

SECTION 4

Device Handling

4.1 INTRODUCTION

The device handling features and conventions have been designed for simple but flexible operation, with few restrictions relating only to cooperation with system device operations, i.e. virtual memory on the disk. Each device is assigned a device type which is a small positive integer. Device type numbers are assigned at system build time. This section is aimed at describing the specific operation and the internal structures used by many of the devices in the system. First, the general scheme for handling I/O requests is presented, later sections detail specific devices.

4.2 DEVICE DECODING

Each device in the system that requires interrupt processing maintains a Device Descriptor Block (see section on interrupts). This descriptor block contains all the information that the system needs for processing requests and interrupts for this device. The Device Descriptor Block maintains information about the device state. A separate structure, the Request Block (RQB) is used to transfer request information. The first word in the Device Descriptor Block is used as a link word for the interrupt decoder and points to the next device descriptor on the same interrupt level. The second word is called the device information word. This word contains information needed to determine what type of processing is needed when an initiate I/O request is sent to the device and also what type of interrupt processing is required. The device type implicitly specifies the number of device specific words required. The details of some specific devices follow. The initiation and interrupt processors use the device specific portion of the block to maintain information pertaining to the device and the outstanding requests. Therefore, every device in the system that requires microcode handling must provide an entry point in the initiate dispatch table and an entry point in the interrupt handler dispatch table. The entries are placed in the tables according to device type.

To initiate an I/O request the %IO miscop (miscellaneous operation) is used. The parameters are the device descriptor and

a request descriptor. There are two basic forms for the request descriptor. The first form is a fixnum. In this case, the single word specifies the operation being requested. The second form is a request block. The request block is an array. The specifics of the request block are defined by the device handler. To initiate a request the device type is fetched from the Device Descriptor Block and the device initiation handler is called for this device type. The device initiation handler is responsible for servicing this request by either starting the device operation or placing the request in a queue for later processing.

To make the best use of the peripheral resources on the machine there is a mechanism to queue requests for I/O and continue processing. In any case, we would not want the processor to just sit idle during I/O requests, but rather run other processes that might be ready to run. To be able to do this some devices maintain a list of outstanding requests, and when one request has completed automatically start the next request. This requires that the interrupt handlers for a device must maintain the queue as part of its duties.

4.3 DEVICE-HANDLER DATA STRUCTURE ALLOCATION

Memory references made by interrupt handlers must be to static, wired virtual memory or to the physical address space. Device-specific control registers are generally accessed as NuBus physical memory which has no mapping in the virtual address space. Other interrupt-handling data structures, however, generally are mapped into the Lisp virtual address space so that both the microcode and Lisp can access them as normal virtual memory. Allocation of these data structures follows the following conventions:

- Device descriptor data structures as well as the interrupt servicing queue itself are allocated in the SYS:DEVICE-DESCRIPTOR-AREA. This is a special system-wired no-scavenge area in low virtual memory. The structures are allocated simply as segments of un-typed memory, hence are not viewed as Lisp objects; therefore, this area is not garbage collected or scavenged. Addresses within the data structures can only point to other data structures in the SYS:DEVICE-DESCRIPTOR-AREA or to objects in other static areas.
- Actual I/O buffers themselves are generally allocated as Lisp arrays in static areas, and are wired down before the I/O starts using paging primitives such as SYS:WIRE-ARRAY or SYS:WIRE-PAGE. Examples of current static areas are:

SYS: DISK-BUFFER-AREA	disk and tape RGBs and data buffers
SYS: SERIAL	serial and parallel port I/O Buffers
CHAOS: CHAOS-BUFFER-AREA	Ethernet packet buffers

The remainder of this section describes specific devices and their I/O Interrupt handlers.

4.4 DEVICE DETAILS

4.4.1 The NuBus Peripheral Interface Board.

The NUPI is used for interfacing to disk and tape devices. The NUPI receives operation requests by writing the address of a NUPI Command Block to a special address in the NUPI address space designated as the Command Register. The NUPI then processes this command asynchronously from the Explorer processor. When the request has been completed, the NUPI stores the status of the operation back into the status word of the request block and, if specified in the request block, posts an event to the Explorer processor to signal completion of a command. (The Explorer convention is that all requests to the NUPI post the completion event.) This event posting is fielded by the processor as an interrupt. The device interrupt handler is called to process the completion of the current request and initiate the next one if required.

Figure 4-1 shows the format of the Device Descriptor Block for the NUPI. The Link Word and Device Information Word are standard for all devices. The Control Space Word holds the NuBus address of the control space of the NUPI board. This is needed to correspond with the board. A NUPI may have several disk or tapes units under its control. Each disk or tape unit is connected to a formatter, which is a local controller. A formatter may have one or two devices connected to it. Each device, formatter and the NUPI itself is considered a unit, and all may simultaneously have a request in progress. The device handler maintains a request queue for each device. The request at the front of the queue is the one currently being processed. When a request comes to the front of the queue the Busy Bit in the Information Word (see Figure 4-2) is set to signify that the request is in progress. When the request is completed the Busy Bit is reset and the Done Bit is set. The request block is removed from the queue. If there are any other requests in the queue at this time, processing is started for them.

The Unit Busy Bitmap indicates which units have requests in process. This is used because when a request completes and an interrupt is signalled all devices with requests in process must be checked to find which ones have completed. The bit map allows polling of only those units for which processing is possible.

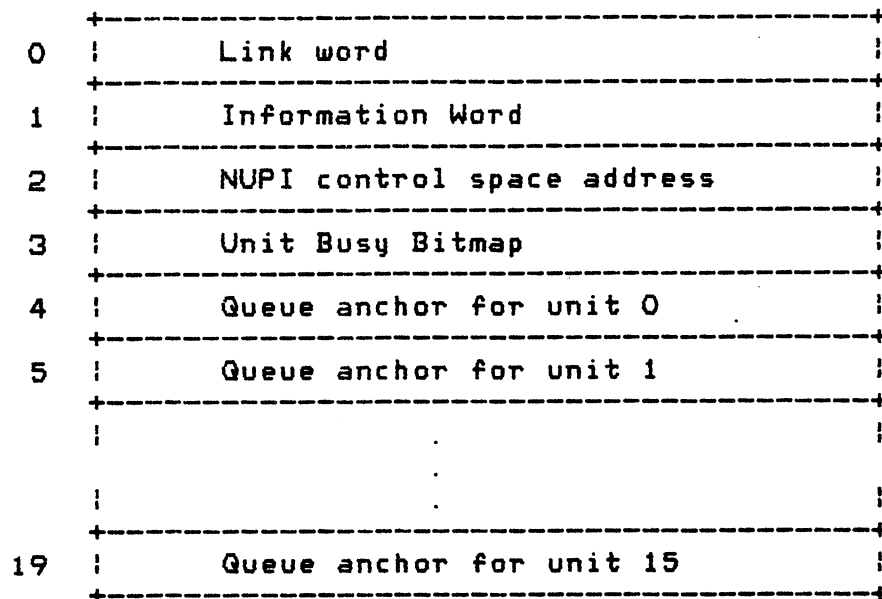


Figure 4-1 NUPI Device Descriptor Block

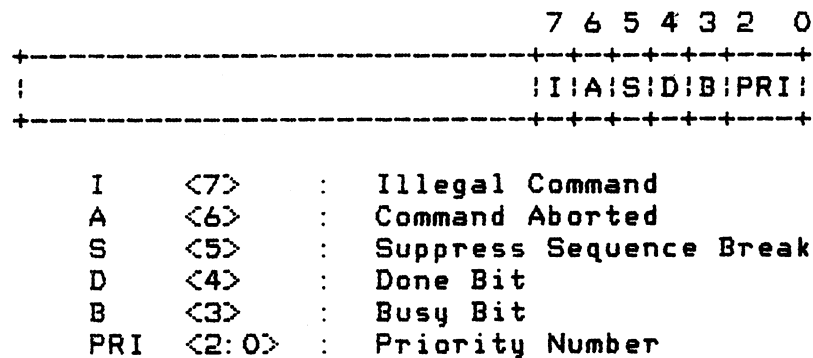


Figure 4-2 NUPI Device Descriptor Information Word

4.4.2 The Keyboard.

When a keyboard interrupt occurs, the keyboard character is copied from the hardware register into the keyboard buffer. The Lisp keyboard process notices that a character is available and

handles it.

4.4.3 CRT Vertical Retrace (Interrupt).

An interrupt occurs when the CRT begins the vertical retrace interval. At this time, the mouse is checked to see if it has moved or if any of the buttons have changed state. Mouse button changes go into the mouse buttons buffer. If the mouse has moved, it is undrawn in its old position and redrawn in its new position. The mouse process then notices either mouse motion or mouse buttons.

This interrupt also handles the timer on the Explorer I (the Explorer II has a microsecond timer on the processor). The short interval timer is read to determine the exact number of microseconds that have elapsed since the last TV vertical retrace interrupt. This value is then added to an internal microsecond timer and the short interval timer is reset.

4.5 SERIAL AND PARALLEL IO PORTS

The Explorer has one serial port for bidirectional asynchronous communication, and one Centronix (tm) standard unidirectional parallel port for high speed data transmission. (Centronix is a registered trademark of Centronics Data Computer Corporation.) Each has microcode support and can be used with a standard stream interface.

4.5.1 RS232 Serial Port.

The Explorer serial port is based on a Zilog Z8530 Serial Communications Controller found on the System Interface Board (SIB). It can be configured to handle all asynchronous formats regardless of data size, number of stop bits, or parity requirements. It can also accomodate all synchronous formats including character, byte, and bit-oriented protocols.

There is both microcode and Lisp support for the serial port. The microcode handles receive and transmit interrupts for the chip. A stream interface to the serial port is provided to make it convenient to use. Although the chip supports synchronous modes, the software currently provides support for asynchronous modes only.

Microcode handling of serial port interrupts enables high speed transmission of characters. Characters to be sent out through the port are queued in a buffer. When the serial port signals a transmit ready interrupt, the microcode removes the next character from the buffer, if any, and sends it out. Similarly,

when the chip signals a receive character ready, the microcode will remove the character from the chip's internal register and store it into another buffer where it can be accessed from Lisp when convenient.

The microcode also has support for an XON/XOFF protocol. When enabled, this will allow external devices to control the transmission of data by the Explorer and will allow the Explorer to stop a device from sending data. When running in this mode, the receipt of an XOFF character will disable further transmission until an XON character is received. If the Explorer's serial input buffer comes within 10 characters of being full, it will automatically transmit an XOFF character to temporarily stop data transmission. Naturally, the external device must also support XON/XOFF protocol for this scheme to work correctly.

The serial port is programmed by writing data into the 14 write registers of the chip. Received characters and chip status are available in seven read registers. For details on the function of these registers see the Explorer System Interface manual. These registers are found within the NuBus address space of the SIB. It is therefore possible to program and use the chip directly from Lisp in polled mode, although this is not recommended.

The port is intended to be used in interrupt driven mode provided by the microcode and serial streams. Serial streams provide a convenient way to initialize the operating parameters of the chip and to send and receive characters. Operating parameters are normally specified in the call to SYS:MAKE-SERIAL-STREAM. Characters are transmitted with :TYO and received with :TYI. For more information on serial streams, see the Explorer Input/Output Reference manual.

4.5.2 Parallel Port.

Like the serial port, the parallel port is also found on the System Interface Board. The programming interface consists of two registers found within the NuBus address space of the SIB. The data register is an 8-bit write-only register. The command register is also 8 bits wide. Writing to the command register allows you to initialize the port, cause data strobe, and enable auto linefeeds or interrupts. Reading from the command register returns port status including busy, paper out, online, and fault.

There is microcode support for running the parallel port in interrupt mode. Data to be transmitted is placed into an output buffer. When the external device is ready to receive a character, an event is posted by the SIB to the processor. The microcode will remove the next character from the output buffer and send it out. If a large number of characters are queued in

the output buffer, data can be transmitted at very high speeds. A stream interface is provided to make the parallel port easy to use.