T H E   U N I V E R S I T Y   O F   M I C H I G A N

Memorandum 11

I/O EXTENSIONS TO RAMP

David Mills

CONCOMP: Research in Conversational Use of Computers
ORA Project 07449
F.H. Westervelt, Director

supported by:

DEPARTMENT OF DEFENSE
ADVANCED RESEARCH PROJECTS AGENCY
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION   ANN ARBOR

October 1967

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

RAMP is a multiprogramming system written for the PDP-8 and designed primarily for real-time interactive systems using a variety of I/O devices. The structure of the basic system nucleus is described in Reference 1. This memorandum describes the system interface to I/O device support routines and the conventions under which they operate. Several different kinds of I/O devices have been attached to PDP-8/RAMP systems, and support routines for these devices have been coded. Some of these are described herein.

Since the publication of Reference 1, several functional improvements have been incorporated into the currently operating PDP-8/RAMP systems. In particular, the copy nucleus has been completely rewritten and is described below. The command language interpreter has also been restructured so that hangups can occur only in those pathological cases where device support routines malfunction. In addition, the I/O utility routine package has been expanded to include routines for reading and printing decimal integers. These latter functional improvements do not directly affect the utility of the RAMP system in its multiprogramming environment and are not discussed extensively in this memorandum.

II. BASIC SYSTEM ARCHITECTURE

The basic architecture of the RAMP system is discussed in depth in Reference 1 and is only briefly summarized here. The basic system nucleus consists of a task-switching monitor, an interrupt identifier, a set of buffer management routines, a rudimentary command language interpreter, and a set of I/O formatting routines. The copy nucleus, added to the basic nucleus for store-and-forward message processing, includes the copy and echo routines together with additional elements of the command language interpreter. To the resultant composite system is added a set of device support routines which interface both to the command language interpreter and to the copy nucleus.

For the purpose of subsequent discussion, the most salient features of the composite RAMP system involve the notion of real-time/task-time operations and the structure of the task itself. Of particular interest in device support routine construction are the methods by which control and text information is passed among the system components and the conventions by which devices are allocated to the various system operations.

2.1 The Task

A task is a subroutine which is designed to be involved asynchronously with other such tasks in the system. It may call upon other tasks in much the same manner that ordinary subroutines may call upon each other. The principal operational difference in the RAMP system is that every such task call is saved in a first-in-first-out queue, so that at any time more than one call request for a given task may be outstanding in the queue. The task-switching monitor retrieves these call requests one at a time from the queue and invokes the task indicated where possible. When a task has completed execution it transfers control directly back to the task which called it.

The task queue entries may be created by two methods. One of these, effected by the system subroutine TASK, places a called-task identifying entry in the task queue and places the calling task in an inactive state. Following execution of the called task, the calling task again becomes active. The other method, effected by the system subroutine INSERT, places the identifying entry in the task queue and then continues execution of the calling task. The first method allows concurrent processing of those tasks which must be delayed while other tasks run to completion, while the second allows parallel processing of those tasks which need not be so constrained.

A principal attribute of a task is its re-entrability status. When a serially reusable task is invoked by the task-switching monitor, its entry point is identified as being busy and subsequent call requests for its use will be automatically requeued. Presumably the invoked task calls upon another task and is placed in the latent state pending completion of its called task. The re-entrability attribute furnishes an interlock which prevents outstanding call requests for the task from being honored during this interval. The interlock is removed explicitly by the task itself before returning to its own calling task.

2.2 Real-Time and Task-Time Operations

The vast majority of processing time spent in task execution is under conditions of the interrupt system being enabled. Thus, while a particular task is in control, I/O device interrupts normally can occur; and the routines servicing these interrupts can insert call requests in the task queue for deferred processing. This operation naturally encourages reference of the interrupt routine processing as being done in real time, and that of the task itself as being done in task time. This terminology has been extended to refer to all processing done with the interrupt system disabled as real time and to refer to all other processing as task time. Real-time routines can share a good deal of common code and temporary storage. However, task-time routines may operate in a parallel-processing mode and therefore must maintain private pools of temporary storage. In any case, of course, real-time and task-time routines cannot share storage unless the task-time routine disables interrupts when the common storage is changed. In particular, to conserve storage, the resident buffer management routines and certain task-switching monitor routines share common storage with many interrupt routines. Thus the operations of fetching and storing characters in a buffer, as well as the task-switching operation itself, take place in real time.

The environment of multiprogrammed operations in task time thus requires a careful analysis of common storage requirements if some degree of storage optimization is to be achieved. The general rule is that if a particular task calls another task via TASK or requeues itself for any reason, then all temporary storage required after the called task returns must be dedicated to the calling task and may not be shared with any other task. Furthermore, if a particular task must always be re-entrable, as is usually the case in device support routines, then any temporary storage it requires must either reside in the task queue itself as part of the call request or must reside in a control-block region pointed to by such a call request.

III. I/O DEVICE SUPPORT ARCHITECTURE

All data transfers between peripheral devices and the system nucleus are standardized in format and allocation procedures, with as much emphasis on common system routines as possible. All transmission occurs on a byte-by-byte (8 bit) basis, using integral cyclic buffers and buffer management routines. (However, these conventions do not disallow transmission on a block-transfer basis or of byte-sizes other than 8 bits.) Full-transparent 8-bit operation is possible using the common routines, with the high-order four bits of each byte available within the system as modifier or format information (see Figure 1).

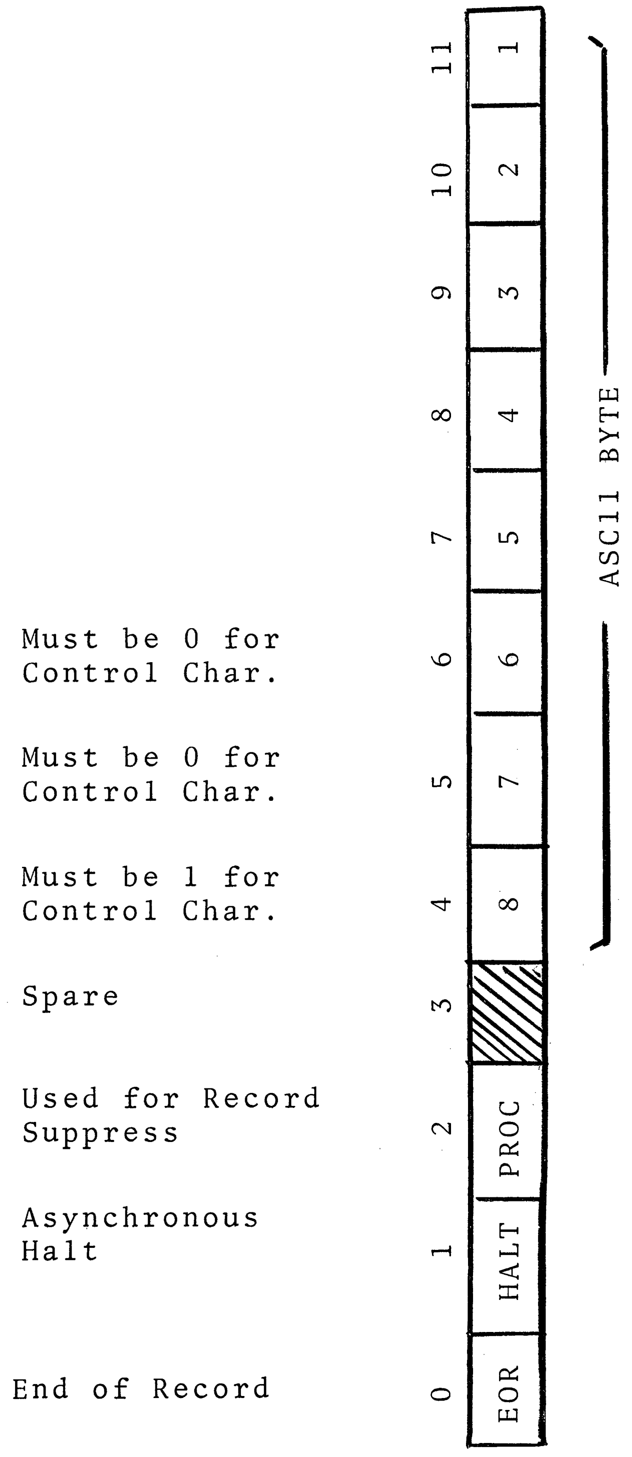

Figure 1. DATA BYTE FORMAT

By convention, all transmissions involving either the command language interpreter or the copy nucleus operate on a unit record basis, with the record delimited by a modifier bit. Since it is possible that command and copy operations may be in progress for a number of devices simultaneously, a means for aborting a transmission and clearing the transmission path is provided in case of device malfunction. The following sections describe these considerations in greater detail.

3.1 Record Formatting

A stream of data bytes (see Figure 1) in transmission between a pair of devices is punctuated occasionally by an end-of-record (EOR) indication. This gives the sink device an opportunity to acknowledge the record for error checking and buffer administration procedures. The decision as to the exact point of punctuation is due to the source device support routine (DSR) and usually occurs at a logical breakpoint, such as at a teletype carriage-return. Common system routines recognize the sign bit of a data byte as the EOR indication. The low-order bits of the byte may contain additional information, such as whether the EOR indication punctuates a logical or physical end-of-block, whether end-of-file is assumed, and so forth.

The sink DSR recognizes this EOR as a request for record acknowledgment. This normally takes the form of a special character transmitted back to the source DSR. In most cases, the acknowledgment can be automatically provided by the system itself in connection with an entry point provided for this purpose. In other cases, the device itself processes the EOR and provides an asynchronous interrupt for acknowledgment purposes. An example of the latter behavior occurs in connection with machine-to-machine communication, where the receiving machine normally indicates ready-to-receive status by a character transmitted over the interconnecting data link. In such cases as these, the sink DSR must provide real-time recognition of the acknowledgment character and insert the appropriate task-time routine. Common routines in the system make this process convenient.

The source DSR interprets the acknowledgment character depending upon device type. In most cases, the only function performed is in connection with device allocation status; but, in some cases, the acknowledgment character may be transmitted to the source device itself. Again, these situations commonly arise in machine-to-machine communication and provide for an indefinite number of store-and-forward repeaters on any one transmission circuit.

These record formatting conventions allow a treatment of message transmission independently of the particular device-type involved, and thus allow a fair number of common service routines. Recognition of the particular EOR and acknowledgment criteria is at the discretion of the particular DSR and may include device characteristics other than the transmitted data stream.

3.2 <u>Device Allocation</u>

As demonstrated above, a good deal of programming effort has gone into defining and implementing a viable interface between the highly tailored device support routines and the common routines in the basic system nucleus and the copy nucleus. The following discussion describes how the common system routines avoid device conflicts while allowing rather general record interleaving among several devices.

Each device attached to the system is regarded as both a source and sink for purposes of record transmission and is identified by a single logical device number (LDN) in the range from 0 to 63 (decimal). (This convention does not preclude attachment of a read-only or a write-only device.)

This LDN also identifies the device for record acknowledgment purposes. Corresponding to each LDN attached to a particular RAMP system configuration is a four-word entry in a device control block (DCB) table (see Figure 2). The four words of this entry are associated symmetrically in pairs with the device as a source and as a sink. One word of each pair represents the entry point of a segment of code which services the device as a source and sink respectively. The other word of each pair contains switches as to the allocation status of the device.

In case of a source device, the allocation status word contains a bit indicating whether the device is busy or not, a bit indicating whether an asynchronous halt is pending or not, and finally a field containing the LDN to which any output activity generated by this device should be directed. Unless asynchronous acknowledgment is required by the source device, this word need be accessed only by the common system routines, and by the source DSR only as described in the next section.

In case of a sink device, the allocation status word contains a bit indicating whether the device is busy or not, a bit indicating whether acknowledgment is pending or not, and finally a field containing the LDN to which the acknowledgment should be directed. Unless an asynchronous acknowledgement is furnished by the sink device, this word need be accessed only by the common system routines.

3.3 Message Processing

Associated with each attached source LDN in a RAMP system configuration are three bit attributes used by the copy nucleus (see Figure 2). The first bit indicates whether the source is input to the command language interpreter or to the copy nucleus, while the last two specify the conditions under which a copy/command task will be inserted. If the source

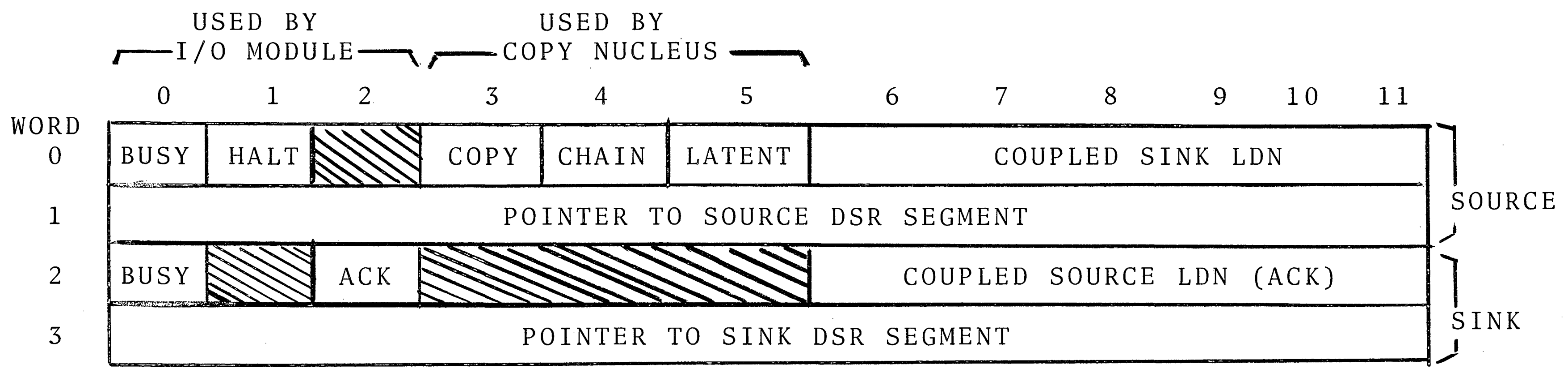

Figure 2. DCB TABLE ENTRY

message is input to the command language interpreter (CLI), then its interpretation (see Reference 1) may generate none, one, or more lines of output. In this case, record acknowledgment is provided by the CLI itself only when the last character of the source message has been processed. If the source message is input to the copy nucleus, then the text is transmitted unaltered to the sink LDN and record acknowledgment is provided by the sink DSR itself.

The copy-mode attribute can be set by the CLI in response to the appropriate command if the device is not busy and not already in copy mode. The coupled sink LDN can be changed if the device is not busy and not in copy mode. However, once a source device is placed in copy mode, these parameters cannot be changed unless the device is returned to command mode via a special escape character transmitted by the source device itself.

The majority of devices which operate on a character-by-character basis at moderate data rates provide an interrupt when character assembly is complete. The interrupt routine for a source device in this category typically edits the text stream while placing the data bytes in an input cyclic buffer using the common system routines. When the EOR criteria is realized, the interrupt routine

a. inserts an EOR identifier in the input buffer, and

b. inserts a dispatcher task in the task queue.

When the dispatcher task is inserted, the source DSR is responsible for specifying as a parameter its own LDN. Further operations on the record text can be performed in task time. At any instant, more than one such record may reside in the input buffer, and a dispatcher task will be outstanding for each in the task queue.

The dispatcher task itself (see Figure 3) is a re-entrable task which performs the following operations:

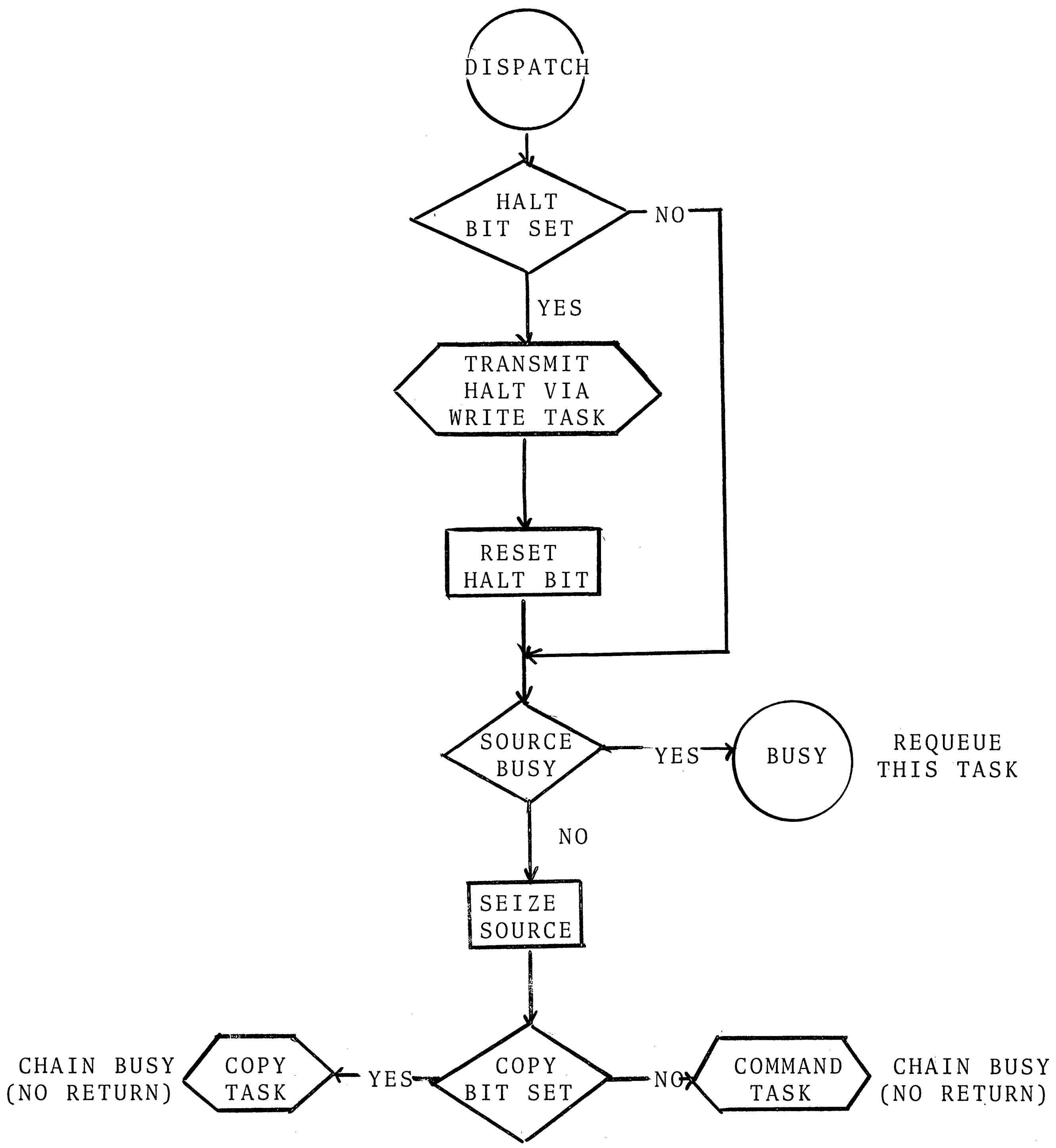

Figure 3. DISPATCHER TASK

a. reserves the source device,

b. performs certain housekeeping chores, such as retrieving the sink LDN from the DCB table and supervising the asynchronous halt condition (see below),

c. inspects the copy/command attribute and calls the appropriate task.

The dispatcher task will be requeued in any case that the source LDN is busy due to some prior uncompleted operation. This organization allows a source device to be reconfigured by its own input messages without confusion when several records have already been stored in the input buffer.

The command language interpreter itself (see Figure 4) is a serially reusable task and remains busy until the entire command message record is processed. This processing may be considerably involved, so that multiple call requests may be requeued for several task queue cycles. However, all requests involving production of output messages are inserted by the CLI and processed in parallel with other tasks in the system. Thus, system delay due to output buffer overflow for some particular device is avoided at the expense of task queue entries.

The copy task itself (see Figure 5) is a re-entrable task which reads a character from a source LDN and writes it unaltered on a sink LDN. The copy task is requeued upon entry if the sink device is busy. After the EOR is transmitted to the sink device, the copy task terminates. However, in some special cases where the source device is not able to provide interrupts, the dispatcher task is inserted prior to any data transmission activity and remains in the queue for the duration of activity. In these cases, the dispatcher task calls either the command language interpreter or the copy task, as appropriate. Once the input message is processed, these tasks exit directly to the dispatcher task to process the next input message. Two bits set in the source DCB table entry control this function (see Figure 2).

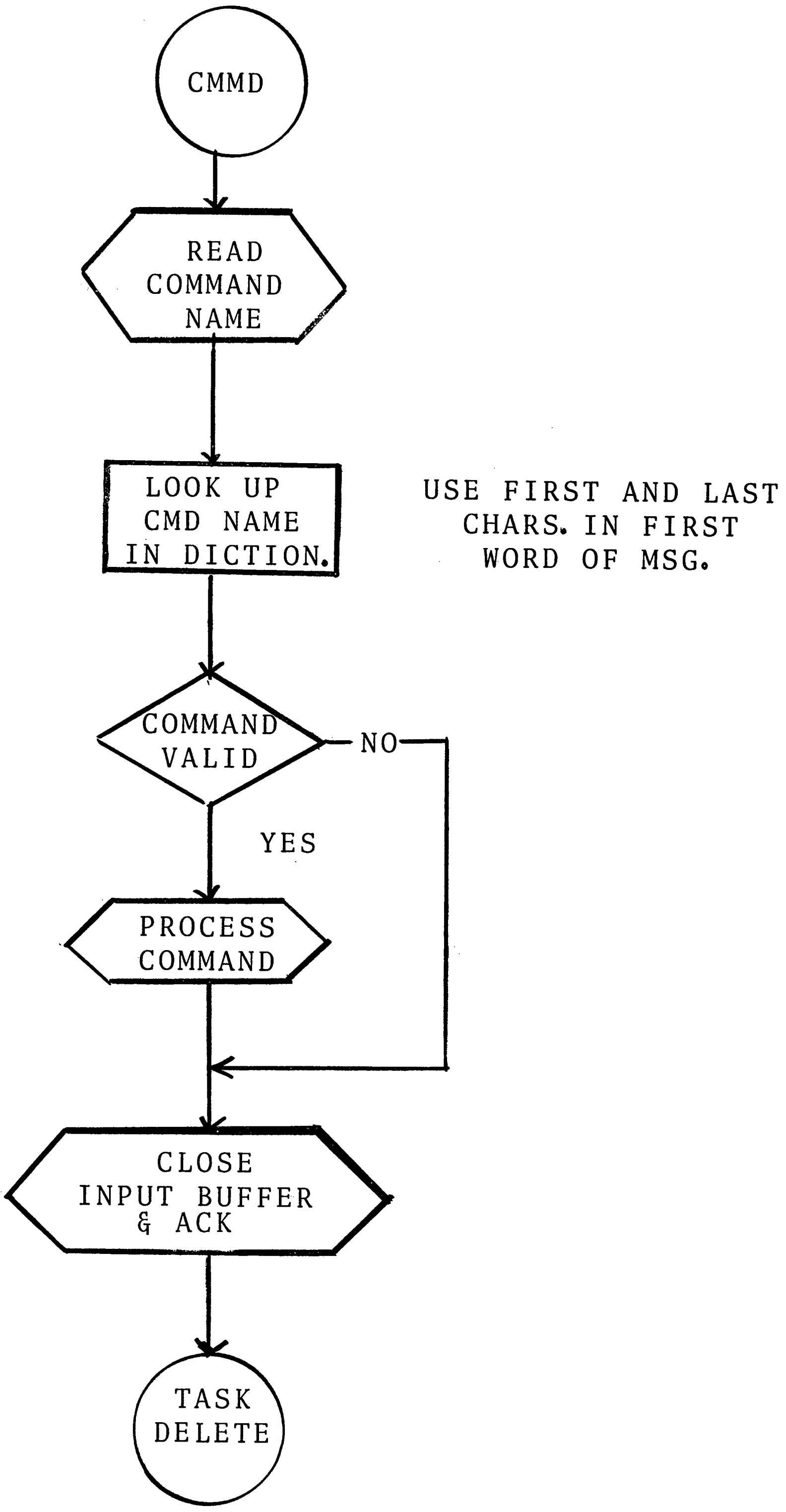

Figure 4. COMMAND LANGUAGE INTERPRETER

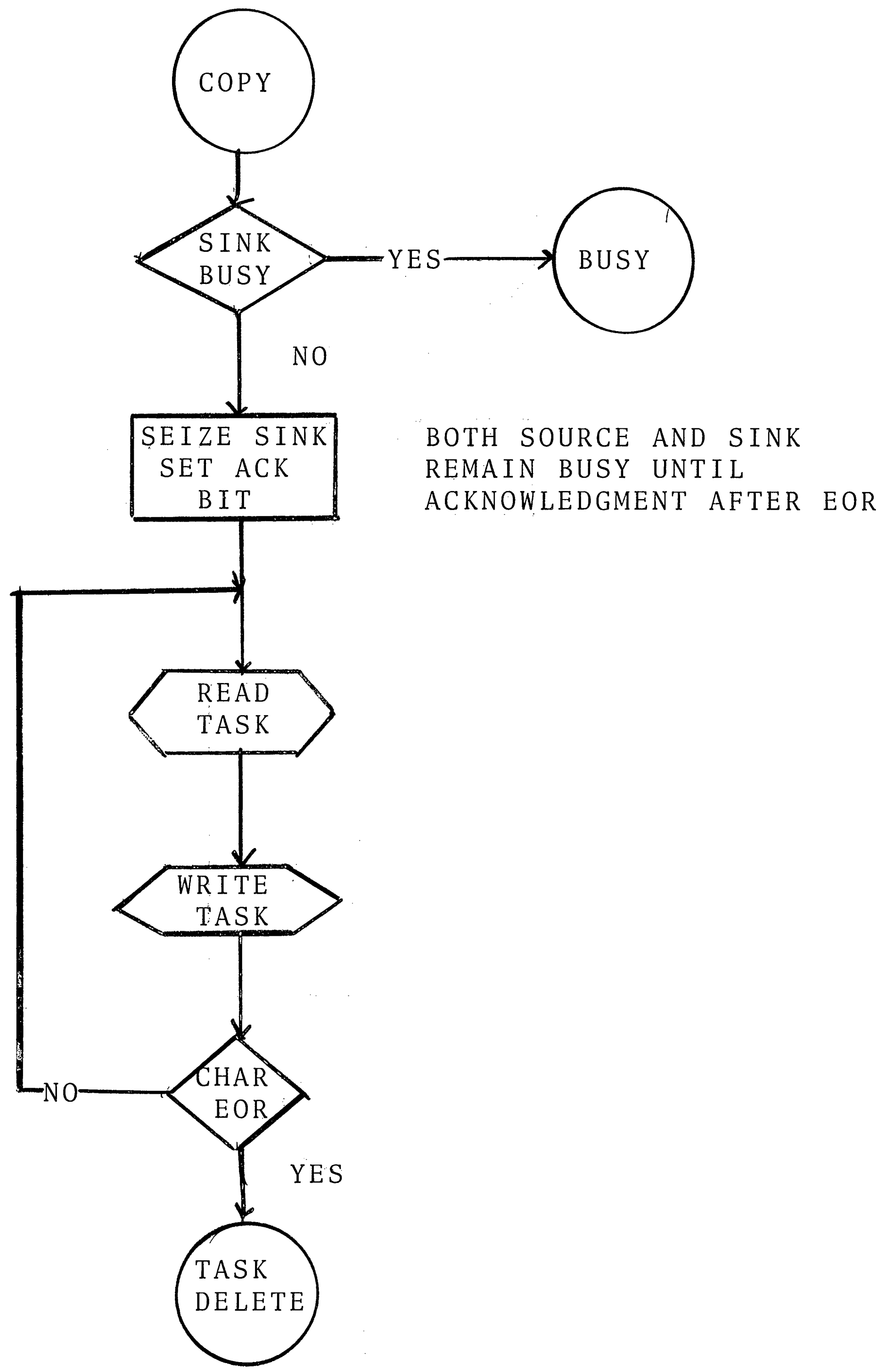

Figure 5. COPY TASK

Note that in the above architecture, each source device is associated with only one sink device, but each sink device may be associated with any number of source devices. Note further that the source-sink device path remains busy until all acknowledgment procedures connected with any particular record have been completed. If several source devices produce simultaneously output for a single sink device, the system will interleave these records to the sink device on a record-by-record basis, acknowledging each source device in turn when its record is transmitted.

3.4 Device Support Routine Interaction

Since record transmission has been specified on a byte-by-byte basis, it is natural to assume that the organization of the device support hierarchy be built about a task which transmits a single byte to or from the device. In order to provide a convenient interface between the common system routines and the device support routines, two tasks have been included within the system: READ, which reads a single byte from a device, and WRITE, which writes a single character on a device (see Figure 6). Each of these tasks sets up DCB pointers and performs other housekeeping functions, then calls a segment of code which services the peculiar demands of the device.

In the case of an input device, the segment of code terminates through the IORTN exit (see Figure 7) which checks for EOR and exits appropriately in turn to the calling task. Thus the only obligatory character-sensitive operation recognized in common by the READ task and the device support segment is the EOR indication. When the acknowledgment is transmitted to the source device, a simple convention is adopted which enables the segment of code called by the READ task to determine whether the task call was made as the result of an input character request or as the result of a record acknowledgment request. In the former case, the source DSR segment

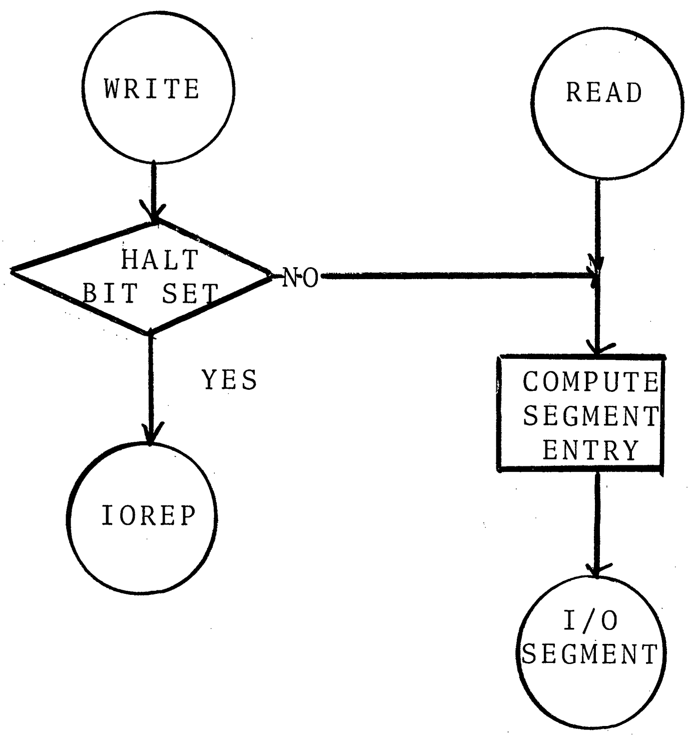

Figure 6. READ/WRITE TASKS

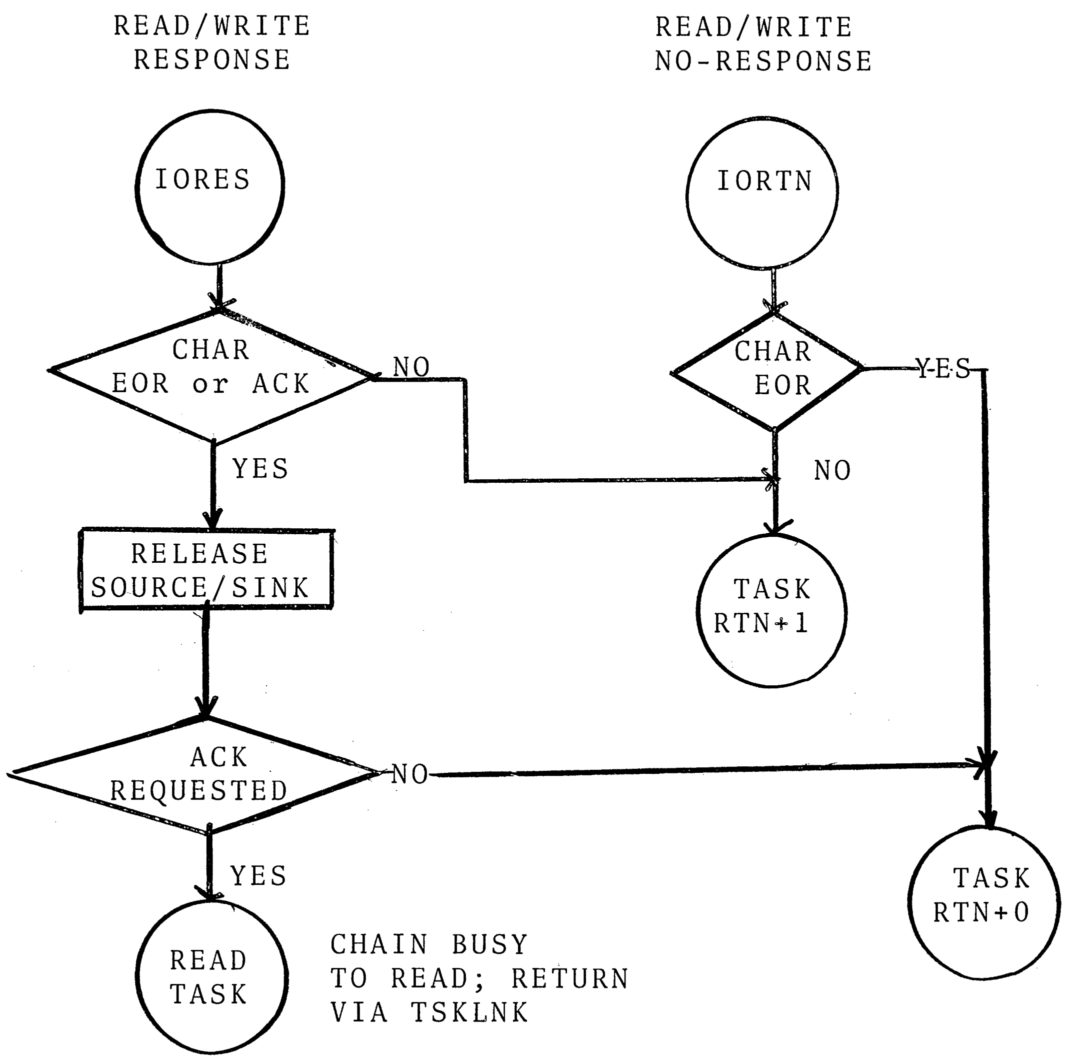

Figure 7. INPUT/OUTPUT TASKS SEGMENT RETURNS

returns to the IORTN exit as described above, while in the latter case it returns to the IOREP exit, which releases the source device for further system use. Also, in the latter case, the source DSR has the option of transmitting a character to the source device itself, an operation which may, for example, activate a data link to a remote machine. A typical READ task segment is described in the next section.

In the case of an output device, the segment of code may terminate through either of two exits. The IOREP exit checks for the EOR, transmits the acknowledgment automatically to the associated source DSR, and then returns to the calling task. The IORTN exit returns directly to the calling task. In the latter case, the system assumes that at some future time an asynchronous interrupt will occur for the sink DSR, which will then insert a utility task, which in turn will transfer control to the first exit above, thus providing the acknowledgment to the associated source DSR. A typical WRITE task segment is described in the next section.

Both input and output buffers can be dynamically allocated in units of a page (128 words) by common routines in the system. The allocation technique makes use of a table with as many entries as allocatable pages in the machine. A buffer is seized by scanning this table for that entry corresponding to the first free page, which is then reserved. Following reservation, a buffer control block is built in the page itself and its location passed to the READ/WRITE task segment requesting buffer storage. If all allocatable pages in the system are in use, the system is said to be in page-wait status, and requests for additional pages are requeued. When a buffer is emptied, its storage is returned to the allocatable page pool for use by other I/O devices.

Since storage is acquired and released by task-time routines, device support routines which are constrained in overrun requirements commonly either preallocate buffer storage before record transmission begins or make use of dedicated buffers not belonging to the allocatable page pool. Most input devices such as keyboards are in this category, while most output devices such as printers are not. For this reason, a convenient output routine is included in the system which can be common to many output devices. This routine allocates buffer storage as required and transmits characters to the buffer. Each output real-time DSR routine can release the storage so acquired by inserting a utility task when the buffer becomes empty. These common routines recognize the asynchronous halt indication (see below) as a request to discard an entire buffer and to return its storage to the allocatable page pool. Thus page-wait conditions, which occur when multiple device support routines contend for restricted buffer space, can be cleared.

3.5 Store-and-Forward Operations

The operations of message store-and-forward procedures can perhaps best be illustrated by reference to a particular I/O device. The device chosen here as an example is a serial-synchronous line adapter designed to interface two PDP-8/RAMP systems to each other via 2000-baud data links operating on the standard switched-telephone network. The peculiar feature of this device, which recommends it for study here, is that record acknowledgment is expected from the remote machine via the data link itself. The following discussion describes the appropriate real- and task-time device support routine segments and how they interface with the system so far described. Finally an example is drawn from typical system operating procedures showing how the various parts of the system interact.

A typical READ I/O segment is shown in Figure 8. This segment is activated in task time by the READ task (see previous section) and exits through the IORTN or IDREP return as appropriate. Note the convention by which a record

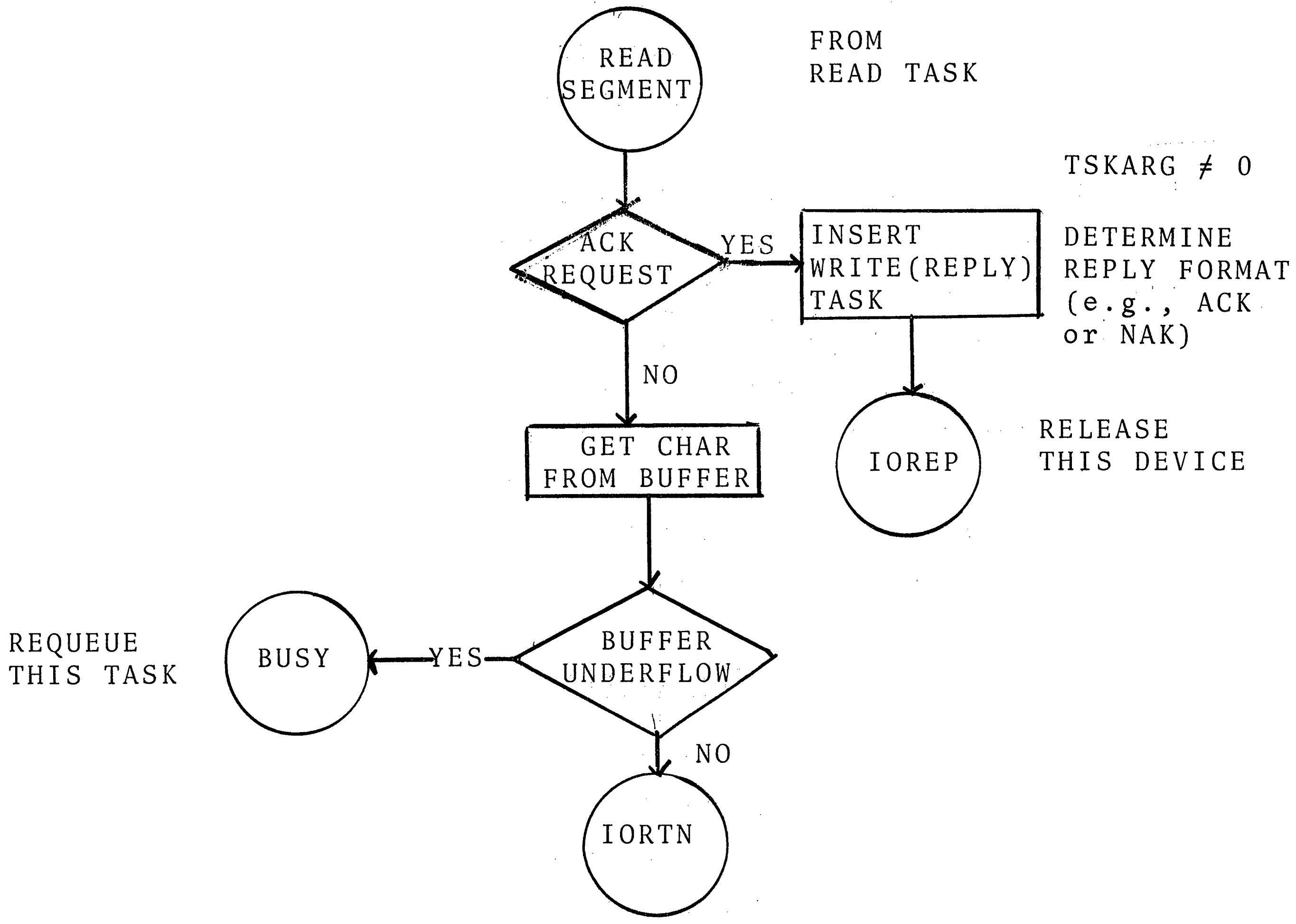

Figure 8. TYPICAL READ SEGMENT

acknowledgment is indicated—a nonzero TSKARG in a call to the READ task. In this case such an acknowledgment, presumably produced by the system or some other I/O device in response to a previous message, is transmitted over the data link to the sender of the previous message. The intent here is that this acknowledgment be transmitted asynchronously with possible outgoing traffic and be detected at the other end of the link by virtue of its special character code.

A typical input device interrupt routine is shown in Figure 9. This routine is called by the interrupt identifier when a service request for this device has been recognized. The routine has the responsibility of clearing the interrupt, reading the input character, and arming the device so that the next character can be received.

There are three special characters recognized by this routine—the EOR, ATN, and ACK characters. Others may of course be identified, such as those for line-editing and error-recovery procedures, but these do not contribute greatly to the present discussion. All of the non-special characters are simply parked in the input buffer as they arrive. The three special characters cause task inserts, as shown for the dispatcher task (see previous discussion).

A typical WRITE I/O segment is shown in Figure 10. This segment is activated in task time by the WRITE task and exits to IORTN return. In this example, if a buffer is not attached to the device when the segment is activated,then one is seized and attached. If a buffer is already attached and if the WRITE character has the halt modifier set (see next section), then the buffer is discarded and the task is requeued. This behavior ensures recovery from page-wait conditions if the device, in this case a data link, jams or becomes inoperative and all available buffer storage has been allocated. In this case the halt indication transmitted to the jammed device will release its buffer storage so that some other device can begin operation while the jammed device

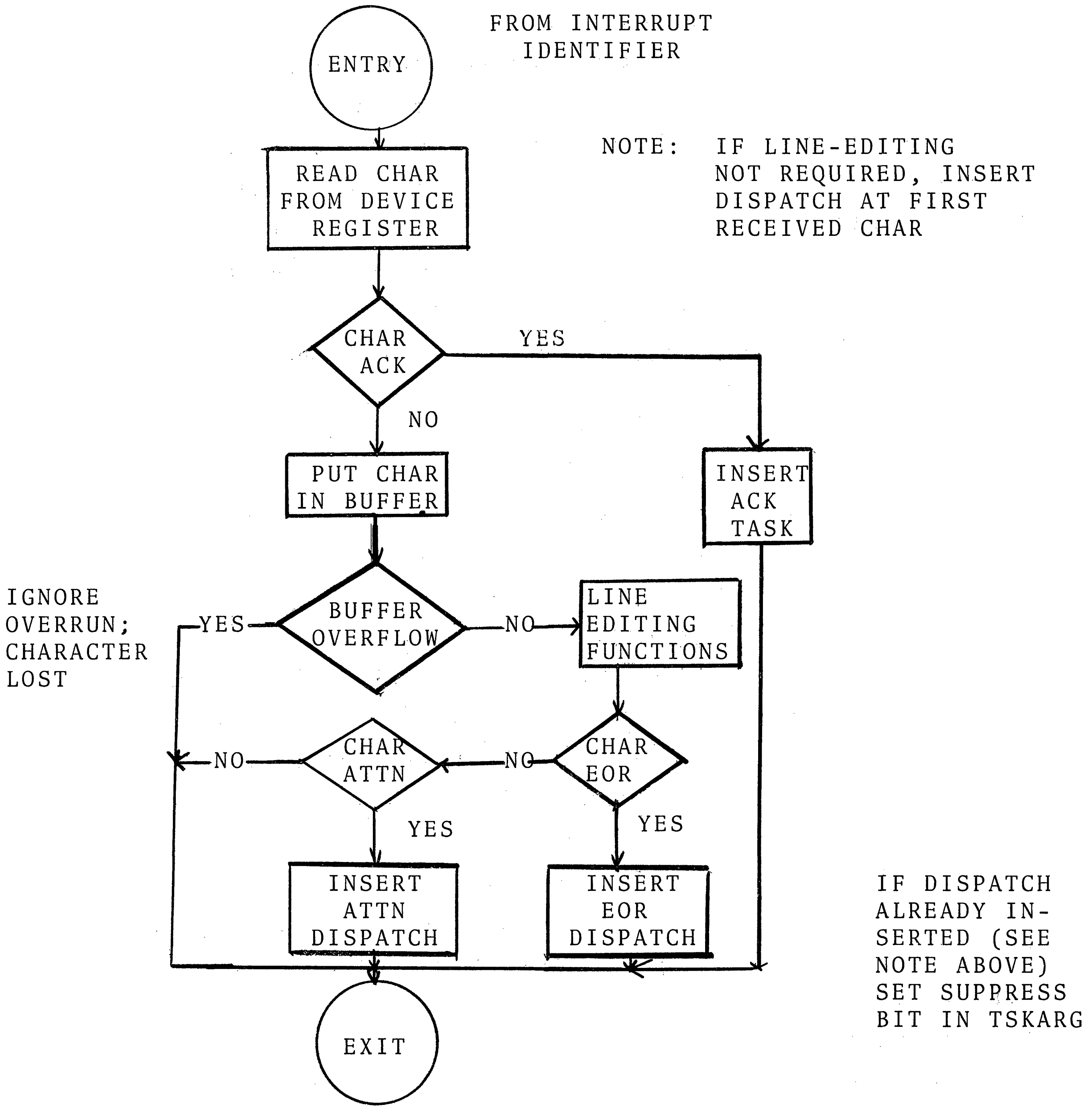
FROM INTERRUPT IDENTIFIER
ENTRY
NOTE: IF LINE-EDITING NOT REQUIRED, INSERT DISPATCH AT FIRST RECEIVED CHAR
READ CHAR FROM DEVICE REGISTER
CHAR ACK
YES
NO
PUT CHAR IN BUFFER
INSERT ACK TASK
IGNORE OVERRUN; CHARACTER LOST
YES
BUFFER OVERFLOW
NO
LINE EDITING FUNCTIONS
NO
CHAR ATTN
NO
CHAR EOR
YES
YES
INSERT ATTN DISPATCH
INSERT EOR DISPATCH
IF DISPATCH ALREADY IN-SERTED (SEE NOTE ABOVE) SET SUPPRESS BIT IN TSKARG
EXIT

Figure 9. TYPICAL INPUT DEVICE INTERRUPT ROUTINE

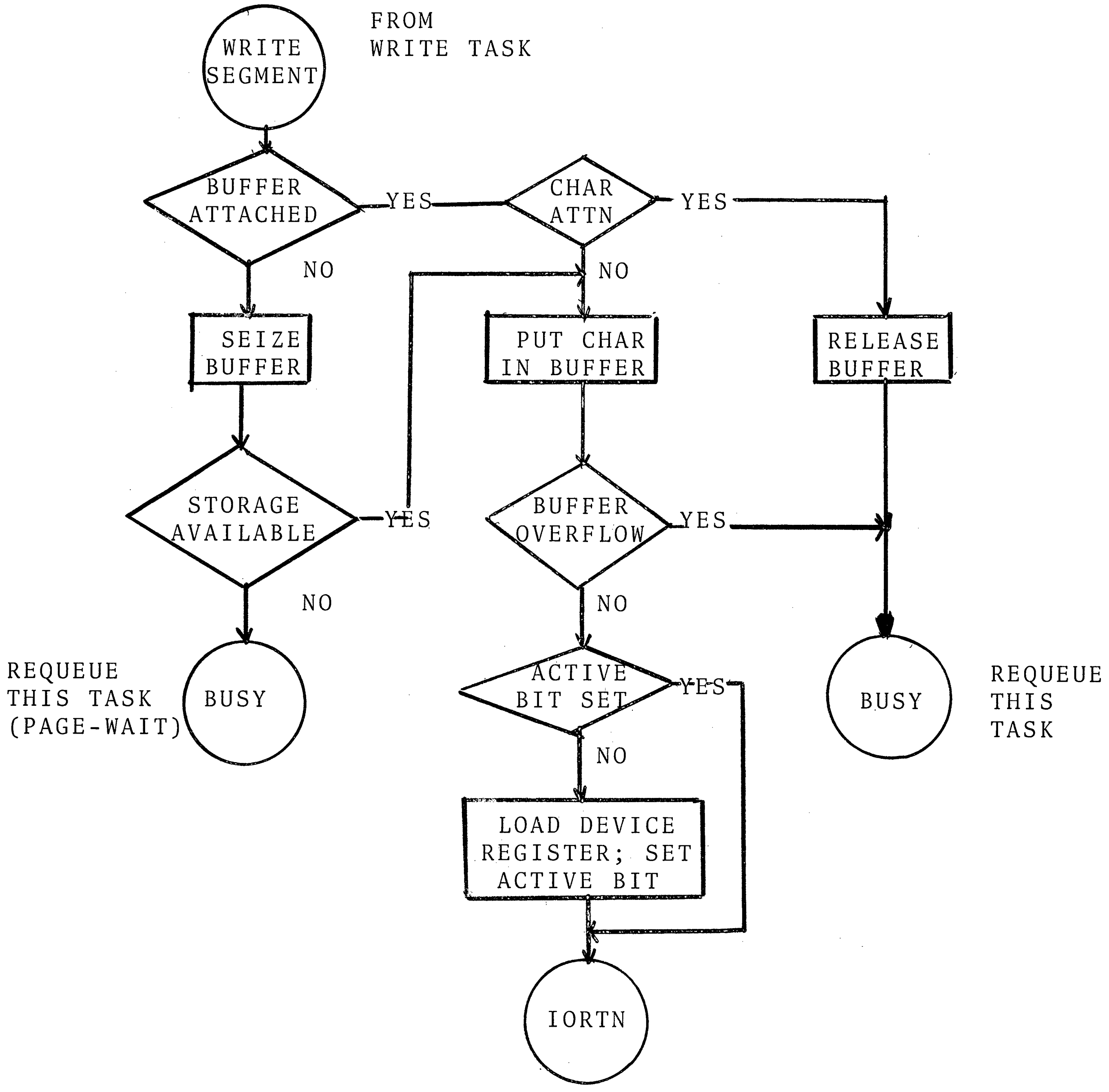

Figure 10. TYPICAL WRITE SEGMENT

unsnarls its own problems. In any case, if buffer storage is either not available or the attached buffer is full, the task is requeued.

Note the use of an "active bit" to identify whether a device has a pending interrupt or not. Setting this bit is equivalent to arming the device. If the device is idle when this I/O segment is activated, then the device is started and the first character is loaded in its buffer. Following this operation, normal device interrupts following each transmitted character cause the restarting and reloading procedures to continue.

A typical output interrupt routine is shown in Figure 11. This routine is called by the interrupt identifier when a service request for this device has been recognized. This routine has the responsibility of clearing the interrupt, loading the next output character (if any) and arming the device so that the next character (if any) can be transmitted. Note that if an attempt is made to fetch a character from an empty buffer, the device is not restarted and the attached buffer storage is returned to the allocatable page pool.

The manner in which these specialized I/O device support segments operate in conjunction with the remainder of the system is illustrated in Figure 12, which shows a typical situation in which requests for device activity are queued for both transmission and reception. The process involved here is a store-and-forward operation between two remote machines using a PDP-8/RAMP configuration as the store-and-forward element. In this diagram the vertical axis represents time, increasing toward the bottom of the diagram. Across the top, several concurrent processes are identified as the real- and task-time activity for a pair of devices called A and B. The primary operations represented here are

1. real-time transmit/receive activity for device A,

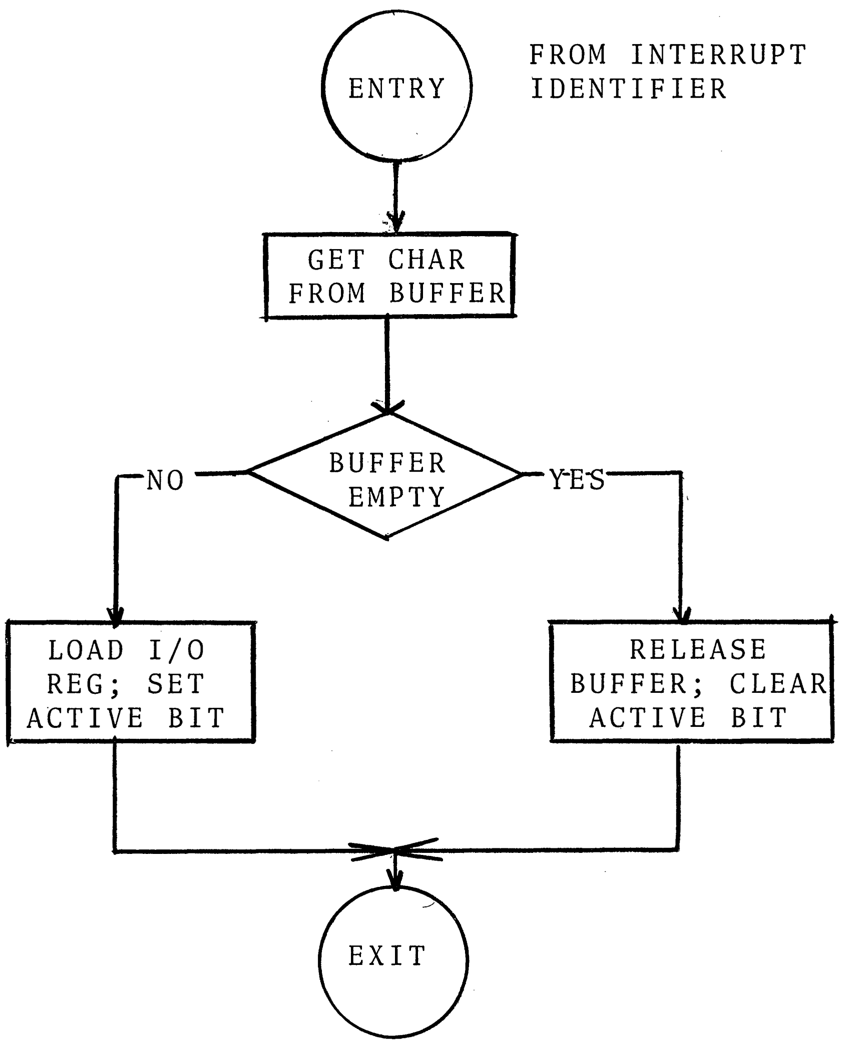
ENTRY
FROM INTERRUPT IDENTIFIER
GET CHAR FROM BUFFER
BUFFER EMPTY
NO
YES
LOAD I/O REG; SET ACTIVE BIT
RELEASE BUFFER; CLEAR ACTIVE BIT
EXIT

Figure 11. TYPICAL OUTPUT DEVICE INTERRUPT ROUTINE

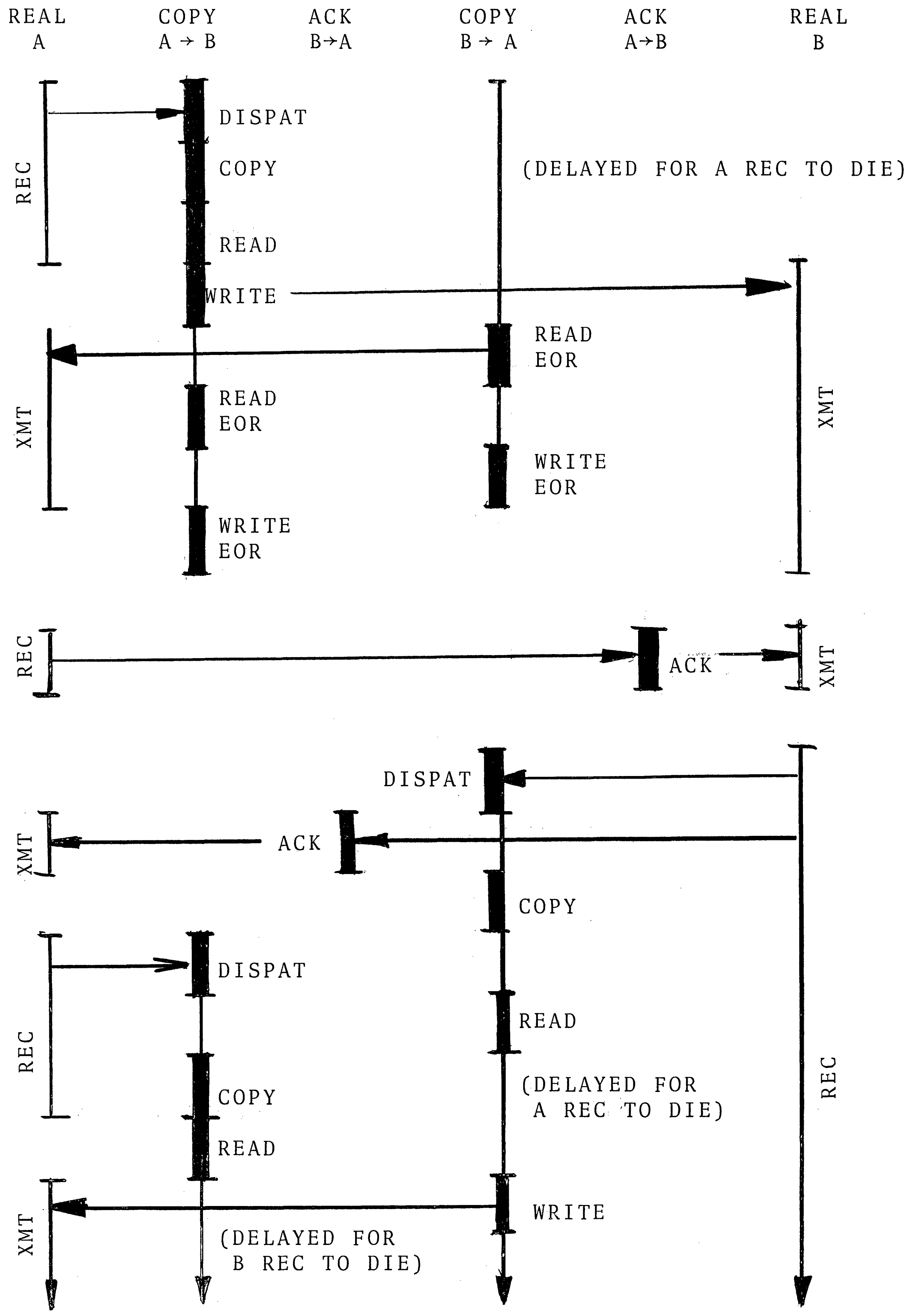

Figure 12. STORE-AND-FORWARD PROCESSING

2. task-time activity for a copy operation from A to B ,

3. record-acknowledgment task-time activity for the copy operation from A to B ,

4. task-time activity for a copy operation from B to A ,

5. record-acknowledgment task-time activity for the copy operation from B to A , and

6. real-time transmit/receive activity for device B.

Although real-time operations may proceed simultaneously for both A and B , only one active task-time operation is possible at any instant. The particular active task is identified in the figure as a heavy line; other inactive tasks are identified as light lines.

Reading Figure 12 from top to bottom, an input operation is first detected on device A in real time. The interrupt routine then inserts the dispatcher task which in turn calls on the copy task. Following device seizure, alternate calls on READ and WRITE then occur which copy the incoming text from device A's buffer to device B's buffer. Meanwhile, device B is started in real time to begin transmitting the text.

Meanwhile, a second copy task, presumably inserted at some past time by device B , has been held in the inactive state because device A has not been available for transmission. Once the record received from A has been terminated, however, device A is presumably ready to begin transmission. Note the manner in which the tail end of the task-time copy from A to B is interleaved with the beginning of the task-time copy from B to A .

Following transmission of both the A-to-B and the B-to-A records, devices A and B each wait for record acknowledgment from the remote machine to be transmitted over the respective data link. The figure shows that device A transmits its acknowledgment as a single-character record, while device B embeds its acknowledgment in the midst of a text record. Note the parallel processing of the acknowledgment and the text as received from B , and the text record as received from A.

3.6 Error Recovery

Situations can arise during normal system operation which result in a failure of the acknowledgment to be returned by the sink DSR in response to an EOR from the source DSR. This may be due to a buffer overrun condition in the sink device, a very long processing time for the message at the sink device itself (say due to human response-time at a terminal), a hardware or transmission circuit failure, or a wait-before-transmit request on the part of the sink device. Under these conditions the source device has the option, usually initiated by human intervention at a terminal, of transmitting a special message which has the effect of clearing all device paths and purging the sink device of a possibly jamming condition. This message is represented by a special character which has the halt modifier set. When this character is detected in real time by the source DSR, a special dispatcher task is inserted in the task queue, and the halt character is inserted in the input buffer. The special dispatcher task causes a WRITE task to be invoked which immediately kills transmission on the coupled sink LDN and sets a bit in the source DCB entry which causes all further transmission to be ignored. As a result, all outstanding system outbound-text activity referencing the source device will eventually run dry for lack of input text. At this time, the halt character previously inserted in the input buffer is transmitted to the sink DSR, which then resets the halt indicators and resumes transmission.

This function is useful in connection with machine-to-machine communication where asynchronous interrupts to operating systems are required, and in particular to a time-sharing system such as MTS, where the attention operation from a terminal, perhaps a RAMP system, suspends execution of a system program and returns control to the terminal.

Note that the sink DSR is unaware that the attention condition has occurred until the attention character is in fact transmitted to the device. The WRITE task itself diverts the output text stream, and the device support segment is not called during the interval between execution of the special dispatcher task and the appearance at the WRITE task of the halt character. If the source device is in command mode, then the halt character does not appear at a WRITE task unless the ECHO is invoked. Except in this case, then, the only effect of the halt operation from a device in command mode is to suspend the associated sink device transmission at the WRITE task. The convention has been adopted that if the halt operation occurs a second time, the WRITE task resumes transmission, again without the attention character being transmitted to the sink device. This feature has been highly useful within the RAMP system itself during debugging operations.

IV. TIMING AND LOADING CONSIDERATIONS

A large-scale PDP-8/RAMP system supporting a number of I/O devices, most of which are transmitting concurrently on a character-by-character basis, would probably not be described as a high-speed system. Currently operating systems using the PDP-8 can realize an aggregate throughput of about 400-1000 bytes per second, with peak rates during any one record of about 3000-10,000 bytes per search. If DSR support for a pair of devices is organized about a record-by-record basis using specially tailored buffer transmission routines, then the peak rate can be realized in the aggregate for that pair of devices. If data break operations are possible for a particular device, then the character interrupt time included here can be deleted, making the peak rate well up in the tens-of-kilobytes range. The following discussion summarizes only those considerations applying to character-by-character transmission operations using devices that interrupt on every character.

The timings indicated below were measured using special equipment attached to a PDP-8/RAMP system operating as a store-and-forward environment and including a System/360 interface, teletype adapters, medium-speed (2000 baud) synchronous data sets, and miscellaneous other adapters. This particular system, chosen as an example of a very heavily loaded system, includes two banks of core memory, one of which includes the buffer management routines and the buffers themselves, and the other of which includes the basic system nucleus, copy nucleus, and device support routines. The overhead involved in core-bank switching contributes from 30 to 90μS in the figures shown below. All timings indicated apply to the PDP-8. Those for the PDP-8S can be derived very closely by multiplying the PDP-8 timings by 15.

The cyclic buffer routines operate with the interrupt system disabled, so that they can be called directly by real-time routines and, after toggling the interrupt system, by task-time routines. The operation of fetching or storing a single character in a cyclic buffer requires about 200μS. This, added to about 100μS required for interrupt identification, represents effectively the minimum time that any one particular device may have to wait before another outstanding interrupt is cleared. If a buffer is either empty or full, only about 70μS is required to establish the fact.

When the system is idling with no call requests in the task queue,a 113μS loop is in process, with the interrupt system being disabled for the majority of this time. About

300μS is required to insert a call request in the task queue, and about 425μS is required to switch between one task and the next in the task queue.

The common routine processing in connection with the READ and WRITE tasks involves an overhead of about 150μS per character, during which time the interrupt system is enabled. Thus, assuming

1. an interrupt on every character which is then placed in a cyclic buffer, and

2. a task which fetches this character from the buffer,

a total of 300μS is required in the interrupt routines and 875μS is required in the task switching process. The peak input character rate then is limited to about 3300 characters per second on any one device, and the aggregate character rate to about 1100 characters per second considering all devices in the system. A simple analysis of queueing delay shows that if N is the number of devices in the system considered above, then 300 x N(μS) would be the worst-case waiting time for any one interrupt to be honored, and 1175 x N(μS) would be the worst-case delay for task-time processing. Since store-and-forward operations on a character-by-character basis require two such transactions, one for the source device and the other for the sink device, an aggregate throughput of about 400 characters per second seems to represent the upper limit on performance for the particular PDP-8/RAMP system analyzed here.

REFERENCE

1. Mills, D., _RAMP: A PDP-8 Multiprogramming System for Real-Time Device Control_, Concomp Project Technical Memorandum, The University of Michigan, May 1967, 24 pp.