To: Distribution

From: F. A. Canali

Date: March 12, 1974

Subject: New Tape DCM

ORGANIZATION OF THIS DOCUMENT

This document is divided into three parts. The first, an introduction, discusses the organization and purpose of this document, motivation for the proposals made herein and the logistics for implementation of the proposals. The second part discusses the design of user interfaces and the third part discusses special interfaces.

PURPOSE

The purpose of this document is to discuss motivation for designing an interface between the ANSI standard tape DIM and the current tape device control module (tdcm) which may eventually be used as a replacement for the current tape device control module.

MOTIVATION

The current tape device control module provides minimally adequate functional capability for present and foreseeable use of tapes on the Multics system. However, desirability of a piece of system software is not founded on functional capability alone, but also on a number of other factors, collectively called aesthetic appeal, a major part of which is the ability to easily support new applications of system software (including modifications to current applications.)

Three factors affect the ability to support new applications.

---

i) Ease of use.

The interface between applications and the system should be clean and simple. System software should provide adequate services of a routine nature so that applications of the system need only be concerned with their own particular tasks and require a minumum of fussing with the system interface. In no case should system software add unnecessary "hair" to an interface even in the name of efficiency.

ii) Ease of understanding.

Application programs should be able to be written and modified with a minimum investment of programmer time in studying or referencing system interfaces.

iii) Integration into the total system.

The use of a particular piece of system software should feel natural to programmers using it who are familiar with the overall system.

The current tape DCM communicates with users primarily through the tseg data base. Perusal of the tseg data base documentation, reproduced in appendix C of this document, should convince anyone that its use is somewhat less than clean and simple. A user of the current tape DCM has only one entry point which he calls to perform reading, writing and tape positioning, which of these functions to be performed being determined by tseg stuffing. In addition to this unnecessary "hair" a user of the current tape DCM must keep track of his own buffers within the tseg data base, a routine and unnecessary task.

The new tape DCM proposed in this document will adopt the philosophy of supplying separate entry points for various tape operations to be performed instead of tseg stuffing. There will be a structure passed between user and DCM which will be much smaller than the current tseg and will contain only data not usually subject to change during the operation of a program using a tape. Buffer allocation and optional error recovery will be handled by the DCM, obviating the need for user programs to perform this onerous task.

The new tape DCM will be designed to be fully integrated with the tape registration and mounting package now being written. Thus, the implementation of the DCM proposed in this document will present users with a more unified and internally consistent tape interface.

LOGISTICS

It is believed that the interface herein presented can be implemented in rings higher than the hardcore ring.

Let me make it perfectly clear that three phases of installation are proposed and that while it is believed that the interfaces presented in this document have been designed in sufficient detail and generality to allow replacement of the current tape DCM the author will only accept responsibility for installation of the first phase.

Phase one.

The new DCM would be installed with the ANSI standard tape DIM. It would not replace the current tape DCM but would merely provide a simpler interface to it for the new ANSI standard tape DIM. The new DCM would use the current DCM to actually perform tape operations, the tseg data base and buffers being hidden in the segment used to communicate with user programs.

Phase two.

The new DCM would replace the current DCM and would be the sole interface between user programs and the tape drives. Buffers would remain hidden in the segment used to communicate with user programs. Data would still be moved to/from wired down buffers for input and output and part of the new tape DCM may have to be in ring zero through this phase. A simulator of the current tape DCM, which will run in the user ring, may be written for unconverted user programs.

Phase three.

An interface with the new GIM which is now under design would be incorporated into the tape DCM.

BUFFERS

Input and output of physical records would be provided through buffers. A call to read a record would result in the address of a buffer of information being passed to the calling user program (usually, and hereinafter refered to as, a DIM.) Similarly, to write a physical record a DIM would make a call to tapeio_ to allocate a buffer for use by the DIM for data stuffing, followed by a call to tapeio_ to write the stuffed buffer. The number of buffers available to a DIM may depend on the maximum buffer size the DIM expects to use, a quantity the DIM specifies during initialization of the tapeio_ package. A maximum buffer size of 8192 characters is currently planned.

A special buffer, called the error recovery buffer, will always be available to a DIM for use in recovering from conditions such as end of volume. This buffer will participate in output only on a synchronous basis and no guarantee will be made concerning the integrity of its contents after participating in a write operation. The error recovery buffer will always be 8192 characters in length regardless of the maximum block size specified by a DIM during initialization.

Buffers used by tapeio_ may be considered to be in one of five states:

(1) Free - A buffer contains no useful data and is available for allocation to a DIM when a write buffer is requested, or is available for use as a read buffer.

(2) Allocated - A buffer is in use by a DIM. It may have been allocated as a write buffer for the DIM or given to a DIM when a read request was issued and the buffer contained the desired data from the tape.

(3) Busy - A buffer is participating in an input or output operation. Either it is on the write behind queue and contains data to be written to tape or it is on the read ahead queue and data will be read from tape into the buffer.

(4) Suspended - A buffer contains data but because of an error on tape write behind or read ahead was terminated. A suspended buffer will be allocated to a DIM only if a buffer is needed and no free buffers exist. If any suspended buffer is allocated all other suspended buffers will have their status switched to free.

(5) Chaos - A buffer is in a state of chaos if a write request is made specifying the buffer and a tape error is signaled. The buffer remains in a state of chaos while a DIM handles the condition raised. If return is made from the condition the buffer is switched to the busy state by being put onto the end of the write behind queue. If no return is made from the DIMs condition handler (I.E. a DIM performs a non-local goto out of the condition handler) the buffer will be assumed to be free. See the discussion of error handling.

ENTRY POINTS

This section describes a list of proposed entry points to tapeio_ which will be available through gates, with calling sequences and a short description of what each does. It should give the reader the general flavor of the proposed DCM. Arguments used are:

cP - Control pointer. A pointer to the control structure, given to a DIM by the tape registration and mounting package, used to identify a tape drive.

bP - Buffer pointer. A pointer to a buffer which is available for writing or which contains data which has been read from tape.

oP - A pointer to a structure used in describing order codes to perform.

code - A return code given on each call to a tapeio_ entry point which is an error_table_ code.

count - A count of characters in a buffer. It is a fixed bin(17) number.

sP - A pointer to the software status block.

ring_# - A ring number. It is a fixed bin(3) number.

hs - A bit(72) hardware status.

The following are the proposed entry points available to a DIM.

get_buffer(cP,bP,code)

This entry will allocate a write buffer for a DIM and return a pointer to the buffer. If no free buffer exists which can be allocated then:

(i) If read ahead buffers exist the most recently read buffer in the read ahead queue will be returned and the tape backspaced one record.

(ii) if write behind buffers exist tapeio_ will wait until a write buffer becomes free and return that buffer.

(iii) If a suspended buffer exists all suspended buffers are marked as free and one of these will be given to the DIM.

(iv) All buffers are already allocated to the DIM so the code is set to indicate an error and return is made to the DIM with the buffer pointer set to null.

release_buffer(cP,bP,code)

This entry will change the state of the specified buffer from allocated to free and the buffer pointer will be reset to null.

read(cP,bP,count,code)

This entry will return a pointer to the next logical record and set the count to the number of characters in the record. If read ahead was not previously in effect it will be initiated. If the previous call to tapeio_ was an output operation (write, write eof) no buffer will be returned and a logic error will be signified in the return code but write behind will continue.

write(cP,bP,count,code)

This entry will place the specified buffer on the write behind queue and eventually write the number of characters specified to tape. If read ahead was previously in progress all read ahead buffers will be freed and the tape backspaced the appropriate number of records before writing begins.

synchronize(cP,code)

This entry will synchronize tape operations before returning. If read ahead is in effect all read buffers are freed and the tape backspaced appropriately before returning. If write behind is in effect return will not be made until all writes are complete and all the write behind buffers have been freed.

order_code(cP,oP,code)

This entry will perform order codes on a tape in a write behind fashion.

software_status(cP,sP,code)

This entry will return a pointer to a structure describing the software status of tapeio_.

synchronous_read(cP,bP,count,code)

This entry point will be similar to the read entry point except that the read operation is performed in a synchronous manner.

synchronous_write(cP,bP,count,code)

This entry point will be similar to the write entry point except that the write is performed in a synchronous manner.

synchronous_order_code(cP,oP,code)

This entry point will be similar to the order_code entry point except that the order codes are performed in a synchronous manner.

promote(cP,ring_#,code)

This entry will set ring access to the tapeio_ data base and associated tape drive. It will ignore attempts to set ring access to rings lower than the calling ring.

open(cP,code)

This entry will be called by a DIM to cause tapeio_ to initialize in preparation for using a tape.

close(cP,code)

This entry will be called by a DIM to perform cleanup after having finished with a tape.

restack_buffers(cP,code)

This entry will restack all suspended operations onto the write behind queue. (See appendix A on error recovery.)

TAPE ERRORS

An error on a tape drive will be detected on the next call to tapeio_ after the error occurs. Recovery and restart of the operation from the point of error will be attempted as many times as specified in the error_retries field of the control structure (see definition of control data structure below.) If the error recurs on all retries a "tape_error_" condition will be raised and a DIM may use the software_status entry point to gain information concerning the hardware status and the status of buffers which may have been queued on the device. See appendix A for an example of a DIM error recovery procedure.

The tape_error_ condition will be raised during read ahead when a call to the read entry point is made which would access the read buffer in which the error occured. The tape_error_ condition is raised as soon as possible when it occurs during write behind.

STRUCTURES

The following structures are proposed for communication between a DIM and tapeio_.

(A) ORDER CODE LIST

```
dcl 1 order_codes based(oP),
      2 number_of_codes fixed bin(35),
      2 codes(0 refer(number_of_codes)) unal,
        3 operation char(3),
        3 count fixed bin(8);
```

The meanings attached to the various elements of this structure are:

number_of_codes - The number of order codes in the codes array.

operation - A particular operation code to be performed on the tape.

count - A count associated with each order code. It may be the number of times the order code is to be performed or an argument associated with a particular order code as a modifier.

See appendix B for a list of the proposed order codes and the significance of the count for each.

(B) SOFTWARE STATUS

```
dcl 1 software_status based(sP),
      2 volume char(6),
      2 file fixed bin(17),
      2 error_buffer ptr,
      2 counts,
        3 total fixed bin(17),
        3 free fixed bin(17),
        3 allocated fixed bin(17),
        3 busy fixed bin(17),
        3 suspended fixed bin(17),
      2 hardware_status bit(72) aligned,
      2 status_code fixed bin(35),
      2 suspended_ops fixed bin(17),
      2 suspended_buffers(0 refer(suspended_ops)),
        3 buffer_ptr ptr,
        3 count fixed bin(17),
        3 error_count fixed bin(17),
        3 suspended_order_codes unaligned,
          4 number_of_codes fixed bin(35) initial(1),
          4 operation char(3),
          4 count fixed bin(8);
```

The meanings attached to the various elements of this structure are:

volume - The volume identification of the attached tape.

file - The file number of the current file on the tape.

error_buffer - A pointer to the special 8192 byte error recovery buffer.

total - The total number of buffers which exist in all states. (excluding the error recovery buffer)

free - The number of free buffers available to a DIM.

allocated - The number of buffers which have been allocated and are in use by the DIM.

busy - The number of buffers participating in input or output operations.

suspended - The number of buffers which have been suspended due to an error on the tape.

hardware_status - The hardware status for the error associated with the suspended buffers.

status_code - An error_table_ type of code indicating the nature of the hardware status.

suspended_ops - The total number of operations which have been suspended due to a tape error. It includes the number of read or write operations with associated suspended buffer plus the number of order code operations which have been suspended.

buffer_ptr - A pointer to a suspended buffer.

count - A count of the number of characters of data in the suspended buffer.

error_count - The offset in the suspended buffer at which the error occured, if available, or zero.

suspended_order_codes - An order code which was suspended, suitable to be given directly to the order_code entry point as an order code structure. It contains only a single order code which was suspended.

(C) CONTROL DATA

```
dcl 1 tapeio based(cP),
      2 user_area_ptr ptr,
```

```
2 user_area_length fixed bin(17),
2 buffer_size fixed bin(17),
2 error_retries fixed bin(17),
2 modus_operandi,
  3 mode fixed bin(17),
  3 synch_flag bit(1),
  3 start_read_ahead bit(1);
```

The meanings attached to the various elements of this structure are:

user_area_ptr - A pointer to a region within the tapeio_ data base which is available to a DIM.

user_area_length - The length of the region pointed to by user_area_pointer.

buffer_size - The maximum block size a DIM expects to handle. The number of buffers available to a DIM is computed from this figure at the time of the DIMs call to the open entry point.

error_retries - This number specifies the number of times tapeio_ will retry an operation in error before giving up and raising the tape_error_ condition.

mode - This specifies the mode of read or write operations.
0 - binary
1 - 9 mode
2 - ascii/ebcdic (1)
3 - bcd
4 - ebcdic (1)
5 - ascii (1)

synch_flag - This switch specifies that operations are to be performed synchronously.

start_read_ahead - If this switch is set during a DIMs call to the attach entry point read ahead is begun before returning.

---

(1) Translate feature must be installed at an installation in order to use this mode.

NOTES

There will be several ways of doing synchronous input and output. The synchronous_read, synchronous_write and synchronous_order_code entry points are provided for convenience. Another method of performing synchronous tape operations will be to set the synch_flag before the call to the attach entry point and leave it set thereafter. A DIM may perform synchronous output simply by using the error recovery buffer for all calls to the write entry point. This method, however, is discouraged since it inhibits error recovery. Any DIM using this method will do so at its own risk.

Calls to the write and release_buffer entries will cause the buffer pointer passed to be reset to null unless the pointer specifies the error recovery buffer.

The synch_flag will supercede the start_read_ahead flag during calls to attach. If both are set no read ahead will begin.

An end of file condition on a read operation will cause an error code to be passed back from a read or read_synchronous call, rather than the tape_error_ condition being raised.

THE RING ONE INTERFACE

The ring one interface will exist primarily for use by the tape registration and mounting package. It will deal with tape drive allocation and drive dependent information which should not be alterable in higher rings. Of course, ring one procedures will have the interfaces defined in the previous section of this document available, the interfaces defined in this section being in addition to those defined in part 2.

EXPANDED USER INTERFACE

An expanded list of order codes will be available for ring one procedures calling the order_code entry point. Appendix B indicates which particular order codes will be executable from ring one only.

No checking will be done which prevents a ring one caller from destroying the tape label.

ENTRY POINTS

The list of entry points defined in this section is proposed as an interface to the ring one tape registration and mounting package. Arguments used are as stated in the corresponding section of part 2 of this document with the following additions:

dP - This signifies a pointer to a user defined area having the same structure as the tape data block defined below. The tape data block will contain information relating to the status and characteristics of individual devices.

drive_# - A fixed bin(6) number containing the device address of a tape drive

The ring one entry points are:

attach(cP,code)

This entry point will allocate a drive from the list of drives available for user use. The control segment for the drive is created and its pointer is returned to the caller.

detach(cP,code)

This entry point returns a drive to the list of unused devices on the system. Cleanup is performed, the control segment is deleted and the control pointer reset to null.

priv_attach(cP,code)

This entry point is similar to the mount entry point except that reserved devices are also available for allocation to the caller and no limit is enforced on the number of drives which may be attached to a process.

add_drive(dP,code)

This entry point adds a drive to the list of devices attached to the system. A tape data block will be created and the information pointed to by dP will be used to define device characteristics (e.g. 9 track, reserved.)

delete_drive(drive_#,code)

This entry point will delete a drive from the list of devices available on the system.

tape_data_lookup(cP,dP,code)

This entry point will look up the tape data block associated with a tape drive and copy it into the area pointed to by dP. If the control pointer is non null the drive associated with the given control segment will be looked up. If the control pointer is null the drive number in the structure pointed to by dP will be used for the lookup.

tape_data_change(dP,code)

This entry point will cause a tape data block overlay. The tape data block having the same device address as in the structure pointed to by dP will be overlayed with the information pointed to by dP.

kill_io(cP,code)

This entry point stops all tape operations as soon as possible and frees all buffers. The tape will be left in an anomalous state. This entry should be called only in very special cases (e.g. when a user hangs up) and with care.

TAPE DATA BLOCK

An additional structure defined for ring one interfaces is the tape data block. It exists within a ring one segment and there is one tape data block for each tape drive on the system. The following format is proposed for the tape data block:

```
dcl 1 tape_data_block based(dP),
      2 mount_time fixed bin(71),
      2 dcw_list ptr,
      2 process_id bit(36),
      2 error_count fixed bin(17),
      2 control_ptr ptr,
      2 file_no fixed bin(17),
      2 volume char(6),
      2 drive_no fixed bin(6),
      2 density fixed bin(8),
      2 drive_flags,
        3 reserved bit(1),
        3 track9 bit(1),
        3 ringin bit(1),
        3 protected bit(1);
```

Where the variables in the above structure have these meanings:

mount_time - The time at which the tape on this drive was mounted.

dcw_list - A pointer to a ring one area available for building dcw lists.

process_id - The process id of the process which mounted the tape.

error_count - The count of errors occuring on the associated tape drive since the mount was accomplished.

control_ptr - A pointer to the control segment associated with this tape drive.

file_no - The file number of the current file on the tape mounted on the drive associated with the tape data block.

volume - The volume identification of the volume currently mounted on the tape drive associated with the tape data block.

drive_no - The hardware address of the tape drive associated with the tape data block.

density - The current density of a tape mounted on the associated drive. It is the same as the count in the last set density order code issued. See appendix B.

reserved - A flag which is set to one if the associated tape drive is reserved for allocation through the priv_mount entry point.

track9 - A flag which is set to one for 9 track drives and zero for 7 track drives.

ringin - A flag which is set if the tape has a ring.

protected - A flag which is set if a set file protect order code was issued for the tape.

ERROR RECOVERY

An error in reading or writing a tape which exhausts the error count will result in a tape_error_ condition being raised. A DIMs condition handler may call and software_status to gain information which will enable it to analyze the error and restart the tape operation. For errors on read there will be only one suspended buffer and that will be the buffer in error. For errors on write the first suspended buffer will be the one in error and subsequent suspended buffers will be succeeding write behind buffers which existed at the time of the error.

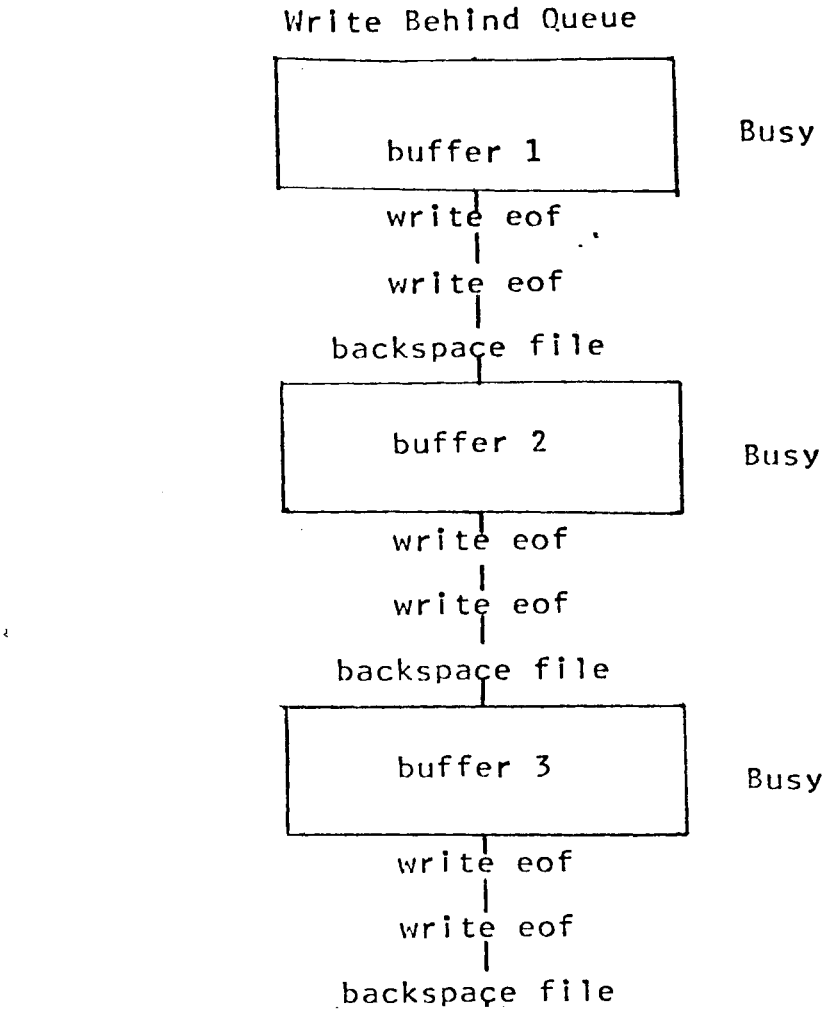
A DIMs condition handler may reposition the tape as it wishes and reissue read or write calls. If return is made from a DIMs condition handler tapeio_ will carry on whatever operation it was called to perform before it raised the tape_error_ condition. Specifically, if a call to write was made, the buffer tapeio_ was called to write (which was in a state of chaos during error recovery) would be stacked at the end of the write behind queue and if a call to read were made the next record from the tape would be read. Thus, a read operation may be restarted by calling order_code to backspace the tape and returning from the condition handler. A write operation may be restarted by backspacing one record, restacking all suspended buffers with calls to write or order_code and returning from the condition handler.

EXAMPLE

Suppose a DIM is written which puts records on tape separated by file marks. After each record is written two file marks are written and the second file mark is backspaced over if the DIM decides to write another record.

Also suppose that at some point in time our hypothetical DIM is running and has caused three buffers to be queued for write behind, together with apprropriate order codes, and the DIM has just gotten a fourth buffer to be filled and written. The situation would be outlined as in diagram A.1.

APPENDIX A



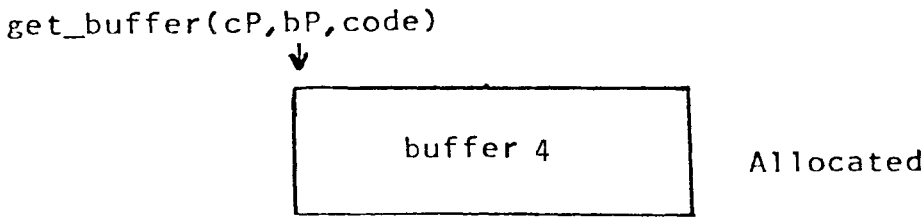


Diagram A.1

Now suppose the DIM makes a call to write buffer 4 and the tape_error_ condition is raised because of an error in writing the second buffer to tape. If the DIM makes a call to the software_status entry point diagram A.2 would outline the situation at this point.

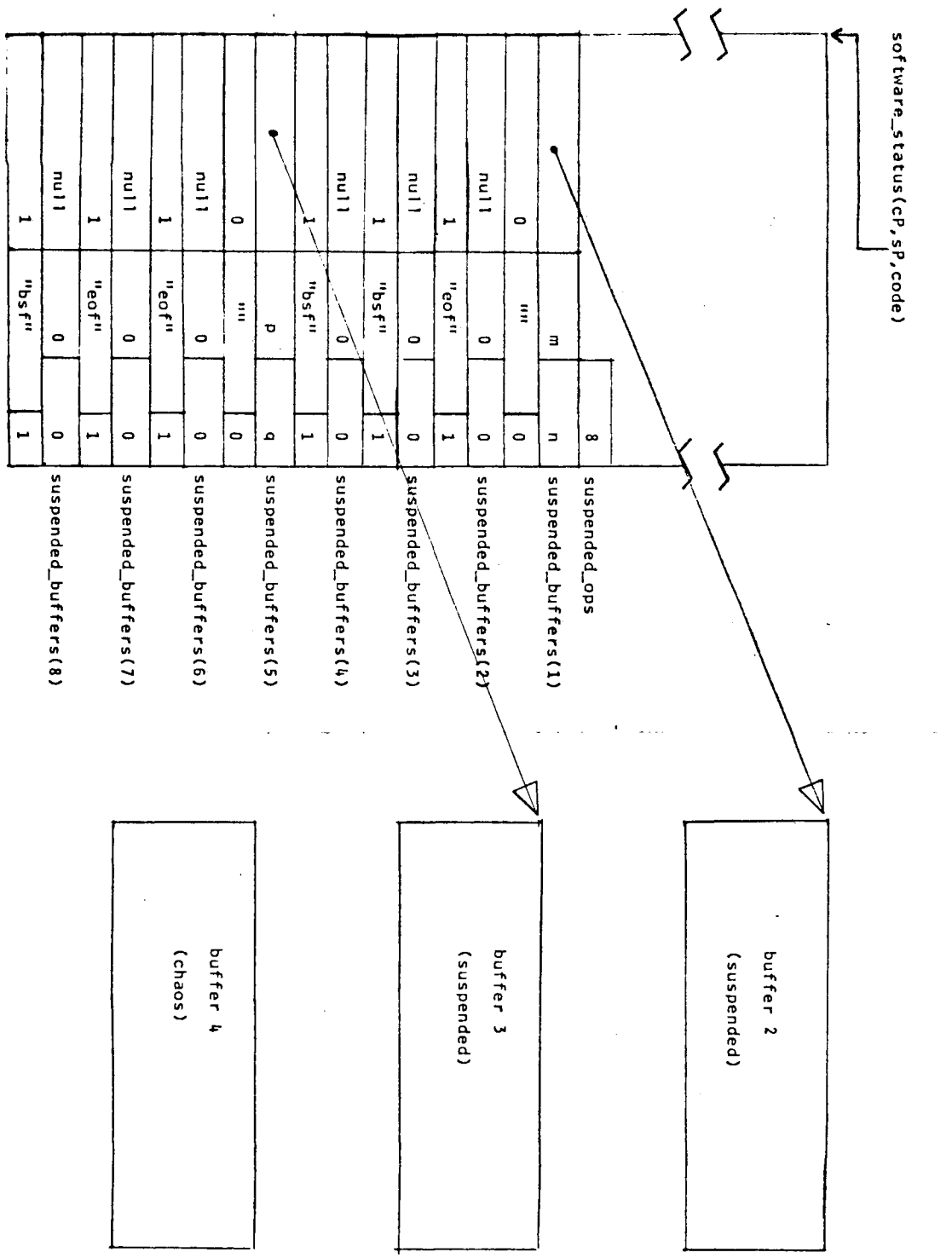

software_status(cP,sP,code)
suspended_ops
8
suspended_buffers(1)
n
0
""
m
suspended_buffers(2)
0
0
"eof"
1
suspended_buffers(3)
1
0
"bsf"
1
suspended_buffers(4)
1
0
null
1
suspended_buffers(5)
q
0
""
p
suspended_buffers(6)
0
0
null
1
suspended_buffers(7)
1
0
"eof"
1
suspended_buffers(8)
0
0
null
1
"bsf"
1
buffer 2
(suspended)
buffer 3
(suspended)
buffer 4
(chaos)

Diagram A.2

If the DIMs error recovery consists of simply restacking all operations which have been suspended (similar to tape_) and returning from the on unit then the situation would be as outlined in diagram A.3

Note that if the error is end of tape the error recovery buffer could have been used to write trailer labels on the tape in error and header labels on a new (continuation) tape before restacking all suspended operations.

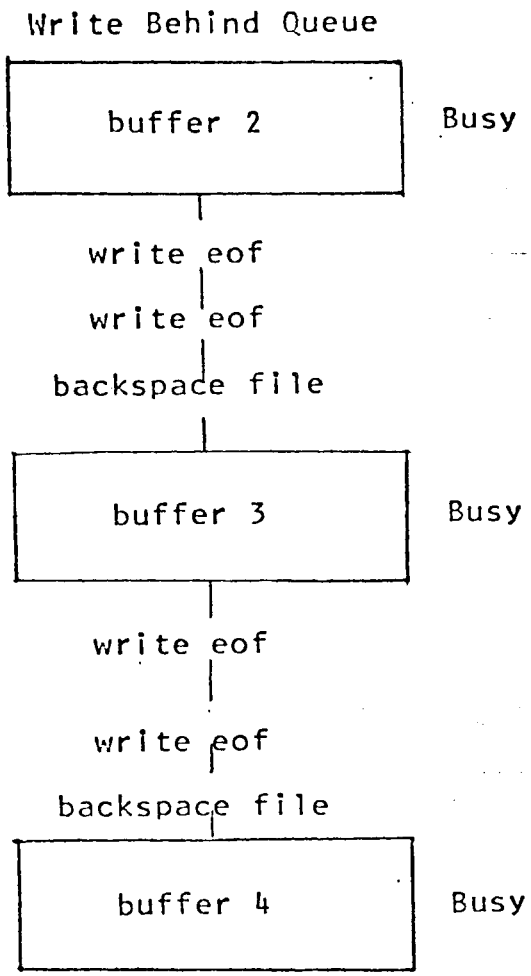

Write Behind Queue
buffer 2
Busy
write eof
write eof
backspace file
buffer 3
Busy
write eof
write eof
backspace file
buffer 4
Busy

Diagram A.3

ORDER CODES

OPERATION	CODE	MEANING OF COUNT
Request Status	rqs	None
Rewind	rew	None
Set Density	sdn	0=200 bpi, 1=556 bpi, 2=800 bpi, 3=1600 bpi (1)
Forward Space Record	fsr	Number of records to forward space.
Backspace Record	bsr	Number of records to back space.
Forward Space File	fsf	Number of files to forward space.
Backspace File	bsf	Number of files to backspace.
Erase	ers	Number of erasures to be done
Write Eof	eof	Number of file marks to be written
Set File Protect	pro	None

The following order codes will only be performed for callers in ring one.

Reset Status	rts	None
Request Device Status	rqd	None
Reset Device Status	rtd	None
Set File Permit	per	None
Tape Load	lod	None
Rewind/Unload	run	None
Reserve Device	rsv	None
Release Device	rel	None

No plans are currently made to support the following instructions from the regular instruction set as order codes or in any other way:

Servey Devices	Load From Device
Read Control Registers	Diagnostic mode control
Write Control Registers	Main Memory overlay
Control Store Overlay	Device Wraparound
Write Timing Character	

No plans are currently made to support the following controller instructions as order codes or in any other way:

Suspend Controller	Read Lock Byte
Release Controller	Write Lock Byte

---

(1) 3 invalid for 7 track tapes. Ring one may change density on 7 track tapes at will and on 9 track tapes at load point. Higher rings may change density similarly but only on unregistered or unlabeled tapes.

Initiate Read Data Transfer	Conditional Write Lock Byte
Initiate Write Data Transfer	Write Controller Main Memory (BIN)
Read Controller Main Memory (ASCII)	Read Controller Main Memory (BIN)
Write Controller Main Memory (ASCII)	

NOTE: The method defined of using character definitions of order code operations leaves open the possibility of software defined order codes. (E.G. "eot" for write end of tape - 2 tape marks followed by a backspace file) None have been defined in this document since it is not clear at the present time which, if any, software defined order codes would be generally useful. Software defined order codes may be the subject of a future MTB by this author.

Name: tseg

This data base is used by the tape Device Control Module (DCM) as its main argument. It resides in the user ring and its pointer is passed to the hardcore tape DCM modules. The user creates this data base as a single segment or area within a segment. One tseg data base must be provided for each drive that is attached.

Usage

```
declare 1 tseg based (tsegp) aligned,
          2 areap ptr,
          2 ev_chan fixed bin(71),
          2 write_sw fixed bin(1),
          2 sync fixed bin(1),
          2 get_size fixed bin(1),
          2 drive_number fixed bin(6),
          2 buffer_offset fixed bin(12),
          2 buffer_count fixed bin(12),
          2 completion_status fixed bin(2),
          2 hardware_status bit(36) aligned,
          2 error_buffer fixed bin(12),
          2 command_count fixed bin(12),
          2 command_queue (10) fixed bin(6),
          2 bufferptr (12) fixed bin(18),
          2 buffer_size (12) fixed bin(18),
          2 mode (12) fixed bin(2),
          2 buffer (12) bit (9792) aligned,
          2 dsm_area area ((1));
```

1) areap — is a pointer to the user structure in dsm_area. (Input/Output)

2) ev_chan — is the event channel name for signalling by the DCM from ring 0. (Input)

3) write_sw — is set to 1 if writing and to 0 if reading. (Input)

4) sync — is set to 1 if synchronous service from the DCM is desired and to 0 otherwise. (Input)

5) get_size — is set to 1 if record sizes are to be returned in the buffer_size array. (Input)

6) drive_number — is the physical drive number assigned by the DCM. (Output)

7) buffer_offset — is the offset from 1 of the first buffer to be processed. (Input)

8) buffer_count — is the number of buffers to be processed. (Input)

9) completion_status — is a returned value which is 0 for no pending I/O or no status avilable, is 1 for normal termination of I/O, and is 2 for nonzero major status from the previous I/O call. (Output)

10) hardware_status — is the 12 bit major and minor hardware status. (Output)

11) error_buffer — is the buffer number in which the I/O error occurred. (Output)

12) command_count — is the number of commands in the command array to be executed. (Input)

13) command_queue — is an array of commands for the tape controller. (Input)

14) bufferptr — is an array of relative pointers to the data buffers. (Input)

15) buffer_size — is an array of the sizes of the array of buffers. (Input/Output)

16) mode — is an array of read/write modes for the buffer array.

    0 = binary;
    1 = bcd;
    2 = 9 track. (Input)

17) buffer — is the array of buffers. (Input/Output)

18) dsm_area — is an area for user-defined I/O structures. (Input/Output)

(END)