27
3-17-78
250 NTIS

IS-4312

MASTER

Programmer's Guide for DISKORG:
The Disk Organization
Subsystem for the
PDP-15 Computer

By

Kent R. Brobst, Leonard C. Moon,
Thomas Ekberg, and
Thomas G. Pinter

AMES LABORATORY, DOE
IOWA STATE UNIVERSITY
AMES, IOWA

PREPARED FOR THE DEPARTMENT OF ENERGY
UNDER CONTRACT W-7405-eng-82

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Programmer's Guide for DISKORG: The Disk Organization
Subsystem for the PDP-15 Computer

By

Kent R. Brobst, Leonard C. Moon,
Thomas Ekberg, and Thomas G. Pinter

Ames Laboratory, USDOE
Iowa State University
Ames, Iowa 50011

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Date Transmitted: December 1977

PREPARED FOR THE U. S. DEPARTMENT OF ENERGY, DIVISION
OF BASIC ENERGY SCIENCES UNDER CONTRACT NO. W-7405-eng-82

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Available from: National Technical Information Service
U. S. Department of Commerce
P.O. Box 1553
Springfield, VA 22161

Price: Microfiche \$3.00

TABLE OF CONTENTS

	PAGE
ABSTRACT	iv
I. INTRODUCTION	1
II. SYSTEM OVERVIEW	1
III. PROGRAMMING CONSIDERATIONS	3
IV. ALGORITHMS FOR THE VARIOUS TASKS IN DISKORG	4
V. ACKNOWLEDGEMENTS	29
VI. APPENDICES	30
A. SYSTEM LIBRARIES	31
B. CATALOG FORMAT	32
C. DIRECTORY FORMAT	33
D. DATA SET DESCRIPTOR (DSD)	34
E. DISK ALLOCATION TABLE	36
F. DIAGRAM OF DISK ORGANIZATION	38
G. DISKORG TASK TREE	39

ABSTRACT

This report describes the disk organization package used with the Ames Laboratory Experimental Control System developed for a Digital Equipment Companies PDP-15 computer. The directory and data set structure is described. Algorithms are given for the individual routines in the package.

Distribution List:

Chicago Patent Office	1
USDOE-TIC	27
Ames Laboratory Library	5
Thomas G. Pinter	5
	<hr/>
Total	38

I. Introduction

DISKORG is a series of tasks to handle the disk organization for the ALECS PDP-15 system. It is written in the ALECS subset of PL/I and compiled on an IBM 360/65. This manual will discuss the organization of disk and the individual tasks which compose the program.

All entries are stored as fixed or character numbers. Fixed numbers will be appended with (8) when they are octal, e.g., 40(8) = 32 decimal. All names are 9 characters in length. Dates are 6 character strings supplied by the user during the program start up.

II. System Overview

The organization of data sets on disk is a two level index with the data sets linked to the second level (see Appendix F). The first level of the index is called the catalogue. The second level is called the directory. A directory and all data sets linked to it are called a library.

The catalogue contains information describing itself (see Appendix B) along with entries containing the library name and its disk octal address (DOA). The DOA points to the directory of the library. The catalogue will always contain entries for @DS, the system library, and UNIVERSAL, the shared default library.

A directory contains information describing itself (see Appendix C), a list of the data sets linked to the directory and their associated data set descriptors (DSD) (see Appendix D). The list of data set names is to simplify the search of a directory for a specific data set. The data set descriptors contain information pertaining to the data sets. The first descriptor describes the directory itself as a data set and is always named @DIRECTRY.

Data sets are either loadable or task, normal, or experiment. A loadable data set is one which was filled using LDR. LDR is a utility program which takes the object code produced by the compiler, places it in its proper form in a data set, and marks the data set as loadable. An experiment data set is used by the system to communicate with peripheral devices.

The means for allocating and deallocating disk space to data sets is by means of an allocation table (see Appendix E). This allocation table is contained in a data set called @ALLOC. The disk is allocated in terms of 400(8) word sectors. Each word in @ALLOC describes an 8 sector track on disk; bits 1-8 are set if the respective sector is allocated.

The system directory, @DS, contains DSD's for @ALLOC and @CATALOG, the catalogue. Both @ALLOC and @CATALOG are described as normal data sets with an end of file (EOF) at the end of the data set. When ALECS is running, all it needs is the DOA of the catalogue since everything else can be found from it.

III. Programming Considerations

Data sets whose names begin with @ can only be opened as input data sets. The catalogue, allocation tables, and directories are such data sets. They can be opened output or update only if the task has system access privileges. System access privileges also allow essentially variable record I/O not allowable under normal system operation.

Normally when a data set is opened, the user library name is used to fully qualify the data set. However, using the fully qualified name in the title option overrides the normal system action. Thus data sets can be opened in different libraries than the user library.

When allocating a directory two separate steps are required. First a data set descriptor in @DS is set up pointing to the directory as a data set. The data set descriptor @DIRECTORY is then set up to point to the directory data set. The DOA of the directory is entered into the catalogue. The data set is then removed from @DS, but the disk space is not deallocated. This retains the directory for access from the system.

Before a library can be deleted, all data set entries must be removed and the disk space deallocated. When the data set descriptor @DIRECTORY is removed from core, the entire library has been deleted. When modifying a directory to allow a different number of entries, a dummy directory is allocated. The old directory is then copied into the new directory, and the old directory is deleted from core.

The directories are allowed only certain maximum entries so that the directory ends on a sector boundary. The offset value of the directory is the key of the first data set descriptor. The data set descriptors start at different blocks since the list of directory entries varies with the size of the directory.

IV. Algorithms for the Various Tasks in DISKORG

The following section contains the algorithms for the separate tasks in DISKORG in alphabetical order. These algorithms are written with PASCAL like structures using English phrases to describe the over-all logic of each task. Since all variables and parameters are defined in the task listings, no term dictionaries will be included here.

The tasks are all written in ALECS, a local subset of PL/I for use in a multi-task system environment. The compiler is resident on an IBM 370/158 with an IBM 360/65 used as a front end processor. The compiler produces a card deck in column binary format which is then read onto the PDP-15. The codes described in this section perform the following tasks:

- a.) allocation, renaming, listing, updating, and deletion of directories
- b.) allocation of data sets
 - i) the input cards for task data sets are read upon allocation
 - ii) normal data sets may or may not have input data on cards
 - iii) experiment data sets are set up for communication with peripheral devices

- c.) deletion, transforming, listing of data set descriptor, and renaming of data sets
- d.) data sets in directories DISK and ALECS2 are protected from deletion by key which is required before the operation will be completed.
- e.) directories DISK, UNIVERSAL, and ALECS2 are protected against modification, deletion, and renaming permanently.

On condition blocks are used throughout the codes for several purposes. If a data set seized error is detected, the task pauses and then executes the open again. This prevents the task from abending when data sets are momentarily seized as the system does not explicitly allow shared data sets. Some routines use the array bound error to terminate search operations.

The rotate accumulator instructions are used to pack bit information into a single word. Since there is no bit to fixed conversion, all bit variables are declared fixed. However, addresses, etc., are really bit variables in logical structure. Thus, they should be considered as such even though they are declared fixed.

ADD_DIR: The purpose of this routine is to add a dummy directory to the catalogue when modifying a directory. This dummy routine becomes the directory entry for the modified directory. The catalogue entry is deleted, but the directory space is not deallocated.

Algorithm:

```

    read the catalogue header
    do until all catalogue entries are searched
        if entry is blank
            then do
                set up dummy directory
                exit loop
            end
        end
    set up directory as data set entry
    if the directory was allocated
        then do
            set up directory header
            enter the dummy directory in the catalogue
            delete the directory entry from @DS, but do not deallocate
            disk space
        end
    end ADD_DIR.

```

ADR: The purpose of this routine is to allocate a directory. The directory header is set up and entered into the catalogue. The first data set descriptor is set up as the directory descriptor.

Algorithm:

```

    11: get the maximum number of entries
        do until all allowable entries are checked
            if maximum entries is a match
                then exit loop
            end
        repeat 11:

```

```

exit:  if operation is to be completed
      then do
        do until all catalogue entries are read
          if entry is blank
            then exit loop
        end
        set up directory as data set in @DS
        if the directory was allocated
          then do
            set up directory header and data set descriptor
            delete directory entry from @DS, but do not
              deallocate the disk space
            enter the directory name into the catalogue
          end
        end
      end
end ADR.

```

ADS: The purpose of this routine is to allocate a new data set. The data set types are task, normal, or experiment. For task data sets, routine TASKLOAD is called to allocate disk space and read the task from cards. For normal or experiment data sets, the space for the data set is allocated and NEWFILE is used for entering information from the card reader. Task data sets have a record length of 256 words and experiment data sets have a record length of 32 words.

Algorithm:

```

  1:  get data set type
      if data set type is experiment
        then
          else if data set type is normal
            then get record length and number of sectors
          else if data set type is experiment
            then
              else repeat 1

```

```

    if operation is to be completed
    then do
        set up the data set
        if data set type is task
        then do
            load task from cards
            if error in load
            then delete the task from core
            else set the status word
        end
        else if data set was allocated
        then if data set was normal
            then if data set is to be loaded from card reader
            then load data set
        else do
            load experiment data set from cards
            set the status bit
        end
    end
end ADS.

```

ALLOCTABL: The purpose of this routine is to allocate disk space for data sets to be allocated. The allocation tables exist for 8 disks. There are 128 words for each disk representing the 128 8 sector tracks. Bits 1-8 are set when the respective sector of the track is allocated. Bit 0 is set to indicate that the track is full. As only 2 disks are currently in use, the allocation table bits are all set for the remaining 6 disks.

Algorithm:

```

    l1: if disk number is less than 8
        then do
            read the allocation table for the disk
    l2: set the bit mask for the excess number of sectors to
        be allocated over a track boundary
        search for a non-full track
        do until all full tracks to be allocated have been tested
            if track has any sectors allocated
            then repeat l2
        end
    end

```



```

if partial track is to be allocated
then do
  if any full tracks are to be allocated
  then if partial track does not fit in first part
    of track beyond the last full track
    then repeat 12
  else do
    test to see if appropriate number of consecutive
      empty sectors exists on the track
    count the number of shifts
    if allocation will not fit in track
    then repeat 12
  end
  set the proper bits in the allocation word
  if the track is now full
  then set bit 0
end
construct the disk address and the allocation bit map
if any full tracks are to be allocated
then do
  set number of full tracks in bits 9-17 of
    first allocation word
  set bits 0-8 of number of full track
    allocation words
end
write the allocation table to disk
end
else not enough disk space for allocation
end ALLOCATABL.

```

CATLIST: This routine dumps the catalogue and directories to magnetic tape along with the track allocation bit map.

```

Algorithm: read the catalogue header for current number of entries
do until all entries have been processed
  if directory name non blank
  then do
    read the directory header
    output directory header to magnetic tape
    get number of data sets and entry offset
    do until all data set descriptors are read
      if data set name non blank
      then do
        if output string pointer is 1 then
          wait on tape write

```

```

        assign the descriptor to output string
        if output string pointer is 4
            then do write the output string to
                magnetic tape
            reset IOUT to 1
        end
        increment output string array pointer
    end
end
    if output pointer greater than one
        then output residual array to mag tape
    end
end
    output allocation table to mag tape
end CATLIST.

```

DATASET: The purpose of this routine is to test the data set operation codes and call the proper task for processing this code. Data set names may not be blank or begin with the @ character which is reserved for the system.

Algorithm:

```

        if directory is in the catalogue
            then do
    41:         get data set name
            if data set name is valid
                then do
                    do until all data set entries have been searched
                        if data set name is in directory
                            then set found bit
                        end
                    if data set is in directory
                        then
                            if operation is list
                                then list
                            else do
                                check for directory protection code
                                if directory is valid
                                    then process operation
                                end
                            else
                                if directory is not full
                                    then add data set
                                end
                            end
                        else repeat 41
                    end
                end
            end
        end DATASET.

```

DATA_DISK: The purpose of this routine is to check security for processing system directories. This allows protection from unauthorized users handling data sets in directories DISK and ALECS2. If the desired directory can be accessed, the directory flag is set on return; otherwise it is reset.

Algorithm:

```

    reset the directory flag
    if directory is protected
    then do
        get protection key
        if protection key is correct
        then set the directory flag
        else reset directory flag
    end
    else set the directory flag
end DATA_DISK.
```

DDR: The purpose of this routine is to delete an entire directory from disk. Each data set must be deallocated individually. After all of the data sets are deleted, then the directory is deallocated, and the name removed from the catalogue. Internal routine DDS handles the individual data set deletion.

Algorithm:

```

    do until current number of entries is 1
        read block of data set entries
        do until block is processed
            if entry is data set
            then deallocate space on disk
            if current number of entries is 1
            then exit loop
        end
    end
    deallocate the directory space
    remove the directory entry from the catalogue
```

DDS: The purpose of this internal procedure is to deallocate disk space.

Algorithm:

```

    read the data set descriptor for the allocation words
    retrieve the allocation array pointer, disk number, and
        allocation bit map
    read the allocation table
    retrieve the number of full tracks allocated
    if any full tracks are allocated
        then reset all allocation bits
    if less than 1 track is also allocated
        then reset appropriate bits in allocation word
    rewrite the updated allocation table to disk
end DDS
end DDR.
```

DDS: The purpose of this routine is to delete a data set from disk.

Algorithm:

```

    read the directory header
    do until data set to be deleted is found
        if directory entry is non blank
            then increment directory entry counter
        if directory entry counter > current number of entries
            then do
                data set not in directory                /*ERROR*/
                exit task
            end
        end
    set directory entry to blank
    if bit 3 of status word set
        then data set active, do not delete
    else do
        if data set descriptor name matches entry
            then do
                if data set is to be deallocated
                    then do
                        retrieve the allocation array pointer, disk number,
                            and allocation bit map
                        read allocation table
                        retrieve number of full tracks allocated
```

```

        if any full tracks are allocated
        then reset all allocation bits
        if less than 1 track is also allocated
        then reset appropriate bit in allocation word
        rewrite the updated allocation table to disk
        blank data set descriptor name
    end
end
else error

```

DIRECTORY: The purpose of this routine is to sort the operation code and call the appropriate task to perform this operation.

Algorithm:

```

    if op code is ADR
    then
        if REC_NO
        then directory already in catalogue                /*ERROR*/
        else if NOTCATFUL
        then allocate directory
        else catalogue is full                            /*ERROR*/
        else
            if REC_NO
            then if op code is LDR
            then list directory
            else if op code is MDR
            then if NOTCATFUL
            then modify directory
            else catalogue if full                        /*ERROR*/
            else if directory is DISK, UNIVERSAL,
            or @DS
            then illegal operation for directory          /*ERROR*/
            else if op code is RDR
            then rename directory
            else if op code is DDR
            then delete directory
            else illegal op code                          /*ERROR*/
            else directory not in catalogue                /*ERROR*/
        end DIRECTORY,
    end

```

DISKORG: This routine is the driving routine for the disk organization programs. The following operations are currently allowed: directory operations allocation, deletion, listing, modification, and renaming. Data set operations allocation, deletion, listing, renaming, and transferring.

Algorithm:

```

    set the system flag
    get creator's name and current date
    get the operation code
    if op code is LCT
        then set flag for catalogue list
    else do
        set flag for catalogue search
        get directory name
    end
    call LIST_CAT to perform the catalogue operation
    if search catalogue flag set
        then
            if directory operation code
                then process directory operation
            else process data set operation
        end
    reset the system flag
end DISKORG.
```

GETFORMAT: This routine reads the format code for input of normal data set from cards.

Algorithm:

```

    request the column format
    do until all columns in format string are processed
        do while a column is a digit
            process digit
            get next column
        end
        if number was processed
            then set repetition factor
        else repetition factor is one
    end
```

```

set last column processed counter
if column is non blank
then do
  if column is proper format code
  store input symbol
  retrieve the field width
  compute column number of input card
  if format symbol is A
  then if column number  $\neq$  72 or 80
  then do
    print error message          /*ERROR*/
    repeat format request
  end
end
else repeat format request
store counter in column array
end
end GETFORMAT.

```

LDR: The purpose of this routine is to print the contents of the directory on the teletype. This list can be printed in abbreviated form or in extended form as in the data set list.

Algorithm:

```

read the directory header
print teletype headings
get abbreviated list flag
do until all data sets are listed
  read the data set descriptor
  if data set name is non blank
  then
    if list flag is set
    then do
      print data set descriptor
      if data set is task
      then print task information
    end
  else do
    print abbreviated list
    if data set is task
    then print task information
  end
end
end LDR.

```

LDS: The purpose of this routine is to list the data set descriptor onto the teletype.

Algorithm:

```

    read the directory header
    do until all entries are processed
        if data set name matches entry
            then exit loop
        if data set name is non blank
            then increment data set counter
        if data set counter > the number of entries
            then data set not in directory          /*ERROR*/
    end
    read the data set descriptor
    retrieve the disk number and disk address
    print data set descriptor on teletype
    if data set is task
        then print task information
    end
end LDS.
```

LIST_CAT: The purpose of this routine is to search the catalogue.

If the search flag is set, then the catalogue is searched for the directory name passed in the call list, otherwise the contents of the catalogue are printed on the teletype.

Algorithm:

```

    read the catalogue header
    do until all catalogue entries have been read
        if entry in catalogue (non blank name)
            then
                if flag is set
                    then if entry matches directory name
                        then mark directory found
                    else print catalogue entry and address
                end
            calculate the number of blank entries in the catalogue
    end LIST_CAT.
```


NEWFILE: The purpose of this routine is to read cards from the card reader when loading a normal data set from cards.

Algorithm:

```

Experiment interrupt handler
  store interrupt oncode for testing
  clear the interrupt level
  if bit 7 is set
    then hopper empty /*ERROR*/
  if bit 16 is set
    then
      if 80 columns have not been read
        then card not complete /*ERROR*/
        else exit interrupt handler
      if bits 4/6, 8, or 9 are set
        then bad data /*ERROR*/
      if bit 17 is not set
        then data not ready /*ERROR*/
      read the column of the card
      extract the 12 bits of useful information
      increment the column counter
      get next number to be processed
end interrupt handler.

Initial task entry point
  11: get the format from the teletype
    if no more format cards
      then exit routine
    initialize pointers and flags
    read in card
    if bad card flag has been raised
      then do
        12: interpret error message /*ERROR*/
          repeat 11
        end
      if card done flag raised and no text on card
        then do
          13: if columns 1 and 2 are '/'
            then get next format statement
          end
          /* convert the column binary number to character */
          do until the character has been translated
            if bad card flag has been raised
              then go to 12 /*ERROR*/
            if cards are finished
              then go to 13

```

```

        if column has not been read yet
            then wait until column has been read
            search for character
            store character in string
        end
        get data type from format string
        store characters to be processed
        convert and write the proper data type to disk
        increment the number of input numbers
    end NEWFILE.

```

MDR: The purpose of this routine is to modify the maximum number of entries in a given directory. A new dummy directory is created. The contents of the old directory are copied into the new directory. The old directory is deleted and new directory renamed and placed in the catalogue. The data sets in the directory are untouched.

Algorithm:

```

    1: get the maximum number of entries
    if maximum number of entries is acceptable
        then continue
    else repeat 1
    read the directory header
    if new maximum number of entries < current number of entries
        then repeat 1
    add dummy directory to catalogue
    if dummy directory was successfully allocated
        then do
            compute data set descriptor offset value
            write directory header into dummy directory
            do until all old directory entries are processed
                if directory entry is non blank
                    then do
                        read data set descriptor from old directory
                        if data set name is @DIRECTRY
                            then reset date, creator's name, and
                                reference count
                        write data set descriptor to new directory
                    end
                end
            end
            update the current number of old directory entries
            delete old directory but do not deallocate data sets
            rename catalogue entry for new directory
        end
    end MDR.

```

RDR: The purpose of this routine is to rename a directory.

Algorithms:

```

    read the directory header
    if directory name agrees with directory entry
    then do
    11:   get new directory entry
        if directory name is illegal
        then repeat 11
        read catalogue entry
        search the catalogue for new directory name
        if new directory name in catalogue
        then directory already in catalogue      /*ERROR*/
        enter the new directory name into the catalogue
    end
end RDR.
```

RDS: The purpose of this routine is to rename a data set.

Algorithm:

```

    get new data set name
    read directory header
    do until all directory entries are searched
    if directory entry is data set name
    then mark directory entry
    else if entry is new data set name
    then data set already in directory      /*ERROR*/
    end
    if old data set was found
    then do
    read data set entry
    read the data set descriptor
    if bit 3 in status word is set
    then data set is active
    else enter new directory name into entry and
    data set descriptor
    end
end RDS.
```

SETUPDSET: The purpose of this routine is to set up a data set entry in a directory.

Algorithm:

```

    read the directory header
    if directory is not full
        then do
            do until all entries are searched
                if empty is entry
                    then if first empty entry
                        then mark entry pointers
                    else
                        if entry matches data set name
                            then data set already in use          /*ERROR*/
                end
            if data set is norm or experiment
                then do
                    compute number of tracks and excess sectors
                    if excess sectors across track boundary
                        then construct allocation bit map
                    allocate disk space
                    if space was allocated
                        then do
                            enter new data set into the directory
                        end
                    end
                end
            end
        end SETUPDSET.

```

STATUSSET: The purpose of this routine is to set the status word in the data set descriptor for a newly allocated task or experiment data set.

Algorithm:

```

read the directory header
do until all entries are searched
  if entry matches data set then exit loop
  if entry is not blank
    then do
      increment data set counter
      if data set counter > current number of entries
        then do
          data set not found          /*ERROR*/
          exit routine
        end
      end
    end
  end
locate data set descriptor
if data set descriptor entry does not match data set name
  then exit routine                  /*ERROR*/
if data set type is experiment
  then status is 100000
  else status is 400000
write the status word in the data set descriptor

```

TASKLOAD: This routine loads a task from the CR15 card reader.

Additional leading comments: The card reader is treated as an experiment file. After each column is read, an interrupt is received from the card reader. The oncode is interrogated to interpret the cause of the interrupt. The following bits may be set:

```

bit 4 - photo error
bit 5 - motion error
bit 6 - pick error
bit 7 - hopper empty error
bit 8 - data missed
bit 9 - trouble
bit 16 - card complete.

```

The card deck to be read begins with a name card which is identified by @NM in columns 1-3. This card is followed by the text cards which are identified by @TX in columns 1-3. These cards contain the object code punched in column binary and stored in columns 4-72.

Since the PDP-15 instruction is 18 bits, 3 columns make up 2 PDP-15 words. These words must be unpacked from the columns as they are read from the card.

Following the text cards are the relocation word cards which are identified by @RL in columns 1-3. These cards contain PDP-15 words with a bit set for relocation of the address when the task is loaded into core. There is 1 word for each 18 instructions in the object code. Finally a card with @EN in columns 1-3 signifies the end of the object deck.

The function EXEC_CRCS is a built in function for the clear status register and data buffer instruction 706704 for the CR03B card reader. Bits 13, 15, and 17 are set in the accumulator to initiate the reader when the interrupt level is cleared. Bit 17 causes a program interrupt to enable the interrupt handler to start processing the card. After the initial start, only bits 13 and 15 need to be set in the accumulator when the status is cleared. The card is then read 1 column at a time until the entire card has been read. An interrupt is received after each column has been read.

Algorithm:

```

Experiment interrupt handler
if bits 4, 5, 6, 7, 8, or 9 set
  then do
    el:    set all events complete          /*ERROR*/
           set error message
           cxlt handler
  end
if bit 16 is set
  then do
    if column count < 80
      then do
        columns missing on card
        go to el
      end
    end
  end

```

```

if bit 17 is set
then do
  read the column
  set 6 low order bits to zero
  increment the column count
  if column count < 4
  then if column count is 3
  then do
    save contents of column 3
    set name event complete
  end
  else if column count is 2
  then save contents of column 2
  else save contents of column 1
  else
    if column count < 73
    then do
      store data in column array
      increment array pointer
    end
    else
      if column count > 77
      then do
        translate punch into numeric value
          by shifting until reaches bit 9.
        Double value for each shift.
        This is value N
        store sequence number
        (SEQ=SEQ+10+N)
        if column count is 80
        then set sequence event complete
      end
    end
  end
end interrupt handler

```

Internal Procedure SETUPREAD: This routine initializes variables and reads a card.

```

status=0
do until bit 13 of status is set
  read the card reader status
end
reset the card complete flag and events
read the first column of the card
end SETUPREAD.

```

Internal Procedure PACK: This routine packs 12 bit column binary information from a card into 18 bit words.

```

    set 18-bit word array pointer to 1
    do until the card array is completely processed
        wait for column to be read
        if first column is set of 3
            then do
                shift bits 6 places left
                store in 18-bit array location 0 module 2
            end
        else
            if second column in set of 3
                then do
                    shift bits 6 places right
                    merge into array location 0 module 2
                    increment 18-bit array pointer
                    shift bits 12 places left
                    store in word 1 module 2
                end
            else do
                merge word into word 1 module 2
                increment 18-bit array pointer
            end
        end
    end PACK.

```

Internal Procedure DOIT: This routine processes text and relocation cards.

Algorithm:

```

    pack the 12 bit words into 18 bit words
    if process set
        then do
            compute amount of disk space for write
            wait for card to complete
            if column counter < 42
                then write disk from one word
            else write card contents to disk
        end
        wait for card to complete
        if card counter does not agree with sequence number
            then card out of order

```

/*ERROR*/


```

do until card has completed
  if error flag raised then card error      /*ERROR*/
end
read next card
increment card counter
wait for name to be read
end DOIT.

```

Main Routine Algorithm:

```

initiate the card read
wait for name (columns 1-3) to be read
if column number not 34
  then do
    wait for card to be read
    if error flag raised
      then do
        print error message
        if return flag set
          then set error flag to return flag
          else set error flag to '1'
        end
      end
    if card_done flag set
      then start card read
      else restart the loop
    end
  else do
    if columns 2, 3 not NM
      then if 2, 3 not TX and 2, 3 not RL
        then skip card
        else name card not found      /*ERROR*/
      else do;
        pack first nine columns of name card
        wait for end of card
        if sequence number not 1
          then name card not first card      /*ERROR*/
        increment card counter
        allocate space and set up DSD entry of task
        if additional sectors beyond full track needed
          then do
            complete track allocation bit map
            assign allocation word 2-bit map and
              number of full tracks
          end
        end
      end
    end
  end
end

```

```

        allocate the disk space
        if disk space was allocated
            then do
                set up task
                if error then card error;          /*ERROR*/
            end
            else card error;          /*ERROR*/
        end
        open allocated disk file to write object code
        read the next card
        increment card counter
    41:  if columns 1-3 not @TX
        then text card not found          /*ERROR*/
        else do
            process text and relocate records
            if all text cards not read
                then repeat 41
            else do
                if process is set
                    then align file on 8 word boundary
                skip over extra text cards
                set process
                if mismatched text length
                    then card error          /*ERROR*/
                do until all relocate cards are processed
                    if columns 1-3 not @RL
                        then relocation card not found          /*ERROR*/
                    process text and relocate records
                end
                if columns 1-3 not @EN
                    then end card not found          /*ERROR*/
                if relocation length mismatched
                    then card error          /*ERROR*/
                wait until card is finished
                if card counter does not agree with number
                    then card out of sequence          /*ERROR*/
            end
        end
    end TASKLOAD.

```

TASKSETUP: This routine sets up a task data set when allocated
from DISKORG.

Algorithm:

```

compute number of relocation words needed-(the coresize
  of the task + 17)/18 bits per word; bit is set if
  address is to be relocated, else reset
compute the relocation offset-[(task length + 7)/8]*8
  task length module 8. Address is the first 8 word
  boundary beyond the task disk storage
compute the number of sectors required-relocation
  offset + number of relocation words
compute number of full tracks-8 sectors/track
compute excess sectors-number of sectors-8* number
  of tracks
if excess sectors are computed
  then build the allocation bit map for partially
    filled track
  left justify the allocation bit map
  assign allocation word 2 the allocation bit map
    + the number of full tracks
open the directory
do until all data set blocks are read
  do until all data sets are processed in block
    if entry in directory matches data set name
      then exit outer loop
    if entry in directory is non blank
      then update entry counter
    if entry counter > number of entries
      then exit routine
  end
end
compute key for data set descriptor
read the data set descriptor
if data set descriptor name does not match data set name
  then exit routine
fill the data set descriptor with appropriate values
update the data set descriptor
end TASKSETUP.

```

TDS: The purpose of this routine is to transfer a data set from one directory to another directory.

Algorithm:

```

read the directory entry
do until all entries have been searched
  if name matches entry
    then exit loop
  if name is non blank
    then do
      increment data set counter
      if data set counter > the current number of entries
        then data set not in directory
      end
    end
  end
get new directory name
if new directory matches old directory
  search catalogue for new directory
  if new directory is not in catalogue
    then do
      get new data set name
      read new directory header
      if space remains in new directory
        then do until new directories have been searched
          if entry matches new name
            then data set is in use
          if data set is non blank
            then increment data set counter
          else if first blank entry
            then mark entry pointers
          if data set counter > current number of entries
            then exit loop
          end
        set up new data set descriptor
        write new directory
        remove old directory entry
      end
    end
  end
end TDS.

```

V. Acknowledgements

This work was supported by the U. S. Department of Energy,
Division of Basic Energy Sciences.

The authors wish to thank W. Lembke for her expert typing
and valuable assistance in preparing this manual for printing.

VI. Appendices

The following appendices consist of tables for the catalogue, directories, and data set descriptors. Remarks on the system libraries and the allocation and its allocation words are also included.

A. System Libraries

There are two "system" libraries. One is UNIVERSAL and the other is @DS.

UNIVERSAL is the default library. When a data set or task (i.e., loadable data set) is not found in the user library then UNIVERSAL is searched for it. UNIVERSAL contains those data sets and utility programs which are available to all users. UNIVERSAL is searched immediately if the user library name indicated at task tree creation (REQ or INQ) is blank.

@DS is the library of system data sets. @DS can be accessed for other than INPUT by those tasks with system access privileges and no others. @DS contains the following data sets (as well as @DIRECTRY):

<u>Name</u>	<u>Record Length</u>	<u>Contents</u>
@CATALOG	40 (32)	the catalog
@ALLOC	400 (256)	the disk allocation table
@TAAREA	400 (256)	the transient area library
@IPL	undefined	The code to be loaded to start up the system. This code is loaded by a bootstrap tape. (Not implemented at present.)

B.. Catalog Format

<u>Offset</u>	<u>Contents</u>	<u>Words</u>
0	current number of non null entries	1
1	maximum number of entries	1
2	date of creation	2
4	allocation pointers	2
6	reserved for future use	2
10	first entry	4
14	second entry	4
20	third entry	4
	.	
	.	
	.	

Each entry has the form:

<u>Offset</u>	<u>Contents</u>	<u>Words</u>
0	library name	3
3	DOA of Directory	1

- Notes:
1. The catalog is a data set (see @DS section) and begins on a 400 (256) word boundary.
 2. The current number of entries indicates how many entries there are, not how far to search. The entire catalog must be searched.

C. Directory Format

<u>Offset</u>	<u>Contents</u>	<u>Words</u>
0	current number of non null entries	1
1	maximum number of entries	1
2	library name	3
5	date of creation	2
7	creator's name	3
12	allocation pointers	2
14	DSD offset in number of 40 word block	1
15	reserved for future use	11
30	series of three word name entries . . .	
? (mod 40)	first DSD entry	40
?+40	second DSD entry . . .	40

- Notes:
1. The three word entries beginning at offset 30 are the data set names. The n^{th} DSD associated with the n^{th} name is found by using $n*40$ as an offset from the beginning of the DSD's. The three word entries are in the directory this way to facilitate searching.
 2. The symbol ? represents (word 14) *40.
 3. As with the catalog the current number of entries indicates how many entries, not how far to search.
 4. Each directory is aligned on a 400 (256) word boundary and is between 1 and 8 sectors in length. The maximum number of data sets allowed in a directory for each of the 8 sizes is (in decimal): 5,12,20,27,34,42,49,56 plus one for the directory itself (@DIRECTRY).

D. Data Set Descriptor (DSD)

<u>Offset</u>	<u>Contents</u>	<u>Words</u>
0	data set name (level two name)	3
3	date of creation	2
5	creator's name	3
10	date of last reference to data set	2
12	reference count	1
13	data set status word	1
14	the number of formal parameters of the task*	1
15	DOA of first relocation word within the data set*	1
16	number of relocation words*	1
17	size in words of task in storage*	1
20	length in words of task code on disk*	1
21	record length in words	1
22	EOF address in full DWA form	2
24	allocation pointers (beginning data set ADDR bit pattern)	2
26	DOA of end data set	1
27	reserved for future use	11

*Meaningful only if the data set is loadable data set.

Notes: 1. The date of last reference is filled in and the reference count is incremented when the file associated with the data set is closed (not implemented yet).

2. The data set status word has the form:

bit 0: 1 loadable data set, 0 normal data set
 bit 1: 1 EOF "written", 0 EOF not "written"

D. Data Set Descriptor

bit 2: 1 an experiment data set, 0 normal data set.
bit 3-17: unused.

3. The EOF address is the full DWA of the last word written if the data set has been opened for OUTPUT or the full DWA of the last word of the data set has been opened for UPDATE. The EOF address is initialized when the data set is created to point to the first word of the data set.
4. The DOA of the end of the data set points to the last eight word block in the data set.
5. If a data set is a loadable data set then its record length is 400 (256) and the EOF address is set to the end of the data set.

E. Disk Allocation Table

The Disk Allocation Table is a system data set (@DS.@ALLOC) that describes the allocated or unallocated status of the disks. Each of the 4000 (2048) word tracks on the disks are divided into arbitrary 400 (256) word sectors that are each described in the Disk Allocation Table. Thus there are 8 sectors on each track. DISKORG will allocate space on the disk in terms of the number of 400 word sectors that are needed.

Within the Disk Allocation Table, one word will be used to describe the track. The rightmost 9 bits of the word will be zero while the leftmost 9 bits of the word will be used to describe the track and the eight-400 word sectors on that track. Schematically,

$$\begin{array}{r} \text{a } \text{LLLLLLLL} \text{ 00000000} \\ \hline 0 \text{ 1} \quad \quad 8 \text{ 9} \quad \quad 17_{10} \end{array}$$

"0" bits are ignored

"a" bit describes the track as a whole

0 -- track is entirely free or partially seized

1 -- track is entirely seized

"b" bits describe each of the 8 sectors in order, forming a "mask" of the track

0 -- the corresponding 400 word sector is free

1 -- the corresponding 400 word sector is seized

When an amount of disk is requested, the amount of disk allocated will be determined (space is allocated in terms of the number of 400 word sectors that are needed), and three items of information will be saved for deallocation purposes. The information will be saved in two words.

Word 1: The DOA of the disk, track, and sector of the first sector seized by allocation of this data set.

Word 2: Sector mask and the count of the number of complete tracks seized for this allocation. If 8 or more sectors are allocated, they will be track-aligned, and the count field will contain the number of tracks entirely seized, with the sector mask marking the sectors seized on the next track beyond the last full track. The sector mask is bits 1-8 of the allocation table word for the track. If less than 8 sectors are allocated, then the sector mask marks the sectors seized on the one track and the count field is zero. An allocation of 8n sectors is represented by a count field containing the number n and a sector mask of all zeros.

Schematically,

Word 1: ddd ttt ttt tss sss sss
disk track "sector"

number number number

Word 2: mmm mmm mmc ccc ccc ccc
sector count field

mask

F. Diagram of Disk Organization

.CAT
in
VARCON

@CATALOG

@DIRECTRY

UNIVERSAL

catalog

default
library
directory

other
data
sets

other
directories

@DIRECTRY

@DS

system
directory

@ALLOC

other
data
sets Allocation
table

G. DISKORG Task Tree

Task Tree A

Task Tree B