of a group) to which a file, global section, mailbox, or event flag cluster belongs.

owner process The process (with the exception of the job controller) or subprocess that created a subprocess.

packed decimal A method of representing a decimal number by storing a pair of decimal digits in one byte, taking advantage of the fact that only four bits are required to represent the numbers 0 through 9.

packed decimal string A contiguous sequence of up to 16 bytes interpreted as a string of nibbles. Each nibble represents a digit except the low-order nibble of the highest addressed byte, which represents the sign. The packed decimal string is specified by its lowest addressed location and the number of digits.

page 1. A set of 512 contiguous byte locations used as the unit of memory mapping and protection. 2. The data between the beginning of file and a page marker, between two markers, or between a marker and the end of a file.

page fault An exception generated by a reference to a page which is not mapped into a working set.

page fault cluster size The number of pages read in on a page fault.

page frame number (PFN) The address of the first byte of a page in physical memory. The high-order 21 bits of the physical address of the base of a page.

page marker A character or characters (generally a form feed) that separates pages in a file that is processed by a text editor.

pager A set of kernel mode procedures that executes as the result of a page fault. The pager makes the page for which the fault occurred available in physical memory so that the image can continue execution. The pager and the image activator provide the operating system's memory management functions.

page table entry (PTE) The data structure that identifies the location and status of a page of virtual address space. When a virtual page is in memory, the PTE contains the page frame number needed to map the virtual page to a physical page. When it is not in memory, the page table entry contains the information needed to locate the page on secondary storage (disk).

paging The action of bringing pages of an executing process into physical memory when referenced. When a process executes, all of its pages are said to reside in virtual memory. Only the actively used pages, however, need to reside in physical memory. The remaining

pages can reside on disk until they are needed in physical memory. In this system, a process is paged only when it references more pages than it is allowed to have in its working set. When the process refers to a page not in its working set, a page fault occurs. This causes the operating system's pager to read in the referenced page if it is on disk (and, optionally, other related pages depending on a cluster factor), replacing the least recently faulted pages as needed. A process pages only against itself.

parameter See command parameter.

per-process address space See process address space.

physical address The address used by hardware to identify a location in physical memory or on directly addressable secondary storage devices such as a disk. A physical memory address consists of a page frame number and the number of a byte within the page. A physical disk block address consists of a cylinder or track and sector number.

physical address space The set of all possible 30-bit physical addresses that can be used to refer to locations in memory (memory space) or device registers (I/O space).

physical block A block on a mass storage device referred to by its physical (device-oriented) address rather than a logical (volume-relative) or virtual (file-relative) address.

physical I/O functions A set of I/O functions that allow access to all device level I/O operations except maintenance mode.

physical memory The memory modules connected to the SBI that are used to store: 1) instructions that the processor can directly fetch and execute, and 2) any other data that a processor is instructed to manipulate. Also called main memory.

position-dependent code Code that can execute properly only in the locations in virtual address space that are assigned to it by the linker.

position-independent code Code that can execute properly without modification wherever it is located in virtual address space, even if its location is changed after it has been linked. Generally, this code uses addressing modes that form an effective address relative to the PC.

primary key The mandatory key within the data records of an indexed file; used by VAX-11 RMS to determine the placement of records within the file and to build the primary index. See key (indexed files) and alternate key.

primary vector A location that contains the starting address of a condition handler to be executed when an exception condition occurs. If a primary vector is declared, that condition handler is the first handler to be executed.

private section An image section of a process that is not shareable among processes. See also global section.

privilege See process privilege, user privilege, and image privilege.

privileged instructions In general, any instructions intended for use by the operating system or privileged system programs. In particular, instructions that the processor will not execute unless the current access mode is kernel mode (e.g., HALT, SVPCTX, LDPCTX, MTPR, and MFPR).

procedure 1. A routine entered via a Call instruction. 2. See command procedure.

process The basic entity scheduled by the system software that provides the context in which an image executes. A process consists of an address space and both hardware and software context.

process address space See process space.

process context The hardware and software contexts of a process.

process control block (PCB) A data structure used to contain process context. The hardware PCB contains the hardware context. The software PCB contains the software context, which includes a pointer to the hardware PCB.

process header A data structure that contains the hardware PCB, accounting and quota information, process section table, working set list, and the page tables defining the virtual layout of the process.

process header slots That portion of the system address space in which the system stores the process headers for the processes in the balance set. The number of process header slots in the system determines the number of processes that can be in the balance set at any one time.

process identification (PID) The operating system's unique 32-bit binary value assigned to a process.

process I/O segment That portion of a process control region that contains the process permanent RMS internal file access block for each open file, and the I/O buffers, including the command interpreter's command buffer and command descriptors.

process name A 1- to 15-character ASCII string that can be used to identify processes executing under the same group number.

processor register A part of the processor used by the operating system software to control the execution states of the computer system. They include the system base and length registers, the program and control region base and length registers, the system control block base register, the software interrupt request register, and many more.

Processor Status Longword (PSL) A system programmed processor register consisting of a word of privileged processor status and the PSW. The privileged processor status information includes: the current IPL (interrupt priority level), the previous access mode, the current access mode, the interrupt stack bit, the trace fault pending bit, and the compatibility mode bit.

Processor Status Word (PSW) The low-order word of the Processor Status Longword. Processor status information includes: the condition codes (carry, overflow, zero, negative), the arithmetic trap enable bits (integer overflow, decimal overflow, floating underflow), and the trace enable bit.

process page tables The page tables used to describe process virtual memory.

process priority The priority assigned to a process for scheduling purposes. The operating system recognizes 32 levels of process priority, where 0 is low and 31 high. Levels 16 through 31 are used for time-critical processes. The system does not modify the priority of a time-critical process (although the system manager or process itself may). Levels 0 through 15 are used for normal processes. The system may temporarily increase the priority of a normal process based on the activity of the process.

process privileges The privileges granted to a process by the system, which are a combination of user privileges and image privileges. They include, for example, the privilege to: affect other processes associated with the same group as the user's group, affect any process in the system regardless of UIC, set process swap mode, create permanent event flag clusters, create another process, create a mailbox, and perform direct I/O to a file-structured device.

process section See private section.

process space The lowest-addressed half of virtual address space, where per-process instructions and data reside. Process space is divided into a program region and a control region.

Program Counter (PC) General register 15 (R15). At the beginning of an instruction's execution, the PC normally contains the address of

a location in memory from which the processor will fetch the next instruction it will execute.

program locality A characteristic of a program that indicates how close or far apart the references to locations in virtual memory are over time. A program with a high degree of locality does not refer to many widely scattered virtual addresses in a short period of time.

programmer number See member number.

program region The lowest-addressed half of process address space (P0 space). The program region contains the image currently being executed by the process and other user code called by the image.

Program region Base Register (P0BR) The processor register, or its equivalent in a hardware process control block, that contains the base virtual address of the page table entry for virtual page number 0 in a process program region.

Program region Length Register (P0LR) The processor register, or its equivalent in a hardware process control block, that contains the number of entries in the page table for a process program region.

program section (psect) A portion of a program with a given protection and set of storage management attributes. Program sections that have the same attributes are gathered together by the linker to form an image section.

project number See group number or account number.

pure code See re-entrant code.

quadword Eight contiguous bytes (64 bits) starting on an addressable byte boundary. Bits are numbered from right to left, 0 to 63. A quadword is identified by the address of the byte containing the low-order bit (bit 0). When interpreted arithmetically, a quadword is a 2's complement integer with significance increasing from bit 0 to bit 62. Bit 63 is used as the sign bit. The value of the integer is in the range -2^{63} to $2^{63} - 1$.

qualifier A portion of a command string that modifies a command verb or command parameter by selecting one of several options. A qualifier, if present, follows the command verb or parameter to which it applies and is in the format: "/qualifier:option." For example, in the command string "PRINT filename/COPIES:3," the COPIES qualifier indicates that the user wants three copies of a given file printed.

queue 1. n. A circular, doubly-linked list. See system queues. v. To make an entry in a list or table, perhaps using the INSQUE instruction. 2. See job queue.

queue priority The priority assigned to a job placed in a spooler queue or a batch queue.

quota The total amount of a system resource, such as CPU time, that a job is allowed to use in an accounting period, as specified by the system manager in the user authorization file. See also limit.

random access by key Indexed files only: Retrieval of a data record in an indexed file by either a primary or alternate key within the data record. See key (indexed files).

random access by record's file address The retrieval of a record by its unique address, which is provided to the program by RMS. This method of access is the only means of randomly accessing a sequentially organized file containing variable length records.

random access by relative record number Retrieval of a record by its relative record number. See relative record number. For relative files, random access by relative record number is synonymous with random access by key. See random access by key (relative files only).

read access type An instruction or procedure operand attribute indicating that the specified operand is only read during instruction or procedure execution.

record A set of related data that your program treats as a unit.

record access block (RAB) An RMS user data structure that represents a request for a record access stream. A RAB relates to operations on the records within a file, such as UPDATE, DELETE, or GET.

record access mode The method used in RMS for retrieving and storing records in a file. One of three methods: sequential, random, and record's file address.

record blocking The technique of grouping multiple rcords into a single block. On magnetic tape, an IRG is placed after the block rather than after each record. This technique reduces the number of I/O transfers required to read or write the data; and, in addition (for magnetic tape), increases the amount of usable storage area. Record blocking also applies to disk files.

record cell A fixed-length area in a relative file that can contain a record. The concept of fixed-length record cells lets VAX-11 RMS directly calculate the record's actual position in the file.

record format The way a record physically appears on the recording surface of the storage medium. The record format defines the method for determining record length.

record length The size of a record; that is, the number of bytes in a record.

record locking A facility that prevents access to a record by more than one record stream or process until the initiating record stream or process releases the record.

Record Management Services A set of operating system procedures that are called by programs to process files and records within files. RMS allows programs to issue READ and WRITE requests at the record level (record I/O) as well as read and write blocks (block I/O). RMS is an integral part of the system software. RMS procedures run in executive mode.

record-oriented device A device such as a terminal, line printer, or card reader, on which the largest unit of data a program can access in one I/O operation is the device's physical record.

record's file address The unique address of a record in a file, which is returned by RMS whenever a record is accessed, that allows records in disk files to be access randomly regardless of file organization. This address is valid only for the life of the file. If an indexed file is reorganized, then the RFA of each record will typically change.

re-entrant code Code that is never modified during execution. It is possible to let many users share the same copy of a procedure or program written as re-entrant code.

register A storage location in hardware logic other than main memory. See also general register, processor register, and device register.

register deferred indexed mode An indexed addressing mode in which the base operand specifier uses register deferred mode addressing.

register deferred mode In register deferred mode addressing, the contents of the specified register are used as the address of the actual instruction operand.

register mode In register mode addressing, the contents of the specified register are used as the actual instruction operand.

relative file organization The arrangement of records in a file where each record occupies a cell of equal length within a bucket. Each cell is assigned a successive number, called a relative record number, which represents the cell's position relative to the beginning of the file.

relative record number An identification number used to specify the position of a record cell relative to the beginning of the file; used as the key during random access by key mode to relative files.

resource A physical part of the computer system such as a device or memory, or an interlocked data structure such as a mutex. Quotas and limits control the use of physical resources.

resource wait mode An execution state in which a process indicates that it will wait until a system resource becomes available when it issues a service request requiring a resource. If a process wants notification when a resource is not available, it can disable resource wait mode during program execution.

return status code See status code.

RMS-11 A set of routines which is linked with compatibility mode programs, and provides similar functional capabilities to VAX-11 RMS. The file organizations and record formats used by RMS-11 are identical to those of VAX-11 RMS.

Run Time Procedure Library The collection of procedures available to native mode images at run time. These library procedures (such as trigonometric functions, etc.) are common to all native mode images, regardless of the language processor used to compile or assemble the program.

scatter/gather The ability to transfer in one I/O operation data from discontiguous pages in memory to contiguous blocks on disk, or data from contiguous blocks on disk to discontiguous pages in memory.

secondary storage Random access mass storage.

secondary vector A location that identifies the starting address of a condition handler to be executed when a condition occurs and the primary vector contains zero or the handler to which the primary vector points chooses not to handle the condition.

section A portion of process virtual memory that has common memory management attributes (protection, access, cluster factor, etc.). It is created from an image section, a disk file, or as the result of a Create Virtual Address Space system service. See global section, private section, image section, and program section.

self-relative queue A circularly linked list whose forward and backward links use the address of the entry in which they occur as the base address for the link displacement to the linked entry. Contrast with absolute addresses used to link a queue.

sequential file organization A file organization in which records appear in the order in which they were originally written. The records can be fixed length or variable length.

sequential record access mode Record storage or retrieval which starts at a designated point in the file and continues in one-after-the-

other fashion through the file. That is, records are accessed in the order in which they physically appear in the file.

shareable image An image that has all of its internal references resolved, but which must be linked with an object module(s) to produce an executable image. A sharable image cannot be executed. A shareable image file can be used to contain a library of routines. A shareable image can be used to create a global section by the system manager.

shell process A predefined process that the job initiator copies to create the minimum context necessary to establish a process.

signal 1. An electrical impulse conveying information. 2. The software mechanism used to indicate that an exception condition was detected.

slave terminal A terminal from which it is not possible to issue commands to the command interpreter. A terminal assigned to application software.

small process A system process that has no control region in its virtual address space and has an abbreviated context. Examples are the working set swapper and the null process. A small process is scheduled in the same manner as user processes, but must remain resident during its execution.

software context The context maintained by the operating system that describes a process. See software process control block (PCB).

software interrupt An interrupt generated on interrupt priority level 1 through 15, which can be requested only by software.

software process control block (PCB) The data structure used to contain a process' software context. The operating system defines a software PCB for every process when the process is created. The software PCB includes the following kinds of information about the process: current state; storage address if it is swapped out of memory; unique identification of the process, and address of the process header (which contains the hardware PCB). The software PCB resides in system region virtual address space. It is not swapped with a process.

software priority See process priority and queue priority.

spooling output spooling: The method by which output to a low-speed peripheral device (such as a line printer) is placed into queues maintained on a high-speed device (such as disk) to await transmission to the low-speed device. Input spooling: the method by which input from a low-speed peripheral (such as the card reader) is placed into queues maintained on a high-speed device (such as disk) to await transmission to a job processing that input.

spool queue The list of files supplied by processes that are to be processed by a symbiont. For example, a line printer queue is a list of files to be printed on the line printer.

stack An area of memory set aside for temporary storage, or for procedure and interrupt service linkages. A stack uses the last-in, first-out concept. As items are added to ("pushed on") the stack, the stack pointer decrements. As items are retrieved from ("popped off") the stack, the stack pointer increments.

stack frame A standard data structure built on the stack during a procedure call, starting from the location addressed by the FP to lower addresses, and popped off during a return from procedure. Also called call frame.

stack pointer General register 14 (R14). SP contains the address of the top (lowest address) of the processor-defined stack. Reference to SP will access one of the five possible stack pointers (kernel, executive, supervisor, user, or interrupt) depending on the value in the current mode and interrupt stack bits in the Processor Status Longword (PSL).

state queue A list of processes in a particular processing state. The scheduler uses state queues to keep track of processes' eligibility to execute. They include: processes waiting for a common event flag, suspended processes, and executable processes.

status code A longword value that indicates the success or failure of a specific function. For example, system services always return a status code in R0 upon completion.

store through See write through.

strong definition Definition of a global symbol that is explicitly available for reference by modules linked with the module in which the definition occurs. The linker always lists a global symbol with a strong definition in the symbol portion of the map. The librarian always includes a global symbol with a strong definition in the global symbol table of a library.

strong reference A reference to a global symbol in an object module that requests the linker to report an error if it does not find a definition for the symbol during linking. If a library contains the definition, the linker incorporates the library module defining the global symbol into the image containing the strong reference.

subprocess A subsidiary process created by another process. The process that creates a subprocess is its owner. A subprocess receives resource quotas and limits from its owner. When an owner process is

removed from the system, all its subprocesses (and their subprocesses) are also removed.

supervisor mode The third most privileged processor access mode (mode 2). The operating system's command interpreter runs in supervisor mode.

suspension A state in which a process is inactive, but known to the system. A suspended process becomes active again only when another process requests the operating system to resume it. Contrast with hibernation.

swap mode A process execution state that determines the eligibility of a process to be swapped out of the balance set. If process swap mode is disabled, the process working set is locked in the balance set.

swapping The method for sharing memory resources among several processes by writing an entire working set to secondary storage (swap out) and reading another working set into memory (swap in). For example, a process' working set can be written to secondary storage while the process is waiting for I/O completion on a slow device. It is brought back into the balance set when I/O completes. Contrast with paging.

switch See (command) qualifier.

symbiont A full process that transfers record-oriented data to or from a mass storage device. For example, an input symbiont transfers data from card readers to disks. An output symbiont transfers data from disks to line printers.

symbiont manager The function (in the system process called the job controller) that maintains spool queues, and dynamically creates symbiont processes to perform the necessary I/O operations.

symbol See local symbol, global symbol, and universal global symbol.

Synchronous Backplane Interconnect (SBI) The part of the hardware that interconnects the processor, memory controllers, MASSBUS adapters, and the UNIBUS adapter.

synchronous record operation A mode of record processing in which a user program issues a record read or write request and then waits until that request is fulfilled before continuing to execute.

system In the context "system, owner, group, world," the system refers to the group numbers that are used by operating system and its controlling users, the system operators and system manager.

system address space See system space and system region.

System Base Register (SBR) A processor register containing the physical address of the base of the system page table.

system buffered I/O An I/O operation, such as terminal or mailbox I/O, in which an intermediate buffer from the system buffer pool is used instead of a process-specified buffer. Contrast with direct I/O.

System Control Block (SCB) The data structure in system space that contains all the interrupt and exception vectors known to the system.

System Control Block Base register (SCBB) A processor register containing the base address of the system control block.

system device The random access mass storage device unit on which the volume containing the operating system software resides.

system dynamic memory Memory reserved for the operating system to allocate as needed for temporary storage. For example, when an image issues an I/O request, system dynamic memory is used to contain the I/O request packet. Each process has a limit on the amount of system dynamic memory that can be allocated for its use at one time.

System Identification Register A processor register which contains the processor type and serial number.

system image The image that is read into memory from secondary storage when the system is started up.

System Length Register (SLR) A processor register containing the length of the system page table in longwords, that is, the number of page table entries in the system region page table.

System Page Table (SPT) The data structure that maps the system region virtual addresses, including the addresses used to refer to the process page tables. The System Page Table (SPT) contains one Page Table Entry (PTE) for each page of system region virtual memory. The physical base address of the SPT is contained in a register called the SBR.

system process A process that provides system-level functions. Any process that is part of the operating system. See also small process, fork process.

system programmer A person who designs and/or writes operating systems, or who designs and writes procedures or programs that provide general purpose services for an application system.

system queue A queue used and maintained by operating system procedures. See also state queues.

system region The third quarter of virtual address space. The lowest-addressed half of system space. Virtual addresses in the system region are shareable between processes. Some of the data structures mapped by system region virtual addresses are: system entry vectors, the System Control Block (SCB), the System Page Table (SPT), and process page tables.

system services Procedures provided by the operating system that can be called by user processes.

system space The highest-addressed half of virtual address space. See also system region.

system virtual address A virtual address identifying a location mapped by an address in system space.

system virtual space See system space.

task An RSX-11/IAS term for a process and image bound together.

terminal The general name for those peripheral devices that have keyboards and video screens or printers. Under program control, a terminal enables people to type commands and data on the keyboard and receive messages on the video screen or printer. Examples of terminals are the LA36 DECwriter hard-copy terminal and VT100 video display terminal.

time-critical process A process assigned to a software priority level between 16 and 31, inclusive. The scheduling priority assigned to a time-critical process is never modified by the scheduler, although it can be modified by the system manager or process itself.

timer A system fork process that maintains the time of day and the date. It also scans for device timeouts and performs time-dependent scheduling upon request.

track A collection of blocks at a single radius on one recording surface of a disk.

transfer address The address of the location containing a program entry point (the first instruction to execute).

translation buffer An internal processor cache containing translations for recently used virtual addresses.

trap An exception condition that occurs at the end of the instruction that caused the exception. The PC saved on the stack is the address of the next instruction that would normally have been executed. All software can enable and disable some of the trap conditions with a single instruction.

trap enables Three bits in the Processor Status Word that control the processor's action on certain arithmetic exceptions.

two's complement A binary representation for integers in which a negative number is one greater than the bit complement of the positive number.

two-way associative cache A cache organization which has two groups of directly mapped blocks. Each group contains several blocks for each index position in the cache. A block of data from main memory can go into any group at its proper index position. A two-way associative cache is a compromise between the extremes of fully associative and direct mapping cache organizations that takes advantage of the features of both.

type ahead A terminal handling technique in which the user can enter commands and data while the software is processing a previously entered command. The commands typed ahead are not echoed on the terminal until the command processor is ready to process them. They are held in a type ahead buffer.

unit record device A device such as a card reader or line printer.

universal global symbol A global symbol in a shareable image that can be used by modules linked with that shareable image. Universal global symbols are typically a subset of all the global symbols in a shareable image. When creating a shareable image, the linker ensures that universal global symbols remain available for reference after symbols have been resolved.

unwind the call stack To remove call frames from the stack by tracing back through nested procedure calls using the current contents of the FP register and the FP register contents stored on the stack for each call frame.

urgent interrupt An interrupt received on interrupt priority levels 24 through 31. These can be generated only by the processor for the interval clock, serious errors, and power fail.

user authorization file A file containing an entry for every user that the system manager authorizes to gain access to the system. Each entry identifies the user name, password, default account, User Identification Code (UIC), quotas, limits, and privileges assigned to individuals who use the system.

user environment test package (UETP) A collection of routines that verify that the hardware and software systems are complete, properly installed, and ready to be used.

User File Directory (UFD) See directory.

User Identification Code (UIC) The pair of numbers assigned to users and to files, global sections, common event flag clusters, and mailboxes that specifies the type of access (read and/or write access, and in the case of files, execute and/or delete access) available to the owners, group, world, and system. It consists of a group number and a member number separated by a comma.

user mode The least privileged processor access mode (mode 3). User processes and the Run Time Library procedures run in user mode.

user name The name that a person types on a terminal to log on to the system.

user number See member number.

user privileges The privileges granted a user by the system manager. See process privileges.

utility A program that provides a set of related general purpose functions, such as a program development utility (an editor, a linker, etc.), a file management utility (file copy or file format translation program), or operations management utility (disk backup/restore, diagnostic program, etc.).

value return registers The general registers R0 and R1 used by convention to return function values. These registers are not preserved by any called procedures. They are available as temporary registers to any called procedure. All other registers (R2, R3,...,R11, AP, FP, SP, PC) are preserved across procedure calls.

variable-length bit field A set of 0 to 32 contiguous bits located arbitrarily with respect to byte boundaries. A variable bit field is specified by four attributes: 1) the address A of a byte, 2) the bit position P of the starting location of the bit field with respect to bit 0 of the byte at address A, 3) the size, in bits, of the bit field, and 4) whether the field is signed or unsigned.

variable-length record format A file format in which records are not necessarily the same length.

variable with fixed-length control record format Property of a file in which records of variable-length contain an additional fixed control area capable of storing data that may have no bearing on the other contents of the record. Variable with fixed-length control record format is not applicable to indexed files.

VAX-11 Record Management Services (VAX-11 RMS) The file and record access subsystem of the VAX/VMS operating system for VAX. VAX-11 RMS helps your application program process records within

files, thereby allowing interaction between your application program and its data.

vector 1. A interrupt or exception vector is a storage location known to the system that contains the starting address of a procedure to be executed when a given interrupt or exception occurs. The system defines separate vectors for each interrupting device controller and for classes of exceptions. Each system vector is a longword. 2. For exception handling, users can declare up to two software exception vectors (primary and secondary) for each of the four access modes. Each vector contains the address of a condition handler. 3. A one-dimensional array.

version number 1. The field following the file type in a file specification. It begins with a period (.) and is followed by a number which generally identifies it as the latest file created of all files having the identical file specification but for version number. 2. The number used to identify the revision level of program.

virtual address A 32-bit integer identifying a byte "location" in virtual address space. The memory management hardware translates a virtual address to a physical address. The term "virtual address" may also refer to the address used to identify a virtual block on a mass storage device.

virtual address space The set of all possible virtual addresses that an image executing in the context of a process can use to identify the location of an instruction or data. The virtual address space seen by the programmer is a linear array of 4,294,967,296 (2^{32}) byte addresses.

virtual block A block on a mass storage device referred to by its file-relative address rather than its logical (volume-oriented) or physical (device-oriented) address. The first block in a file is always virtual block 1.

virtual I/O functions A set of I/O functions that must be interpreted by an ancillary control process.

virtual memory The set of storage locations in physical memory and on disk that are referred to by virtual addresses. From the programmer's viewpoint, the secondary storage locations appear to be locations in physical memory. The size of virtual memory in any system depends on the amount of physical memory available and the amount of disk storage used for non-resident virtual memory.

virtual page number The virtual address of a page of virtual memory.

volume
Disks: An ordered set of 512-byte blocks. The basic medium that carries a Files-11 structure.

Magnetic tape: A reel of magnetic tape, which may contain a part of a file, a complete file, or more than one file.

volume set A collection of related volumes.

wait To become inactive. A process enters a process wait state when the process suspends itself, hibernates, or declares that it needs to wait for an event, resource, mutex, etc.

wake To activate a hibernating process. A hibernating process can be awakened by another process or by the timer process, if the hibernating process or another process scheduled a wake-up call.

weak definition Definition of a global symbol that is not explicitly available for reference by modules linked with the module in which the definition occurs. The librarian does not include a global symbol with a weak definition in the global symbol table of a library. Weak definitions are often used when creating libraries to identify those global symbols that are needed only if the module containing them is otherwise linked with a program.

weak reference A reference to a global symbol that requests the linker not to report an error or to search the default library's global symbol table to resolve the reference if the definition is not in the modules explicitly supplied to the linker. Weak references are often used when creating object modules to identify those global symbols that may not be needed at run time.

wild card A symbol, such as an asterisk, that is used in place of a file name, file type, directory name, or version number in a file specification to indicate "all" for the given field.

window See mapping window.

word Two contiguous bytes (16 bits) starting on an addressable byte boundary. Bits are numbered from the right, 0 through 15. A word is identified by the address of the byte containing bit 0. When interpreted arithmetically, a word is a 2's complement integer with significance increasing from bit 0 to bit 14. If interpreted as a signed integer, bit 15 is the sign bit. The value of the integer is in the range −32,768 to 32,767. When interpreted as an unsigned integer, significance increases from bit 0 through bit 15 and the value of the unsigned integer is in the range 0 through 65,535.

working set The set of pages in process space to which an executing process can refer without incurring a page fault. The working set

must be resident in memory for the process to execute. The remaining pages of that process, if any, are either in memory and not in the process working set or they are on secondary storage.

working set swapper A system process that brings process working sets into the balance set and removes them from the balance set.

world In the context "system, owner, group, world," world refers to all users, including the system operators, the system manager, and users both in an owner's group and in any other group.

write access type The specified operand of an instruction or procedure is written only during that instruction's or procedure's execution.

write allocate A cache management technique in which cache is allocated on a write miss as well as on the usual read miss.

write back A cache management technique in which data from a write operation to cache are copied into main memory only when the data in cache must be overwritten. This results in temporary inconsistencies between cache and main memory. Contrast with write through.

write through A cache management technique in which data from a write operation are copied in both cache and main memory. Cache and main memory data are always consistent. Contrast with write back.

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

W

NOTES

VAX HARDWARE HANDBOOK
1982-83

READER'S COMMENTS

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our handbooks.

What is your general reaction to this handbook? (format, accuracy, completeness, organization, etc.) _____

What features are most useful? _____

Does the publication satisfy your needs? _____

What errors have you found? _____

Additional comments _____

Name _____

Title _____

Company _____ Dept. _____

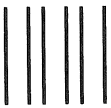
Address _____

City _____ State _____ Zip _____

(staple here)

(staple here)

(please fold here)



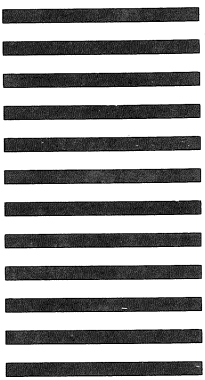
No Postage Necessary if Mailed in the United States

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 33 MAYNARD, MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
NEW PRODUCTS MARKETING
PK3-1/M92
MAYNARD, MASS. 01754



HANDBOOK SERIES

Microcomputers and Memories
Microcomputer Interfaces
PDP-11 Processor
PDP-11 Software
Peripherals
Terminals and Communications
VAX Architecture
VAX Software
VAX Hardware

digital

DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, MA 01754, Tel. (617) 897-5111 — SALES AND SERVICE OFFICES; UNITED STATES — ALABAMA, Birmingham, Huntsville ARIZONA, Phoenix, Tucson ARKANSAS, Little Rock CALIFORNIA, Bakersfield, Costa Mesa, El Segundo, Fresno, Los Angeles, Oakland, Sacramento, San Diego, San Francisco, Monrovia, Pasadena, Santa Barbara, Santa Clara, Santa Monica, Sherman Oaks, Sunnyvale COLORADO, Colorado Springs, Denver CONNECTICUT, Fairfield, Meriden DELAWARE, Newark, Wilmington FLORIDA, Jacksonville, Melbourne, Miami, Orlando, Pensacola, Tampa GEORGIA, Atlanta HAWAII, Honolulu IDAHO, Boise ILLINOIS, Chicago, Peoria INDIANA, Indianapolis IOWA, Bettendorf KENTUCKY, Louisville LOUISIANA, Baton Rouge, New Orleans MAINE, Portland MARYLAND, Baltimore, Odenton MASSACHUSETTS, Boston, Burlington, Springfield, Waltham MICHIGAN, Detroit, Kalamazoo MINNESOTA, Minneapolis MISSOURI, Kansas City, St. Louis NEBRASKA, Omaha NEVADA, Las Vegas, Reno NEW HAMPSHIRE, Manchester NEW JERSEY, Cherry Hill, Parsippany, Princeton, Somerset NEW MEXICO, Albuquerque, Los Alamos NEW YORK, Albany, Buffalo, Long Island, New York City, Rochester, Syracuse, Westchester NORTH CAROLINA, Chapel Hill, Charlotte OHIO, Cincinnati, Cleveland, Columbus, Dayton OKLAHOMA, Tulsa OREGON, Eugene, Portland PENNSYLVANIA, Allentown, Harrisburg, Philadelphia, Pittsburgh RHODE ISLAND, Providence SOUTH CAROLINA, Columbia, Greenville TENNESSEE, Knoxville, Memphis, Nashville TEXAS, Austin, Dallas, El Paso, Houston, San Antonio UTAH, Salt Lake City VERMONT, Burlington VIRGINIA, Arlington, Lynchburg, Norfolk, Richmond WASHINGTON, Seattle, Spokane WASHINGTON D.C. WEST VIRGINIA, Charleston WISCONSIN, Madison, Milwaukee INTERNATIONAL — EUROPEAN AREA HEADQUARTERS: Geneva, Tel: [41] (22)-93-33-11 INTERNATIONAL AREA HEADQUARTERS: Acton, MA 01754, U.S.A., Tel: (617) 263-6000 ARGENTINA, Buenos Aires AUSTRALIA, Adelaide, Brisbane, Canberra, Darwin, Hobart, Melbourne, Newcastle, Perth, Sydney, Townsville AUSTRIA, Vienna BELGIUM, Brussels BRAZIL, Rio de Janeiro, Sao Paulo CANADA, Calgary, Edmonton, Hamilton, Halifax, Kingston, London, Montreal, Ottawa, Quebec City, Regina, Toronto, Vancouver, Victoria, Winnipeg CHILE, Santiago DENMARK, Copenhagen EGYPT, Cairo ENGLAND, Basingstoke, Birmingham, Bristol, Ealing, Epsom, Leeds, Leicester, London, Manchester, Newmarket, Reading, Welwyn FINLAND, Helsinki FRANCE, Bordeaux, Lille, Lyon, Marseille, Paris, Puteaux, Strasbourg HONG KONG INDIA, Bangalore, Bombay, Calcutta, Hyderabad, New Delhi IRELAND, Dublin ISRAEL, Tel Aviv ITALY, Milan, Rome, Turin JAPAN, Fukuoka, Nagoya, Osaka, Tokyo, Yokohama KOREA, Seoul KUWAIT, Safat MEXICO, Mexico City, Monterrey NETHERLANDS, Amsterdam, The Hague, Utrecht NEW ZEALAND, Auckland, Christchurch, Wellington NIGERIA, Lagos NORTHERN IRELAND, Belfast NORWAY, Oslo, PERU, Lima PUERTO RICO, San Juan SAUDI ARABIA, Jeddah SCOTLAND, Edinburgh REPUBLIC OF SINGAPORE, SPAIN, Barcelona, Madrid SWEDEN, Gothenburg, Stockholm SWITZERLAND, Geneva, Zurich TAIWAN, Taipei TRINIDAD, Port of Spain VENEZUELA, Caracas WEST GERMANY, Berlin, Cologne, Frankfurt, Hamburg, Hannover, Munich, Nuremberg, Stuttgart YUGOSLAVIA, Belgrade, Ljubljana, Zagreb

ORDER CODE: EB-21710-20