IRIS 7.3

Interactive Real-time Information System

MANAGER MANUAL

POINT 4

DATA CORPORATION

EDUCATIONAL DATA SYSTEMS

1682 Langley Avenue, Irvine, California 92714

(714) 556-4242

Interactive Real-time
Information System

IRIS 7.3

MANAGER

REFERENCE MANUAL

This manual is intended for use by the System Manager and other persons requiring information about operating and extending an IRIS R7.3 system. Included are detailed procedures for system generation, system modifications and updates, system start up and shut down, maintenance of the users' accounts, use of the system in stand-alone mode, writing a new processor, subroutine, or peripheral driver, system commands and functions accessible only to the manager, interpretation of trap messages, and other useful information. It is assumed that the reader is already familiar with IRIS. Refer to the IRIS User Reference Manual for log-on and log-off procedures, use of system commands, and use of standard language processors. Refer also to the Glossary in Appendix 3 of this manual for definitions of terms used herein.

DISCLAIMER: Every attempt has been made to make this reference manual complete, accurate, and up to date. However, there is no warranty, either express or implied, as to the accuracy of the information contained herein or to its suitability for any purpose. This manual is offered only subject to this disclaimer.

All Rights Reserved
Copyright © 1974, Educational Data Systems
This document contains secret and confidential
information of Educational Data Systems. It may
not be reproduced, used, or disclosed without the
prior written permission of Educational Data Systems

Copyright 1978
Educational Data Systems

EDS 1018-11
5 SEP 78

TABLE OF CONTENTS -- IRIS Manager Reference Manual

page section

Copyright 1978
Educational Data Systems

page section

FIGURES AND APPENDICES

Copyright 1978
Educational Data Systems

1. INTRODUCTION TO IRIS

IRIS is an Interactive Real-time Information System designed to support real time data acquisition and process control, data communications, interactive time-sharing, and background data processing simultaneously. To be practical for use in such a variety of applications a system must be modular and open-ended; that is, it must be easy to configure a system using only the necessary modules such as peripheral drivers, task processors, etc., and it must also be easy to add new drivers, tasks, etc. at any time. IRIS was designed to meet these goals, and this manual is intended to supply all of the information required by the manager of an IRIS system.

1.1 Components of IRIS

The IRIS environment consists of several disc files as follows:

BZUP The Block Zero Utility Package, which resides in block zero of the disc (each Logical Unit), is brought into core from the system disc by the Initial Program Load (IPL) bootstrap. BZUP may be used for debugging purposes, or the IPL sequence may be allowed to continue, in which case the REX disc file is brought into core and initialized (see Section 1.2). Strictly speaking, BZUP is not actually a file, since it has no header and does not appear in the INDEX.

REX The REX file contains the remainder of the IPL routine (in the file's header), the Real-time Executive (REX) which occupies approximately the first 4K words of core (excepting locations 200 through 577 octal), the System Initializing Routine (SIR) which is executed once after the IPL, and DBUG which may be used for troubleshooting SIR and REX.

INDEX The INDEX contains the Filename and Real Disc Address of each file. Each Logical Unit has its own INDEX, which is hash-addressed for fast file lookup. See Section 1.11 for details.

DMAP The Disc Map indicates which disc blocks are in use and which are available for creating new files or expanding old ones. Each Logical Unit has its own DMAP.

DISCSUBS A number of subroutines that are a requisite part of the operating system but may be used too infrequently to be kept in core are stored in this file of Disc-Resident Subroutines. Many of these subroutines are also used by the various processors. The system manager may specify that certain of these subroutines are to become core-resident at next IPL time by setting flags in the CONFIG file (see Section 2.9).

ACCOUNTS This file contains the Account ID, assigned priority, assigned Logical Unit, privilege level, account number (group and user), CPU and connect time allotments, disc block allotment and usage information, and accumulated net charges for each user's account (see Section 7).

CONFIG This file contains all information about the current configuration of the system, a system disc driver for each known type of disc controller, and a disc driver for BZUP for each type of disc controller. The system manager may modify this file and then do an IPL to change the system configuration (see Section 2.4). All other components of IRIS are configuration independent except for actual peripheral device driver files.

The following processors are also required in the minimum IRIS environment:

SCOPE The System Command Processor analyzes all system commands and provides the means for a user to transfer control of his port from one processor to another.

BYE This is the Log-on/Log-off processor which keeps track of the user's CPU time and connect time usage and updates the user's entry in the ACCOUNTS file accordingly.

DSP The Disc Service Processor is used for debugging and updating the system or any file. DSP may be used from any active port by the system manager while the system is in normal use.

PLOAD The Program Loader is used to load new files from paper tape to update or extend the system. PLOAD is contained in the SYSGEN binary tape along with SIR (the System Initializing Routine) and SYSL (the SYStem Loader).

The following processors are also required if the system is to use more than one Logical Unit:

INSTALL The Logical Unit Installation processor is used to bring up each Logical Unit other than the system disc and when installing a disc pack on any changeable cartridge disc drive.

REMOVE The Logical Unit Removing processor is used when removing a disc pack from a changeable cartridge drive.

Other processors, such as BASIC, RUN, SAVE, KILL, COPY, and LIBR, are optional components of the complete IRIS environment. They are not required for operation of the system except when their specific functions are required.

1.2 Initial Program Load Sequence

Initial Program Load (IPL) must be performed after a crash or after using the system in stand-alone mode. IPL brings a fresh copy of REX into core from the disc, and the System Initializing Routine (SIR), which is included in the REX disc file, performs all required initializing functions.

The first step of an IPL is to get BZUP from block zero of the system disc into core page zero. Refer to Section 5 for the various methods of starting an IPL. Also refer to the sections on BZUP and DBUG if the IPL sequence is to be interrupted for use of a debugging package. Each bootstrap program initiates this one-block transfer and then idles at location 377.

Location 377 is finally overlayed by a JMP instruction in the last word of BZUP. This transfers control to a routine in BZUP which copies BZUP to the 400 words (octal) starting at location LBZUP (defined in the Software Definitions). LBZUP is currently 24000 octal. If switch zero is up, control is then transferred to BZUP's control mode. If switch zero is down, then the REX header is loaded into core, and control is passed to the IPL routine in the REX header.

IPL reads the remainder of the REX file into core by use of the disc driver in BZUP. If the switches are set to the starting location of DBUG (currently 25000 octal) control is then given to DBUG; otherwise, control goes to SIR. Control may be passed from DBUG to SIR by executing a jump to location LSIR (defined in the Software Definitions). LSIR is currently 14000 octal. SIR examines the CONFIG file to bring the necessary disc drivers into core, sets up LUFIX and LUVAR tables for each disc, scans the DISCSUBS file to set up the Disc Address Table (DAT) and the Starting Address Table (SAT) with the Real Disc Address and the actual in-core starting address, respectively, of each disc-resident subroutine, brings selected DISCSUBS into core, reserves six disc blocks for use in saving SSA when nesting subroutines, locates the SCOPE, DSP, DISCSUBS, MESSAGES, and BYE files and puts their disc addresses into the INFO table, sets up each port's Port Control Block (PCB), Data File Table (DFT), and Input/Output buffer, scans the INDEX to create a fresh copy of the Disc Map (DMAP), creates an active file for each port, and requests a type-in of the date and time. Control is then transferred to the START routine in REX where the interrupt system is initialized, and finally to the idle task.

Note: if switch one is up at the time control is transferred to SIR then the system will be initialized for minimum configuration (master terminal only, no peripherals, only 16K of core, no core-resident DISCSUBS, etc.). This allows use of DSP in case of a problem such as core overflow, a bug in a peripheral driver, etc. SIR will also do a new IPL and retry initializing for the minimum configuration following any fault condition; see Appendix 7 for possible trap numbers and their meanings.

1.3 Disc and Core Usage

Figure 1.1 is a map depicting the use of core memory by IRIS. The first 128 words are occupied by various pointers and constants used by REX. The decimal accumulator (DA), which is used for all decimal arithmetic and input/output is also in this area. The next 256 words (one disc block) are part of the regnant processor. Next comes the system information table. The Real-time Executive (REX) occupies approximately the next 3K words, following which is patch space up to the defined Beginning of Processor Storage (BPS). The various disc block buffer areas begin near the top of lower core. The processor may use all of the space from 200 through 577, from BPS through MBUS-1, and one user partition.

The Port Control Area (PCA) may be forced into a specific location by hardware restrictions; therefore, there may be space left on each side of it. SIR then allocates space in the starred areas of the core map for Data File Tables, I/O buffers, stacks and tables used by REX, peripheral drivers, core-resident discsubs, task nodes, etc. If there is more than 32K words of core then upper core (ie, all core above 077777 octal) will be used only for additional user partitions (see Sections 2.4 and 2.8).

Each Logical Unit has a copy of BZUP in Real Disc Address zero, an INDEX whose header is in Real Disc Address one, an ACCOUNTS file whose header is in Real Disc Address three, and a DMAP (disc block usage map) whose header is in cylinder zero, track one, sector zero (Real Disc Address LRT). Figure 1.2 shows the structure of the disc map. The system disc (Logical Unit zero) also has the REX (Real-time Executive) file whose header is at Real Disc Address two, and a set of discsub nesting blocks and the DISCSUBS file which immediately follow DMAP. These files must be forced into these specific locations so that they can be found without looking through the INDEX. Because of this it is necessary that tracks zero and one of each Logical Unit do not cause hard data errors. Also, since DMAP and DISCSUBS are forced into successive blocks on Logical Unit zero, there must be enough good tracks on the system disc to hold these files without errors. Any other blocks on any Logical Unit may be marked as "bad" to prevent their use by any IRIS file.

Copyright 1978
Educational Data Systems

Figure 1.1: Map of Core

The diagram below shows how core is used by IRIS.

Location	Contents	Remarks
0	REX page zero	Pointers, constants, etc.
200	Processor page zero	This block is part of the processor file.
600	INFO	System information table.
	REX	Core-resident portion of the Real-time Executive.
PATSP	Patch space *	See note below.
ENDPS	Disc driver	Driver for system disc.
BPS	Processor area	This area and locations 200-577 are occupied by one processor such as BASIC, SAVE, LIBR, etc.
MBUS	Partition #0	Used by processor for user's program and data.
BSA	Block Swap Area	256-word disc block buffers.
HBA	Header Block Area	
HXA	Header Extender Area	
SSA	Subroutine Swap Area	
ABA	Auxiliary Buffer Area	Used for indexed files.
	*	See note below.
PCA	Port Control Area	40 words (octal) per PCB.
	*	* Note: these areas are used by SIR for user partitions, Data File Tables, I/O buffers, drivers, etc. (see Section 1.3).
TOPW		

Figure 1.2: Structure of Disc Map (DMAP)

```
word #
           -----------------------
   0      ! Cylinder address      ! \
          !-----------------------!  \
   1      ! Blocks available      !   )  There are WCM words
          !-----------------------!   )  per cylinder entry in
   2      ! Track 0 map word      !   )  the Disc Map, where
          !-----------------------!   )  WCM (Words per Cylinder
   3      ! Track 1 map word      !   )  in Map) is two plus the
          !-----------------------!   )  number of tracks per
   4      ! Track 2 map word      !   )  cylinder (see below).
          !-----------------------!  /
   5      ! Track 3 map word      ! /
           -----------------------    Real Disc Address of first
   6      ! Cylinder address      ! <--  block in this cylinder.
          !-----------------------!
   7      ! Blocks available      ! <--  Total number of blocks
          !-----------------------!   available (not currently
  10      ! Track 0 map word      !   in use) in this cylinder.
          !-----------------------!
  11      ! Track 1 map word      ! <--  Each available block is
          !-----------------------!   indicated by a "0" in the
  12      ! Track 2 map word      !   corresponding bit in the
          !-----------------------!   map word for its track.
  13      ! Track 3 map word      !   A "1" in a map word means
           -----------------------    that the corresponding
   .      !    .                  !   block is not available.
          !                       !
   .      !    .                  !
          !                       !
   .      !    .                  !
```

The least significant bit of a word is bit number zero, and it maps sector zero of the track mapped by each map word. Therefore, for a disc with twelve sectors, the map word for an empty track would be 170000 octal, indicating that sectors zero through eleven are available, and sectors twelve through fifteen are not available (non-existent in this case).

The above figure represents the map for the first two cylinders of a drive having four disc surfaces (four tracks per cylinder), and the value of NT (number of tracks) in the LUFIX table would also be four.

The DSPS cells and the FMAP cells (currently locations 50 through 170 octal) of the DMAP file header are used for the "bad blocks" list, which is terminated by a zero word. Up to eighty Real Disc Addresses may be listed as bad on each Logical Unit.

1.4 Disc File Structures

IRIS provides facilities for two structurally different forms of disc files; random and contiguous. A file of either form consists of a header and a number of data blocks. The basic difference between the two forms is that a random file's header contains the Real Disc Address of each data block (or of each header extender block if the file is extended) while the contiguous file's header contains only the header's disc address and a value indicating the total number of blocks in the file. The differences are discussed in detail in Sections 1.5 and 1.6. The remainder of this section will describe only the characteristics of a disc file that are common to both forms.

Each file's header contains its Filename, all of the file's attributes, and information regarding the location of all of the file's data blocks. The displacement symbol and its currently assigned value (in octal) are given in the IRIS Software Definitions (DEFS) along with a brief description of each attribute; more detailed descriptions follow. Bit 15 is the most significant bit, and all values are carried in binary except as noted.

NAME — The Filename is a string of up to fourteen ASCII characters, not including the Logical Unit number. As with all strings in IRIS, the top bit of each ASCII code is unconditionally set to one.

ACNT — This account number word is divided into three fields:

bits 15,14	Privilege level
bits 13-6	Account group number
bits 5-0	Account user number

TYPE — The bits in this word are used as follows:

```
bit 15   (not used)
bit 14   Read protected      \ Against users
bit 13   Write protected      )  of any lower
bit 12   Copy protected      /     privilege.
bit 11   Read protected      \ Against users
bit 10   Write protected      )  of the same
bit  9   Copy protected      /     privilege.
bit  8   Runnable processor
bit  7   Load active file when selected
bit  6   Initiate input before first swap-in
bit  5   May be locked in core
bits 4-0 contain the file's type (see Section 8.7)
```

NBLK — The total number of disc blocks currently allocated to the file, including the header.

STAT — Each bit of this file status word is a flag with a specific meaning as follows:

bit 15	File is being built, not yet closed
bit 14	A file is being built to replace this one
bit 13	File is to be deleted when no longer open
bit 12	File is mapped (formatted data file)
bit 11	File has been opened with an OPENLOCK
bit 10	File is not deleteable
bit 0	File is extended

Bits nine through one are not currently in use. A file that is being built and is locked (bits 15 and 11 both set) cannot be closed; an attempt to CLOSE the channel will CLEAR the channel and delete the file.

NITM — In a formatted data file (bit 12 of STAT is set) this word specifies the number of items in each record.

LRCD — In any data file this word specifies the length (number of words) of each record.

NRPB — This Number of Records Per Block has meaning only for a formatted data file. In all other files, including contiguous data files, this word must be zero.

NRCD — This is the total number of records contained in a contiguous data file, or the number of records through the last one currently written (including lower number records not yet written) in a formatted data file.

COST — This is the amount that will be charged to other users who access (open) this file. It is carried as an unsigned decimal (BCD) integer which indicates a multiple of ten cents, thus allowing $999.90 as the maximum cost.

CHGS — This is the accumulated amount that has been charged to other users for access to this file. It is carried as a two-word floating-point decimal number, thus allowing charges to accumulate to $99,999.90 before the least significant digit of the cost is ignored due to truncation of the charges to six significant digits.

LDAT — The two-word last access date is copied from the system clock each time the file is opened by any user. The first word represents hours since 1 January 1976, and the second word represents the remaining part of an hour in tenths of a second. Each word is a binary integer.

CDAT — The file's creation date is in the same form as LDAT, but it is set from the system clock only once when the file is initially built.

NTAC This Number of Times Accessed Counter is incremented each time any user opens this file for any purpose. In the active file header, this cell is used by the timesharing algorithm to save the program value.

CATR If the file is cataloged then this cell will hold the number of the record, in a file whose Filename is CATALOG, which contains the catalog entry for this file.

CLAS These two words are used to classify a cataloged file into up to four classifications. One byte is used for each classification specified, allowing up to 256 classes to be defined.

PPRI A program's assigned priority is used by the timesharing and partition assignment algorithms. A median value of 200 is assumed until a new value is assigned by the system manager in the range zero to 377 octal.

SNUM This word holds the Software Change Order (SCO) Number of the last SCO applied to this file.

ADAT This word holds the date (hours after 1-1-76) that the last SCO was applied to this file.

DASA This eight word Decimal Accumulator Save Area is used by AFSETUP to save the six DA cells at swap-out so that they can be restored by LOADUSER at the next swap-in. The last two words of DASA hold the two save area designator words at the end of AFSETUP's pointer list (see Sections 9.4 and 9.5).

DSPS These sixteen words are reserved in each file's header for use only by the Disc Service Processor and other system routines. DSP also uses the NITM, LRCD, NRPB, NRCD, and FMAP cells in the active file header.

FMAP These 65 words are used only in a formatted data file (see Section 1.5) or an indexed contiguous data file (see Section 1.7). The FMAP cells in an active file header may be used for temporary storage by a processor since the active file cannot be formatted (see DASA above). The FMAP cells in a contiguous file header are used for directory information if the file is indexed (see Section 1.7).

HTEM This word is reserved for temporary storage by the allocate, deallocate, and account lookup system subroutines.

STAD For a machine code (stand-alone or executable) file, this word indicates the program's starting address, or a zero value indicates that no starting address has been specified. In a peripheral device driver file, this word will be set by SIR to the actual core address of the initializing routine's entry point after the driver has been brought into core. In a system driver file, this word will be set by SIR to indicate the routine's actual core location as a debugging convenience. The STAD word is not used for other types of files.

ABLK This Number of Active Blocks cell indicates to Read/Write File how many blocks are to be transferred; ie, the first ABLK disc addresses starting at location 200 octal in the file header are active blocks. Used only for reading a processor into core or for swapping an active file. ABLK must not exceed NBLK-1.

DREP If another file is being built on the same Logical Unit and with the same Filename to replace this file, then this cell will contain the Real Disc Address of the replacing file's header.

DSAF This Default Size of Active File cell is used only in the active file header to hold the size of active file (number of blocks) specified in the port driver's attributes table. This number of blocks will be allocated to the active file initially at IPL time, and the active file will be restored to this size by BYE each time a user logs off.

CORA This is the core address of the first data block, and all other data blocks start at 400 word (octal) increments from the first. If an entire block of core addresses is unused then there will be no disc block allocated, and the corresponding cell in the disc address list (starting at 200 octal) will be zero. CORA will always be zero for a text file or any contiguous file.

UNIT The number of the Logical Unit where this file resides.

DHDR The Real Disc Address of the file's header (on the specified Logical Unit).

Disc Address List -- Cells 200 through 377 contain the Real Disc Address (on the Logical Unit specified by UNIT) of each data block in the file unless this is an extended or a contiguous file. In the case of an extended file, this disc address list points not to data blocks but the header extender blocks, each of which contains up to 256 Real Disc Addresses of data blocks. The first address in this list points to the extender for the first 256 data blocks, etc. A contiguous file has no disc address list; all NBLK-1 data blocks are at sequential disc addresses immediately following the header.

1.5 Formatted Data Files

Any file that is mapped (see STAT and FMAP in Section 1.4) can be accessed as a formatted data file provided it is not protected against the caller. Each record in a formatted data file has the same format as specified by the format map. Each word in the map specifies the format and displacement into the record of the respective item in the record (word zero of the map for item zero, word one of the map for item one, etc.). The top seven bits of each word indicate the item type according to this table:

000	end of map
001	
002	
003	
004	floating point binary
005	decimal (BCD)
006	
007	
010	
011	ASCII string
012	unsigned binary
013	
014	
.	
.	
.	
077	file mark

Types not shown are not currently defined. See Section 1.10 for more information on number types. The lower nine bits of each word indicate the item's displacement; ie, the number of words from the beginning of the record to the beginning of the item. The size of each item is determined as the difference between its displacement and the following item's displacement. Therefore, the map is terminated by an "end of map" dummy item to determine the size of the last item.

A formatted file may be either a non-extended random file (requiring two disc transfers per access) or an extended random file (requiring three disc transfers per access) but may not be a contiguous file. The system's READ ITEM and WRITE ITEM routines use the format map to locate the item addressed by the caller and to check for the correct item type. However, since it is a random file, only the blocks into which data are actually written will be allocated to a formatted file, and a "record not written" error will occur if an attempt is made to read from a block that has not been allocated.

1.6 Contiguous Data Files

A contiguous file consists of only a header and the data blocks. There is no format map; the only "format" is the record length. Since there is no disc address list in the header the entire file must be allocated when it is first built. No holes are allowed in the file (all blocks must be allocated whether in use or not), and the file cannot be expanded at a later time except by building a new larger file and copying the data. A contiguous file can be built only if there are enough logically sequential blocks on the Logical Unit. When the file is built or opened, its header address, number of records, and record length (number of words) are saved in the channel in the port's Data File Table. When the file is accessed, this information is sufficient to calculate the disc address of the desired data block and the displacement into that block for the data transfer to begin. Therefore, if the Read/Write Contiguous subroutine and Get Record subroutines (READC and GETRR) are core-resident (see Section 2.9), then the very first disc access will be the desired data block. Is up to the caller (a processor or an application program) to determine item locations and types within each record.

1.7 Indexed Data Files

An indexed file is a contiguous data file in which a number of data blocks and the header's FMAP cells have been used for file directories. Each directory of an indexed file consists of three levels: master, coarse, and fine. The master level is always one disc block in length. The sizes of the coarse and fine levels depend of the maximum number of data records in the file. Two words of each block of each directory are used by the system, and each key has a one word pointer associated with it. The number of keys per block is therefore the integer value of 254/(L+1), where L is the length of the key in words. The string supplied for each key may be up to 2L bytes long.

Associated with each key in the master level is the disc address of the corresponding block in the coarse level, and associated with each key in the coarse level is the disc address of the corresponding block in the fine level. Each key in the master or coarse level is equal in value to the last (highest value) key in the block to which it points in the next level. The blocks within each level may be in random order within the area reserved for that level, but the keys within each block are in sorted (ascending) order. The system locates a given key by scanning the master level for the first key of equal or higher value. This selects one block of the coarse level which is scanned in a like manner to find the proper fine level block, which is then scanned for a match with the given key. Each key in the fine level references a data record by its real record number.

Figures 1.3 through 1.5 show the structure of an indexed data file. Also, see "Indexed Data Files" in the IRIS User Reference Manual.

Copyright 1978
Educational Data Systems

Figure 1.3: Indexed File Structure

This example shows an indexed contiguous data file with three directories:

```
 -------------------
 :        H        :   File header block.
 :-----------------:
 :       M1        :   Directory #1 master level.
 :-----------------:
 :       M2        :   Directory #2 master level.
 :-----------------:
 :       M3        :   Directory #3 master level.
 :-----------------:
 :                 :
 :       C1        :   Directory #1 coarse level.
 :                 :
 :-----------------:
 :                 :
 :                 :
 :       F1        :   Directory #1 fine level.
 :                 :
 :                 :
 :-----------------:
 :                 :
 :       C2        :   Directory #2 coarse level.
 :                 :
 :-----------------:
 :                 :
 :                 :
 :       F2        :   Directory #2 fine level.
 :                 :
 :                 :
 :-----------------:
 :                 :
 :       C3        :   Directory #3 coarse level.
 :                 :
 :-----------------:
 :                 :
 :                 :
 :       F3        :   Directory #3 fine level.
 :                 :
 :                 :
 :-----------------:
 :                 :
 :                 :   Data records.  If the record
 :                 :   length (R) is not a power of
 :                 :   two words then there may be
 :      Data       :   up to R-1 words unused at
 :                 :   the beginning of the first
 :                 :   data block and again at the
 :                 :   end of the last data block.
 :                 :
 -------------------
```

Figure 1.4: Indexed File Header

The FMAP cells of an indexed file's header are used for directory information as follows:

```
cell      Contains

  0       Number of directories in file

  1       Number of data records currently on free chain (NFDR)

  2       Record number of first data record on free chain (FFDR)

  3       First real data record number (FRDR)

  4       F, N, L   (see at right)          \
                                             \  For directory #1
  5       (not used)                          )
                                              )  F = Fine flag
  6       d(first free block for coarse level) )  N = # keys/block
                                             /   L = Key length
  7       d(first free block for fine level) /

 10       \
           \
 11         \
             )  Same as words 4-7, but for directory #2
 12         /
           /
 13       /

 14       etc. (four words per directory)

 15

  .

  .

  .

 74       \
           \
 75         \
             )  Same as words 4-7, but for directory #15
 76         /
           /
 77       /

100       (not used)
```

Figure 1.5: Directory Block Format

Each block of each directory of an indexed file is structured as shown below.

```
 ---------------------
:F:   N   :     L     :
:---------------------:
:         D           :
:---------------------:
:         P1          :
:---------------------:
:                     :
:         K1          :
:                     :
:---------------------:
:         P2          :
:---------------------:
:                     :
:         K2          :
:                     :
:---------------------:
:         P3          :
:---------------------:
:                     :
:         K3          :
:                     :
:---------------------:
:         P4          :
:---------------------:
:                     :
:         K4          :
:                     :
:---------------------:
:                     :
:         .           :
:                     :
:         .           :
:                     :
:         .           :
:                     :
:---------------------:
:         Pn          :
:---------------------:
:                     :
:         Kn          :
:                     :
:---------------------:
:                     :
:                     :
:                     :
 ---------------------
```

F = Fine level flag. Zero for master and coarse levels of directory, one in fine level.

N = Number of keys. Maximum number of keys per block in master level, number of active keys in block in coarse and fine levels.

L = Key length (# words).

D = Disc address of next block of this level of this directory (zero in last block in level.

K = Key value (ASCII string).

P = Pointer. Disc address of block at next level in master and coarse levels, or real record number of data block in fine level.

There may be up to L words unused at the end of each directory block.

Copyright 1978
Educational Data Systems

1.8 Control and Information Tables

IRIS uses several core-resident system control and information tables as folows:

INFO — This table contains various system information such as the current real time, CPU speed, disc addresses of certain processors and files, size of core, number of Logical Units, etc. The location, INFO, is specified in the Software Definitions tape as are displacements for all items in the table. A page zero pointer to the beginning of the table is defined in the Page Zero Definitions. Refer to a listing of the Software Definitions for a complete list of all items in the INFO table. A replacement for the first part of the INFO table is contained in the CONFIG file; see Section 2.4 for changes that may be made here.

LUT — The Logical Unit Table contains a three-word entry for each active Logical Unit. Each entry consists of:

word 0:	LUFIX pointer (identifies a disc driver)
word 1:	LUVAR pointer (identifies a partition)
word 2:	Logical Unit number

Refer to Section 11.8 for descriptions of the LUVAR and LUFIX tables. The Find Logical Unit Tables subroutine may be used to look up the LUFIX and LUVAR for a given Logical Unit. The LUFIX pointer will be complemented if the unit is not currently active, and the top bit of the Logical Unit number will be one if the unit is only temporarily active (ie, being installed). The LUFIX cell will be zero in each unused LUT entry. MILU (see page 2-12) specifies the size of LUT. There is a pointer to LUT in the INFO table (see page A5-2).

PCB — Each active port has a Port Control Block which contains various information about the status of the port and the user on the port. The Regnant User Pointer (RUP) in page zero points to the PCB of the user whose processor is currently in core, and the Regnant Task Pointer (RTP) points to the PCB or TCB (Task Control Block) of the task that has control of the system at any given time. Refer to a listing of the Software Definitions for descriptions and displacements of all items in the PCB. The PCB's as a group make up the Port Control Area (PCA) with the PCB's arranged in order of increasing port numbers; therefore, .PCA in REX page zero points to port zero's PCB.

DFT — Each active port also has a Data File Table which can be found via a pointer in its PCB. Each DFT has eight words per channel as described in the Software Definitions (see page A5-3).

Copyright 1978
Educational Data Systems

1.9 Flag and Status Words

One of the items in each PCB is a flag word defined as FLW. Each bit in FLW is a flag as follows (bit 15 is the most significant bit):

Bit	Meaning in FLW
15	Binary input mode (pass byte as is)
14	Binary output mode (no parity)
13	DSP breakpoint is set
12	DSP is active on this port
11	A signal will activate from pause
10	A break has been detected
9	
8	Processor task is in queue
7	Output is active
6	Input is active
5	End of pause will cause auto log off
4	Ignore CTRL E (log-on mode)
3	Ignore CTRL O
2	Enable XOFF and XON
1	Suppress parity check
0	Echo input characters

One of the items in each channel in a DFT is a status word defined as STS. Each bit in STS is a flag as follows (bit 15 is the most significant bit):

Bit	Meaning in STS
15	Record is locked
14	File is write protected
13	File is contiguous
12	File is not formatted
11	Peripheral device
10	File is indexed
9	(reserved for byte number overflow)
8	\
.	) Displacement of
.	) record into block
.	) (number of bytes)
0	/

1.10 Number Types and Formats

Nine different number formats are used in the IRIS system. Two of the forms, signed and unsigned binary integers, can be manipulated directly with the computer's machine code instructions. A third form is floating point binary. The other six forms are variations of binary-coded-decimal (BCD) formats. All nine forms are shown in detail in Figure 1.6. The floating binary and BCD formats are manipulated either by software subroutines ($DEC) or by use of the EDS 400-P1 Micro-N (through the $DAU driver).

Copyright 1978
Educational Data Systems

Figure 1.6: Number Formats

```
Unsigned binary integer           !15                          0!
 (type 12 in a format map)        --------------------------------
                                  !       16-bit binary integer  !
                                  -------------------------------o


Signed binary integer            !15!14                        0!
 (not used in a data file)        --------------------------------
                                  !S!     15-bit binary integer  !
                                  ! !   (two's complement if <0) !
                                  -------------------------------o


Floating point binary number (type 4 in a format map)

!              word 0             !             word 1           !
!15                             0 !15          9! 8           1!0!
----------------------------------------------------------------
!         absolute value of mantissa         ! exponent of 2  !S!
!          (23-bit binary fraction)          !  (excess 128)  ! !
o---------------------------------------------------------------


Unsigned BCD integer              !15   12!11    8! 7    4! 3   0!
 (not used in a data file)        --------------------------------
                                  ! BCD  !  BCD  !  BCD  !  BCD  !
                                  ! digit! digit ! digit ! digit !
                                  -------------------------------o


Signed BCD integer               !15!14 12!11    8! 7    4! 3   0!
 (see note 4)                     --------------------------------
                                  !S!3-bit!  BCD  !  BCD  !  BCD  !
                                  ! !digit! digit ! digit ! digit !
                                  -------------------------------o


Floating point BCD (see notes 4 through 7)

!              word 0             !             word 1           !
!15   12!11    8! 7    4! 3     0!15   12!11    8! 7           1!0!
----------------------------------------------------------------
! BCD  ! BCD  ! BCD  ! BCD  ! BCD  ! BCD  !   exponent      ! !
! digit! digit! digit! digit! digit! digit! (see note 6)!S!
o---------------------------------------------------------------
```

Figure 1.6: Number Formats (continued)

Six word unpacked BCD (not used in a data file)

```
¦    word 0     ¦    word 1     ¦    word 2     ¦    word 3     ¦
¦15            0¦15            0¦15            0¦15            0¦
-----------------------------------------------------------------
¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦ D ¦
o----------------------------------------------------------------


¦    word 4     ¦    word 5    ¦
¦15            0¦15           1¦0¦
---------------------------------
¦   exponent    ¦   not used   ¦S¦        D represents a BCD digit
¦ (see note 6)  ¦ (all zeroes) ¦ ¦
---------------------------------
```

Notes:

1) A small circle (o) in the base line of a word represents the binary point or decimal point of the mantissa.

2) An S represents the sign bit of the mantissa. In all cases, zero means positive, and one means negative.

3) The numbers above each figure are bit positions, where bit zero is the least significant bit of each word.

4) Any signed BCD integer or floating point BCD number is type 5 in a format map. The number size is determined by the item size (see Section 1.5). The mantissa is carried as absolute value and sign.

5) The three-word and four-word BCD formats are similar to the two-word format shown, except that the mantissa holds ten or fourteen BCD digits, respectively. The exponent and sign are always in the last word.

6) The exponent of a floating point BCD number is a binary integer representing a power of ten. In any packed form it is carried in excess 64 notation, but in the six-word unpacked form it is carried as an ordinary 16-bit binary number which will be in two's complement form if negative.

7) Any floating number form is assumed to be zero if the first word is zero (first digit if using hardware DAU).

1.11 System INDEX Structure

Each Logical Unit has a system INDEX of the files on that unit only. The Filenames are placed in the INDEX by use of a hashing algorithm such that a file lookup (FIND FILE) will usually find a desired Filename in the first block examined. Each INDEX is a non-extended random file, thus limiting its size to 128 data blocks. Each entry in an INDEX is eight words consisting of a Filename (an ASCII string of up to 14 characters) which occupies the first seven words of the entry, and the Real Disc Address of the file's header (on the same Logical Unit) in the eighth word. Each block can hold 256/8 = 32 such entries for a total of 128*32 = 4096 possible entries. Because of this limitation, a Logical Unit can not hold more than 4093 user files (4096 less three for the INDEX itself and the DMAP and ACCOUNTS files). Also, the hashing algorithm may become inefficient and slow if the INDEX is filled to more than 95% of capacity, so a Logical Unit should not be expected to hold more than about 3890 files. The maximum size of a Logical Unit is 65536 disc blocks, so this limitation will not be a problem as long as the average file size is not less than 65536/3890 = 17 blocks. On smaller Logical Units, the average file size can be proportionately smaller.

The INDEX is not hashed on Logical Unit number zero (the system unit). Entries are made in unit zero's INDEX in chronological order. A gap is left in the INDEX when a file is deleted, and the INDEX entry for the next file built will fill the first gap.

During an IPL, SIR scans the INDEX on Logical Unit zero. As each block is scanned, the disc addresses of the entries are sorted into ascending order, then the files are examined in that sequence in order to mark the used disc blocks in the DMAP. When any driver file is encountered during this procedure, it is made core-resident. Therefore, if it is necessary for two or more drivers to be brought into core in a specific sequence, it is best to put their INDEX entries in different blocks of the INDEX (DSP must be used to do this manually).

CAUTION! Do not manually change or move an INDEX entry in any INDEX other than for Logical Unit zero. Because of the hashing algorithm used, FIND FILE may not be able to find such a modified INDEX entry.

Copyright 1978
Educational Data Systems

2. SYSTEM INSTALLATION AND CONFIGURATION

The installation of a computer system may be separated into five main tasks:

1) Site preparation and maintenance,

2) Hardware selection and configuration (the computer and all peripheral devices),

3) Hardware installation and testing,

4) System generation (loading the operating system and other software onto the disc), and

5) Configuring the system for optimum performance.

Site preparation and maintenance are discussed in Section 2.1, and hardware selection is discussed in Section 2.2. Installation and testing procedures for the hardware must be derived from the manuals supplied by the various hardware manufacturers, however hardware testing is also discussed in the first few steps of the system generation procedure in Section 2.3. These considerations are often overlooked when purchasing or installing a computer system, so their importance cannot be overly stressed here. Such simple things as noisy devices on the power line (elevators, air conditioners, mechanical business machines, etc.) or a dirty environment may be overlooked or underrated when a mini computer is being installed, but remember that special power distribution and air conditioning are usually installed for a large computer system. A mini computer system is often expected to operate in the normal office environment. The assumption that it will do so with no special preparation may lead to distress.

System generation is outlined in a step-by-step procedure in Section 2.3, and this procedure should be followed carefully for an efficiently operating IRIS system. System generation is then followed by configuring the system to operate the most effeciently in the specific user job mix. The configuration of an IRIS system may be easily modified at any time if the job mix changes, or if peripheral devices are added or replaced, etc. Updates can be made to the system at any time to add new features or correct problems.

2.1 Site Preparation and Maintenance

Site preparation for an IRIS system is not very critical. Some minimal precautions must be taken, however, to assure reliable system performance. The following environmental conditions must be maintained to keep the EDSI warranty in effect:

Ambient Temperature:	15 to 35 degrees C (60 to 95 degrees F)
Relative humidity:	15% to 90% (non-condensating)
Telephone:	A telephone within reach of the computer console, which can be used for outgoing calls while seated at the master terminal, is required for maintenance purposes.
Electrical power:	105 to 125 VAC 47 to 63 Hz* single phase

* Some peripherals require 57 to 63 Hz line frequency and must be ordered special if 50 Hz operation is desired. Some large disc drives may require 230 VAC or three phase power.

The electrical power requires standard 3-wire, 15 ampere receptacles with a good earth ground (green wire), and it should be free of transients such as are caused by elevators, air conditioners, and large motors. In some cases, a separate line from the main power distribution box may be necessary.

Do not plug any device that has a motor that is started and stopped during system operation into the power outlet on the computer or into the same electrical circuit. Teletypes and the high speed paper tape punch are in this exclusion catagory, but the high speed paper tape reader may be plugged into the CPU for power. If an EDSI multiplexer is included in the system then its power supply must be plugged into the CPU's convenience outlet.

The air should be free of dust, smoke, and corrosive chemicals. Air filters must be inspected and cleaned regularly.

Metal or wooden chairs are recommended. Plastic chairs (solid plastic or vinyl upholstered) and carpets must be avoided in the computer room and at all terminals to avoid static electricity problems.

After the system is installed, run thorough diagnostic tests on the computer and each peripheral device before starting the system generation. A minor hardware failure could cause much time to be wasted in attempting to diagnose the problem later.

2.2 Hardware Configuration

Any operating system requires a certain minimum set of hardware for its operation. For IRIS, the hardware system must include:

1) Any Nova-type* computer (other than an Eclipse*) with at least 16K words of core or semiconductor memory and a control console. Power Fail Auto Restart is recommended, although not required. Automatic restart is impossible after power failure if the computer has semiconductor memory without battery backup, or if any disc drive on the system will not automatically return to a ready state when power is restored.

2) Any disc or drum system with at least 128K 16-bit words, and which will transfer data to and from CPU memory in blocks of exactly 256 words or any sub-multiple of 256 words. At least two disc drives (or one dual drive having a fixed platter and a removable cartridge) are required so that the system can be backed up by a disc-to-disc copy. See Section 2.2.2.

3) A master Teletype or program-equivalent terminal with standard device codes 10 and 11. An EDSI model 310 multiplexer with a user terminal on its first port will satisfy this requirement. Hard copy is desirable for maintenance and update purposes. If Teletypes are to be used in the system, it is recommended that they be ordered with type NU print wheels; this print wheel has the slashed letter O and the plain numeral zero.

4) A real time clock. Any device which will supply interrupts at ten Hertz or any multiple of ten Hz not exceeding 1000 Hz. The standard Real Time Clock, an EDSI multiplexer, or a bit-banger multiplexer will satisfy this requirement.

5) A paper tape reader. The standard device 12 high speed reader is recommended, but the reader on the master terminal (device 10) will suffice if it has been modified so that an NIOS TTI instruction will cause the reader to feed one character.

6) A telephone. A telephone which may be used for both incoming and outgoing calls must be within easy reach of the master terminal and the computer.

7) A Pico-N. IRIS requires an EDSI Pico-N to be installed on the computer backplane. A Pico-N is supplied without charge with each paid IRIS license. The behavior of IRIS will be erratic without a Pico-N. See Section 2.2.1 for details.

IRIS is a very open-ended system making it easy to add a wide variety of devices to the system without modifying the system itself. Drivers are already available for a wide variety of devices, and IRIS has a straightforward and well defined software interface, making it easy to add drivers for new devices.

* Nova and Eclipse are trademarks of Data General Corporation. Nova-type refers to any computer that is software compatible with Nova computers.

2.2.1 Pico-N Installation

IRIS requires a Pico-N for normal operation. The Pico-N consists of a 100-pin connector with some encapsulated circuitry. Turn off CPU power, and install the Pico-N by pushing its connector over the "A" side pins of any slot except slot one on the computer backplane. Be sure that the Pico-N is oriented properly (the top and bottom are clearly marked), and that all 100 pins are in its connector; the Pico-N may be destroyed if installed incorrectly (eg, shifted either right or left). It does not matter whether or not the selected slot contains a board. The Pico-N draws power from pins A97 through A100, and connects to the even-numbered pins A38 through A74, and to pins A80 and A82. There is no connection to any other pin. The Pico-N will not affect normal operation of the computer nor of any peripheral device.

In most cases, there is no need to remove the Pico-N to run diagnostics or any other programs or operating systems. The standard test routines (CPU exerciser, logic test, memory address test, memory checkerboard, disc reliability, etc.) should be run if any hardware problem is suspected. If any standard test runs with the Pico-N removed, but errors are indicated when the Pico-N is installed, it may be necessary to modify the test routine; EDSI will assist the user in making necessary modifications if provided with a listing and object tape of the routine. If the Pico-N causes errors with a test routine that has been so modified, then it should be returned to Educational Data Systems, and a new one will be supplied.

The Pico-N will remain the property of Educational Data Systems. A Pico-N is supplied without charge under a non-transferrable license with each paid IRIS license.

2.2.2 Data Channel Priority

Route the data channel priority (DCHP on pins A93 and A94 on the computer backplane) so that the disc controller has the highest priority. If the system includes a nine-track magnetic tape then its controller should have second highest priority on the data channel. Data Channel multiplexers and other DMA devices must have lower priority so as not to interfere with disc and tape transfers. Most disc controllers that are completely external to the computer require modifications to allow the required routing; the System Industries models 3015 and 3045 disc controllers are in this category. A memo is available from Educational Data Systems giving detailed modification instructions. If the system includes a Micro-N, it should have the lowest DCH priority.

2.2.3 Interrupt Priority

Route the interrupt priority (INTP on pins A95 and A96 on the computer backplane) in the normal manner. High interrupt rate devices, such as most non-EDSI multiplexers, should have the highest interrupt priority. INTP must reach all devices.

2.3 System Generation

The SysGen (SYStem GENeration) procedure is used only when it is necessary to generate a complete new IRIS system on the disc. Procedures for later additions and modifications are discussed in Section 4 of this manual. Fill out an IRIS SysGen Record form as the SysGen is done, and save this record for later reference. The following steps should be followed carefully in the order given:

2.3.1 Run all computer diagnostic routines. It is particularly important to run the Power Fail Auto Restart test, the Memory Address test, and the Memory Checkerboard test. Also run the Exerciser with the punch and reader operating to check out the interrupt system. If the computer is new, the Exerciser should be left running over night, but it is not necessary to run the punch and reader for more than two passes of the Exerciser.

2.3.2 Run the disc reliability test routines, thoroughly testing all disc surfaces. If there are any problems, correct them before continuing the SysGen. The disc reliability test should be left running over night. Any bad blocks other than in the first few tracks may be identified to IRIS in 2.3.4 below.

2.3.3 Use the EDSI binary paper tape loader (described in Appendix 8) to load the four system generation tapes (REX, SYSGEN, DBUG, and BZUP object tapes), then load the proper Sysgen OVerlay tape (SOV). The SOV tape must be loaded last, after which the computer should automatically initiate system generation; if it does not, then RESET and START at location 34000.

2.3.4 SYSL (the SYStem Loader, which is contained in the SYSGEN tape) will print instructions for its operation. The first question asked is "BAD BLOCK?". Type the Real Disc Address of any block on the system disc that is known to be bad, and press the RETURN key. After all bad blocks have been reported in this manner, or if there are none to report, press only the RETURN key to proceed with the SysGen.

2.3.5 SYSL will ask for the DISCSUBS Group #1 object tape to be loaded in the tape reader and will then ask which reader is to be used. Place the tape in the reader, and then press the "1" key if using the high speed paper tape reader, or press the "0" key if loading through the master terminal. The DISCSUBS tape will be loaded onto the disc. If tapes are loaded through the master terminal's reader then a null code will be sent to the master terminal after each tape record (about four inches) is check summed; this will cause the master terminal to cycle (printing nothing) to indicate that the tape is being read without errors.

2.3.6 SYSL will print PLEASE WAIT while it does the basic system generation, and will then ask "NAME?" for each additional tape. SCOPE, BYE, DSP, CONFIG, and the real time clock driver (eg $RTC, $MMUX, or $BBMX) must be loaded at this time. The NAME requested by SYSL is the Filename which will be written on each tape. The standard IRIS names must be used for all files. The "TYPE?" requested by SYSL is a five-digit octal number given on the SysGen Record. See Section 1.4 to determine the proper file type for a file not listed on the SysGen Record.

2.3.7 Press CTRL C when asked "NAME?" after loading the five tapes listed above. This will start an IPL (see Section 5.1 for responses to printouts by SIR during an IPL). After the IPL, check that the CARRY light is flashing; if it is not, there is a problem that <u>must</u> be rectified before continuing the SysGen; if the problem is in the hardware (eg, device not in system), the SysGen may be resumed by doing another IPL after the problem is fixed; the SysgGen must be restarted from scratch if it is a software problem. After a successful IPL, press the ESC key and sign on to the Manager's account by typing MANAGER when asked for an Account ID (see "How to Log On" in the IRIS <u>User Reference Manual</u>).

2.3.8 Use PLOAD (see Section 4.1) to load any remaining interactive port drivers such as $DGMX, $ALU, $TTY, $PHA, and $CRT. Load $PTR and $PTP if the high speed paper tape reader and punch are on the system; otherwise load $PTM to use the master terminal's punch and reader as peripherals. For best system performance, do not load any other tapes at this time.

2.3.9 All processors requiring passwords are assembled with "X" as the password. Therefore, DSP <u>E</u>X may now be typed to call the Disc Service Processor (an underlined letter in a command indicates a control character; eg, CTRL E in this case). Use DSP to make any necessary modifications to CONFIG and to set all multiplexer, phantom port, and peripheral driver attributes as required, then do another IPL (RESET and START at location 4) to bring the drivers into core and allocate the active files. PLEASE WAIT may be printed several times during this IPL as drivers are encountered and adustments are made by SIR to accommodate them.

2.3.10 Log on to the manager account again, and load the DISCSUBS Group #2 and/or Group #3 obect tapes if they have been supplied. DISCSUBS Group #2 contains subroutines required by Business BASIC, and Group #3 contains the subroutines for extended and contiguous data files and text files. Give DSP an "F DISCSUBS" command, then give an "R" command to load each of these tapes. Reload the tape from the beginning if any error occurs. DSP's "R" command will not work if the real time clock is not running as indicated by the one Hertz flashing of the CARRY light. If the master terminal's reader is to be used, see "How to Load a Text File" in the IRIS <u>User Reference Manual</u> for restrictions on the use of $PTM.

2.3.11 Use PLOAD to finish the SysGen by loading all remaining tapes, starting with BASIC, RUN, RUNMAT, ASSEMBLE, and EDIT for best proximity to the active files. ASSEMBLE may be loaded under the name ASM, if desired, for easier typing and to allow more room on the command line for Filenames (see "How to ASSEMBLE a Program" in the IRIS User Reference Manual). Text file and BASIC program tapes (SYMBOLS, UTILITY, ACCOUNTLIST, SPOOLER, CRAM, BUILDXF, USERID, DEFS, PZ, etc.) are not loaded by SYSL or PLOAD: see 2.3.20 through 2.3.22.

2.3.12 Do another IPL (RESET and START at location 4) to make all subroutines accessible, and log on to the manager account again. Use DSP to make any necessary patches (see Section 4.10). Also, a password (any string of up to 15 letters) must be entered at 570 in KILL, PORT, DSP, CLEANUP, and SHUTDOWN. For example, if the password to KILL is to be PROTECT, give DSP the commands:

```
F KILL
I570:PROTECT
```

2.3.13 BYE normally sets parity checking each time a port is logged off. If this feature is not desired (ie, it is desired to leave the parity check "switch" in its then current position when a user logs off) then enter a zero at location 200 in BYE. Any non-zero value in location 200 will cause BYE to set partiy checking each time a port is logged off. To change this switch, give DSP the commands:

```
F BYE
200:0    (or 200:1 to set parity checking each log off)
```

2.3.14 BYE normally prints all accounting information each time any user logs on or off. However, if not all of this information is desired, an entry may be made in location 201 of BYE to suppress specific portions. Each bit in the word suppresses one item of accounting information as follows:

Bit	Item Suppressed	
11	Port #	
10	Account #	
9_	Date & Time	During
8	(not used)	Log-On
7	Time left	
6_	Blocks Avail.	
5	Account #	
4	Date & Time	
3_	Charges	During
2	CPU Time	Log-Off
1	Connect Time	
0	Blocks Avail.	

Note: Bit 15 is the most significant bit. Bits 15-12 are not used.

2.3.15 If a different welcome message is desired at log-on time, use DSP to enter the message (63 characters maximum) into BYE at location 540. Enter CTRL Z codes for carriage returns. For example, DSP might be given the commands:

```
   F BYE
   I540:ZZWELCOME TO "IRIS" TIMESHARING !
```

Also, see Sections 2.11 through 2.14 for provisions for additional log-on messages, log-on charges, log-on restrictions, and automatic program start after log on.

2.3.16 SHUTDOWN is normally restricted to the system manager. To let a user on another account shut down the system, enter the account number in octal (same as in ACCOUNTS file) in location 200 in SHUTDOWN, or enter 177777 octal in location 200 to allow any account to shut down the system. Also, SHUTDOWN is normally restricted to operation from port zero: to allow its use from another port, enter the port number (in octal) in location 201 of SHUTDOWN, or enter 177777 octal in location 201 to allow the use of SHUTDOWN from any port. In any case, a user must know the password to SHUTDOWN to shut down the system.

2.3.17 INSTALL normally deletes a "bad" file (eg, Filename or header disc address doesn't match INDEX entry, wrong number of disc addresses in header, or header uses a disc address already marked as "in use"), but retains a file that has its "build" bit set (ie, it was being built and the system was stopped before it was closed) and is otherwise correct. Location 200 of INSTALL controls the decision on questionable files, and the value one in location 200 is the default value that causes the decision to be made as just described. If it is desired to delete a file being built instead of retaining it, then use DSP to enter the value two in location 200 of INSTALL. A zero in location 200 causes INSTALL to retain all questionable files, but a 63277 halt will occur when a "bad" file is encountered (see Section 6.4).

2.3.18 INSTALL FAST is normally not allowed because of the possibility of a bad DMAP in the Logical Unit. However, in a system where disc packs are being changed often, the ability to install the new pack without the long delay of a normal install becomes very desirable. To permit the INSTALL FAST by the system manager, enter the value one in location 201 of INSTALL. To allow the INSTALL FAST by a user on any account, enter any value greater than one in 201. A zero in 201 causes the INSTALL FAST to be prohibited. Note: INSTALL FAST never deletes a file regardless of its status.

WARNING!

Do not use the INSTALL FAST after a system crash or if the data on the Logical Unit being installed is suspect for any reason.

2.3.19 PORT MONITOR is normally restricted to the system manager. To allow any user to use this function, enter a 1 in location 200 of PORT.

↑N WARNING: This function is usually restricted to the system manager (by a zero in location 200) because PORT MONITOR lists each user's program's Filename, including the password (if any).

2.3.20 To load UTILITY, ACCOUNTLIST, CRAM, SPOOLER, BUILDXF, or any other BASIC program, see "How to Load a BASIC Program from Tape" in the IRIS User Reference Manual. After Loading each tape, press CTRL C, and save the program (see "How to SAVE a BASIC Program" in the same manual). Be sure to type NEW before loading each tape.

2.3.21 If ASSEMBLE is on the system then SYMBOLS is required. To load SYMBOLS or any other source file (such as DEFS or PZ) see "How to Load a Text File" in the IRIS User Reference Manual. SYMBOLS is not required for the extended assembler (XASSEMBLE).

2.3.22 Use FORMAT to build a file named USERID (formatted D2,S14) for use by the Account UTILITY and ACCOUNTLIST programs (see "How to FORMAT a Data File" in the IRIS User Reference Manual). Then use the Account UTILITY Program (see Section 7.3) to change the MANAGER, PRIV1 and FREE account ID's to secret ID strings, and add additional users' accounts as desired. The record format of the USERID file may be extended if desired for use in resource charge accounting, etc. Item zero must be type D2, however, and item one must be a string of at least 14 bytes because UTILITY uses these two items. It is strongly recommended that the manager account not be used for purposes other than the system itself!

2.3.23 Use DSP to make any further modifications necessary in the CONFIG file (see Section 2.4). Also set up the port attributes tables, etc., for each peripheral driver as described in Appendix 6. The port definition table for port zero is in the REX file, and there is a pointer to it at location 200 in REX. The baud rate specified for port zero is meaningful only if using an EDSI model 310 multiplexer with the device 10/11 option enabled.

2.3.24 If BASIC, RUN, and RUNMAT have been loaded then press CTRL C, type BASIC, and press the RETURN key twice to link these three processors together. Use BASIC's "SIZE" command to determine if the program area has the desired size; if not, see Section 2.8.

2.3.25 A backup copy of the software system should be made at this time and periodically hereafter (eg, once a day). See Section 8.5 for system BACKUP Procedures.

2.4 How to Modify the System Configuration

The IRIS system has been optimized for a particular hardware configuration by either EDSI or one of EDSI's licensees. In your application you may desire certain system modifications to increase your system's performance. EDSI will supply two system listings: the CONFIG file listing showing the locations of system parameters, and the IRIS Software Definitions (see Appendix 5) which defines all system subroutines, labels, and displacements.

2.4.1 The CONFIG file holds configuration parameters which may be examined and modified by use of DSP. After any modifications have been made, an IPL must be done to load the newly configured system into core.

```
*************************************************
* The following table shows the location of various  *
* information in the CONFIG file for IRIS R7.3 only! *
* Refer to the proper IRIS Software Definitions for  *
* the correct locations in other IRIS systems.       *
*************************************************
```

Location	Contents of CONFIG File
0-277	(Not currently used).
300-377	Disc address of each peripheral driver's initializing routine (first block of driver). This list is set up by SIR for use by RECOVER. Do not change!
400	LBSA (Location of BSA). The assigned location of BSA as defined here must be at least 31400 to allow any DISCSUBS to become core-resident, and the absolute minimum value of LBSA is MBUS+PSIZ (see pages 2-11 and 2-12) or 30400, whichever is greater. LBSA is usually set equal to MBUS+PSIZ for optimum core usage; otherwise, LBSA-MBUS-PSIZ words of core will be wasted because the area between MBUS and LBSA can be occupied only by partition zero. See Sections 2.4.3 and 2.8 for other considerations.
401	PSIZ (Partition SIZe). The size must be a multiple of 400 octal, and all partitions are the same size. The minimum value is 1000 octal, and the maximum value is LBSA-MBUS or 40000 octal, whichever is less. If the value in this cell is not a multiple of 400 octal, then cells 401-404 are ignored, and the system is initialized with one non-lockable partition of size LBSA-MBUS at the top of the processor swapping area. See Sections 2.4.3 and 2.8 for details.

402 NPLC (Number of Partitions in Lower Core, including the partition at the top of the processor swapping area). The minimum value is one, and the maximum value will be determined by the partition size (see PSIZ on opposite page) and by the values of LBSA and MBUS (see Section 2.8).

403 NPUC (Number of Partitions in Upper Core). Must not exceed the amount of upper core (TOPW-077777, or zero if TOPW < 100000) divided by the partition size (see PSIZ opposite), but it may be smaller if it is desired to allow more core for the buffer pool. The number of buffers in upper core will equal (TOPW-077777-PSIZ*NPUC) / 400, or zero if TOPW < 100000. SIR will allocate as many buffers as possible in lower core just before allocating the free nodes, and the total number of buffers in the pool will be put in the NRBP cell of the INFO table.

404 NLOK (Number of LOcKable partitions). Must be less than the total number of partitions (NPLC+NPUC) so that there is at least one non-lockable partition.

405-577 (Not currently used).

600-622 System information as defined for the INFO Table in the IRIS Software Definitions. See the following pages for changes that may be made here.

623-777 (Not currently used).

1000-1377 Core-resident DISCSUB numbers. This list is described in Section 2.9.

1400-1777 Disc driver table. See Section 2.6.

2000-15777 Disc drivers. A BZUP driver and a system driver are included for every disc controller to which IRIS has currently been interfaced. See Sections 11.8 and 11.9 for information on adding disc drivers in the CONFIG file.

16000-16377 Log-on Restrictions (see Section 2.13).

16400-17777 Auto Program Start table (see Section 2.14).

20000-77777 System history (not currently used).

The following parameters in the CONFIG file's INFO table may be changed as described:

Loc. Label **Parameter**

600 SDAT System creation DAte (hours after 1-1-76). Do not change!

601 SPED Average CPU SPEeD in instructions per millisecond:

Computer	Speed (Octal)
Nova	302
Nova 1200 or D-116	653
Nova 2 or D-116H	770
Nova 800	1325
Super Nova	1255
Super Nova SC	1762
Nova 3	100770 *

* Note: bit 15 must be "1" for a Nova 3 only!

602 MILU Maximum number of Installed Logical Units. Must be at least equal to the number of logical partitions (see Section 2.6).

603 NDCH Number of Data CHannels per port. Each data channel occupies eight words of core for each interactive port. NDCH is usually set to ten (octal 12). Minimum value is 2.

604 LPCA Location of Port Control Area (PCB for port zero). Will be adjusted automatically by SIR if any driver's attributes table specifies a PCB location. LPCA must be in the range LBSA+2000+ABUF+TBUF <= LPCA <= TOPW+1-TNAP*40 where TOPW does not exceed 077777 for the purposes of this calculation.

605 TNAP Total Number of interActive ports. Will be automatically increased by SIR if less than the number of interactive ports indicated in all driver's attributes tables.

606 (Not currently used).

607 MBUS Minimum Beginning of User Storage. This cell indicates the first available core space above the RUN processor for user storage. MBUS must be an integral multiple of 400 octal greater than BPS. Do not change MBUS unless RUN is modified accordingly!

Copyright 1978
Educational Data Systems

610 TOPW TOP Word of core memory to be used. IRIS will ignore any core above this address. The minimum value for TOPW is 37777 octal (for 16K words), and the maximum value is 177777 (for 64K words). Any core available above 77777 octal will be used only for user partitions.

611 ABUF Size of Auxiliary BUFfer Area (number of words). Must be 1004 words octal if indexed data files are to be used.

612 TBUF Magnetic Tape BUFfer Area. This cell should contain zero unless the system includes a nine-track magnetic tape ($MTAS), in which case this cell should indicate the buffer size for the largest tape record.

613 NCQN Number of extra Character Queue Nodes. SIR allocates two nodes per interactive port plus this number of extra nodes. Extra nodes are required to handle peak input rates if particularly heavy character processing is required. Each node occupies two words of core. Minimum value is two.

614 NNOD Minimum Number of free NODes desired in core for use as task queue nodes. This value is the only limit on the number of concurrent tasks in the system. Each node occupies ten words (decimal) in core. The difference between the value in this location in core and in this location in the CONFIG file indicates available space for core-resident discsubs, etc., because all remaining core at the end of an IPL is allocated as free nodes.

615 NSIG Number of SIGnal buffer nodes. This is the maximum total number of signals that can be waiting to be received. Each node occupies four words of core. Minimum value is one.

616 NSUB Maximum Number of DISCSUBS subroutines. The minimum value is one greater than the largest subroutine number in the DISCSUBS file. Causes NSUB*2 words of core to be used for pointer tables.

617 PTSL Time SLice Parameters: Left byte = long time slice, right byte = short time slice, both in tenth-seconds (see Section 2.5).

620 TLOK Minimum Time a LOcKed partition is kept locked (see Section 2.5).

2.4.2 To determine the maximum partition size (PSIZ) for a given IRIS system, start with the minimum configuration which is:

One user partition in 16K words core
$DEC (Decimal arithmetic routines)
$RTC (Real Time Clock driver)
Master terminal only (79 byte buffer, 10 data channels)
One buffer in disc buffer pool, one signal buffer node
No core-resident DISCSUBS (system may be slow)
No optional disc buffer areas (can't use indexed files)
No peripheral devices, multiplexers, etc.
No $TERMS (no form control or CRT cursor control)

In this configuration PSIZ may be 13000 octal. This will allow a 5214 word BASIC program, but a large system disc driver may reduce this by 256 words decimal. This limit may be increased only by:

1) Adding a Micro-N (EDSI model 400-P1) to the system, and loading $DAU (the driver for the Micro-N) instead of $DEC ($DAU is about 500 words smaller than $DEC), or

2) Adding more core memory. All core added (up to 32K total) becomes available for all purposes. Upper core (up to 64K words total) may be added to provide more user partitions and/or provide more buffers for the disc buffer pool.

The maximum Business BASIC program size will be decreased by any other changes to the configuration. To add interactive ports, it is necessary to add one or more of the following drivers:

$MMUX (one driver for up to 128 ports, see note)	431 words
$DGMX (Data General 4060, 4 ports)	108
$BBMX (bit-banger mux driver, up to 16 ports)	88
$ALU (DCC ALU mux driver, up to 8 ports)	155
$TTY (single TTY or CRT interface, one port)	51
$PHA (phantom port driver, one or more phantom ports)	7

Note: a PCB (see below) is required for each physical port on an EDSI multiplexer ($MMUX) even if the port is unused! The above numbers are the size of the driver only. $RTC must be deleted (or its name changed to RTC) when either $MMUX or $BBMX is loaded; the size shown for $MMUX or $BBMX is the increase in size over that of $RTC. For each interactive or phantom port, add:

Port Control Block (PCB)	32	words
Data File Table (N = # data channels)	8*(N+4)	
Input/output buffer (B = odd # bytes)	(B+1)/2	
Miscellaneous (character queue, etc.)	44	
Signal buffer nodes (S = # nodes)	4*S	

The input/output buffer for any interactive or phantom port must be at least two bytes larger than the longest line to be output to the port's terminal or to any peripheral device such as the line printer. For non-interactive ports (eg, a multiplexer port used for a peripheral device) only the PCB is required (an I/O buffer is also required if not supplied by the peripheral driver).

Other drivers that may be added to the system include:

$PTR (Paper Tape Reader)	420 words	
$PTP (Paper Tape Punch)	363	*
$PTM (use master terminal as peripheral)	620	
$LPT (line printer, add buffer size also)	342	
$CRD (DGC card reader on parallel interface)	250	
$CRD (Documation card reader on EDSI mux)	272	
$MTA (9-track magnetic tape, see MTBA below)	300 + 95/drive	
$TERMS (Form and cursor control system driver)	224	
$TERMn (Specific terminal type cursor control)	50 to 150	

Either $PTR or $PTM is required for most system updates. It is possible to load one of these as "PTR" or "PTM" and activate it temporarily for an update by using CHANGE to put the "$" in its Filename and doing an IPL, but this should be avoided if possible as it is usually necessary to also change some other part of the configuration to make the core space available.

The available core is also reduced by allocating optional disc buffer areas. These core areas and their uses are:

ABA (516 words required for any indexed file access)
MTBA (as required for largest record on magnetic tape)
Disc block buffer pool (260 words per buffer)

The available core is also reduced by causing any disc-resident subroutine (DISCSUB) to become core-resident (see Section 2.9). Some that are recommended to be core-resident and their uses are:

AFSETUP (active file setup for swap-out)	90	*
GETRR (get record, used for all data file access)	249	
READITEM (used for all formatted file access)	192	
READCONTIG (contiguous and text file access)	247	
BUILD (used when building any disc file)	490	
ALLOCATE (build or add to non-contiguous file)	252	
ALCONTIG (build contiguous file only)	246	
OPEN, CHARGE (open any data file or driver)	276	
CLOSE, CLEAR (close any data file or driver)	188	
ACNTL, FFILE (build, open, close any file)	470	
SEARCH, DIRECTORY, etc. (indexed file operations)	1206	
SPECIAL (BASIC's SPC functions)	103	
PATNF (BASIC's ATN function)	206	
PLOGF (BASIC's LOG function, "↑" operator)	249	
PEXPF (BASIC's EXP function, "↑" operator)	320	
PSQRF (BASIC's SQR, SIN, COS functions)	245	
PSINF, PTANF (BASIC's SIN, COS, TAN functions)	500	
SIGPA (Signal and Pause statements)	119	
Magnetic tape routines (reduce $MTA overhead)	950	

* The sizes given are approximate and may change as the system is updated. It is strongly recommended that AFSETUP be made core-resident as this will save at least one disc revolution each time slice.

2.4.3 Typical Configuration

A typical system might have 48K words of core in which it is desired to have two user partitions in lower core. Assume three phantom ports and an EDSI 8-port mux with one port used for a line printer. The calculation for added port buffers is as follows:

32	PCB (see page 2-14 for source of these numbers)
112	DFT (8*[10+4] for ten channels)
67	I/O buffer (133 bytes, not necessarily same for all ports)
44	Miscellaneous (fixed)
8	Two signal buffer nodes

263	x 10 ports (7 mux, 3 phantom) = 2630 words

The following modules and buffer areas will be required (the numbers come from pages 2-14 and 2-15):

# words	For
7	$PHA (Phantom ports driver)
516	ABA (For use of indexed files)
3328	13 buffers for buffer pool in lower core
431	$MMUX (For EDSI 300 or 310 multiplexer)
2630	I/O buffers, etc. for 10 ports (see above)
310	$TERMS, $TERMn (For cursor control)
374	$LPT plus 32-word PCB for mux port used
512	1024 byte buffer for $LPT
90	AFSETUP core-resident
249	GETRR core-resident
247	READCONTIG core-resident
276	OPEN, CHARGE core-resident
188	CLOSE, CLEAR core-resident
470	ACNTL, FFILE core-resident
1206	SEARCH, DIRECTORY core-resident

10834	Total of above

The increase in available lower core space can now be calculated by subtracting the total words required above from the 16K added to lower core to get 16384-10834 = 5550 (12656 octal). This may be added to the minimum configuration PSIZ for a total of 13000+12656 = 25656 words available for user partitions. Dividing this by two (for two partitions desired) results in a maximum PSIZ of 12727, but this is not a multiple of 400, so set PSIZ = 12400 octal, and set LBSA = MBUS+PSIZ = 20200+12400 = 32600 after confirming that LBSA is not too small to allow core-resident discsubs (see Section 2.4.1). The maximum BASIC program size will be 4958 words (see Figure 2.2). There can be two partitions in upper core for a total of four user partitions, and all upper core not occupied by user partitions will be assigned to the buffer pool, so the remaining 40000-15400*2 = 5000 words of upper core would be used for 5000/400 = 12 buffers octal. Preferrably, set NPUC=1 for a total of three partitions to allow decimal 37 buffers in upper core for a total of 13+37 = 50 buffers. Five to ten buffers per interactive port are recommended for the buffer pool.

2.5 Time-Sharing Algorithm

This section describes the time sharing and partition assignment algorithm. It begins with a few definitions.

2.5.1 States of a Job - A "job" is the set of all tasks required to serve one user at one interactive port. A port, and hence the job, may be in any of seven states as follows:

```
1.  Input/output done   \
2.  Pause done           )-  Job wants CPU
3.  Compute bound       /
4.  Regnant               -  Job has CPU
5.  Input/output active \
6.  Pause                )-  Doesn't want CPU
7.  Not logged on       /
```

2.5.2 Response Time - System response time is the time after a user presses a key that requires an interactive response (eg, RETURN or ESC) until that user's processor is given control so that it can start processing and formulate a response. This is not the same as "apparent response", which is the delay seen by the user between pressing the key and getting the next output (the formulated response). Apparent response is not discussed further because the system has no control over the amount of computing that must be done in an application program in order to formulate the response. It should be noted, however, that apparent response time can sometimes be improved in an application program by outputting some response (followed by a SIGNAL 3,0), before beginning a time-consuming disc operation.

2.5.3 I/O Bound Job (States 1 or 2) - A job is I/O bound if, after each input, it completes the calculations necessary to output a prompt message (if required) and enables the next input before the end of one time slice. A job is also I/O bound if it starts an output each time slice. An output to a peripheral device such as a line printer will also cause a program to be I/O bound if the program generates data faster than the peripheral can accept it, thus causing the program to enter a pause state until the device driver can accept more data.

2.5.4 Compute Bound Job (State 3) - A job is compute bound if it does not start the next I/O in one time slice. In a heavily loaded system, a compromise must be made between keyboard response and compute bound throughput. A normally interactive (I/O bound) job can become compute bound occasionally if its calculations take longer than usual or if the time slice is short due to other interactive users.

2.5.5 Time Sharing Algorithm - All users who want the CPU (ie, users in states 1 through 4 as defined above) are kept on the priority task queue (see Section 8.11). Each processor task has a priority in the range 3 to 1377 octal, and the queue is kept ordered by priority. A user's priority gradually rises while waiting for CPU time, and decreases while receiving CPU time (ie, when the user is "regnant").

Specifically, the algorithm is as follows:

a) The user's Effective Priority (EFP) is computed when the program is invoked, using the formula

EFP = 2 * APRI + PPRI

where APRI = Account Priority (see Figure 2.1), and
PPRI = Program Priority

except that if PPRI < 100 octal then EFP = PPRI, and the job remains a background job, regardless of the user's APRI. The initial CPR (Current Priority) is set equal to EFP.

b) The regnant user's CPR drops at a rate such that it will reach EFP in one long time slice, starting at CPR at the time the user becomes regnant, or starting at 1200 if CPR < 1200. The CPR does not drop below EFP. If less than a long time slice is used, then the job will be left with a CPR greater than its EFP.

c) All other users (both on and off the queue) except background jobs have their CPR's rise uniformly at a slower rate, dependent on the number of users on the queue, and determined such that the average user becomes regnant with a CPR = 1200. The CPR is not allowed to rise above 1377. Background jobs (whose EFP is less that 100) keep their CPR fixed at their EFP, so that they never compete for resources with non-background jobs.

d) When a user's time slice begins, a long time slice is assigned, except that this is reduced to a short time slice whenever there is any I/O bound user with CPR > 1200 on the queue.

e) When a user's time slice ends compute bound (ie, the job still wants CPU time), or when a user starts to want the CPU because of I/O done or pause done, such a job is put on the queue at a position indicated by its CPR at that time. Thus a user who has been typing in input from the keyboard, and therefore not using CPU time, will have its CPR rise to a high value and will therefore get a high position on the queue, resulting in prompt response.

Copyright 1978
Educational Data Systems

2.5.6 Partition Assignment Algorithm - When a job must be swapped into core, it may be necessary to select a partition that is assigned to another job and release it to make room for this job. The algorithm for choosing the partition for the new user is the following:

a) If any unused partitions are available, choose one (if possible) whose lockability status matches the program's. In other words, if the choice exists, give a lockable program a lockable partition and vice versa.

b) If any partitions exist which are not currently on the queue (ie, they are in states 5, 6, or 7) choose the partition which has been off the queue for the longest time, subject however to the constraint that a locked partition is not chosen if its off-queue time is less than TLOK.

c) Otherwise, choose the partition with the lowest CPR on the queue (this is always possible since there must be at least one non-lockable partition in the system).

2.5.7 Time-sharing Parameters - Two words in the system INFO table provide parameters for the partition assignment and time-sharing algorithms. These are PTSL and TLOK. The use and effect of these parameters is discussed in Figure 2.1 below. The system manager may use DSP to modify these parameters in core during normal system operation, then enter the values so determined into the CONFIG file when the most satisfactory compromise has been established.

Figure 2.1: Time Sharing Parameters

Parameter	Remarks
APRI	Account Priority - This is the user's assigned priority as set by the system manager in the ACCOUNTS file. The allowable range is 1 to 377 octal (1 to 255 decimal). A nominal value of 200 octal is assumed if the assigned value is 0. APRI indirectly affects all scheduling for the user (see EFP below).
PPRI	Program Priority - The system manager may use DSP to assign a priority to a user's program by entering a value in the range 1 to 377 in the program's PPRI cell (location 35 of the file's header). A nominal value of 200 is assumed if this cell is zero. The program's priority is carried forward without change whenever it is reSAVEd. A value less than 100 octal designates a background job, which receives CPU time only when no jobs with higher priority want the CPU. See EFP below.

EFP Effective Priority - This is not directly set by the manager, but is calculated by the system when the program is invoked as EFP = 2*APRI + PPRI. A user with high effective priority tends to get faster response and better throughput than a user with lower priority. Since the ranges of APRI and PPRI are 1 to 377 octal, the range of EFP is 3 to 1375 octal with a nominal value of 600 octal.

CPR Current Priority - CPR, the current priority of a particular task, is not set directly by the user or system manager, but the system computes CPR from the user's EFP and his CPU usage as described in Section 2.5.5. Each user's job is made up of a sequence of tasks, and all tasks are executed in order of highest CPR first. The possible range is 100 to 1377 octal for a timesharing task or 1 to 77 octal for a background task.

LPF Lockable Program Flag - The system manager may use CHANGE to set a program's Lockable Program Flag (bit 5 of the TYPE word of a program file's header). User operation is unchanged, but as soon as a lockable partition is available after the program is invoked by any user, the scheduler will assign the program to that partition. Once in a lockable partition, a lockable program will remain there without swapping until it is aborted, it terminates itself, or it becomes inactive for a long period of time (see TLOK below). The program's lockable program flag is carried forward when a program is reSAVEd.

PTSL Time Slice Parameters - Set the PTSL word to octal 400 times the desired long time slice (LTSL) plus the desired short time slice (STSL). Both LTSL and STSL are in tenth-seconds. Their default values are LTSL = 2 seconds and STSL = 0.3 second. Each user is given a time slice equal to LTSL unless an I/O bound user is on the queue, in which case the time slice is limited to STSL. Reducing STSL may give faster response time but at the expense of increased swapping overhead. Further, a user can easily go from I/O bound to compute bound if he is not given enough time to do some processing once he is swapped in.

TLOK Minimum time a locked partition is kept locked, in tenth-seconds. Default value is 30 seconds. A locked partition is not eligible for assignment to a different program - not even to a lockable program - until at least TLOK tenth-seconds after the last time it used the CPU.

2.6 Disc Driver Table

The disc driver table starting at location 1400 of the CONFIG file is set up by the SysGen Overlay (SOV) tape for the disc system on which IRIS is to operate. The table may be modified at any time to reflect additional disc drives and even additional disc controllers, and an IPL will cause the necessary drivers to be linked in to the operating system. Assuming disc controllers for which CONFIG already contains drivers, no other changes are required as long as the system disc (Physical Unit 0.0) remains unchanged, but a new disc controller or an unusual device code may necessitate additional changes to the CONFIG file.

The Physical Unit designator consists of a one digit driver number, a period, and the drive or partition number. Note that the number before the period is the number of the driver (a piece of software, not a controller or disc drive). Usually there is one driver per controller because two controllers cannot have the same device code, but a driver could be written to modify its own I/O instructions to drive several similar controllers; all partitions on all controllers interfaced by this driver would use the same driver number. Active drivers are always numbered sequentially starting with zero.

One or more hardware disc drives are attached to the controller interfaced by a software driver, and each drive has one or more cartridges or other physical storage media. Each such cartridge, fixed platter, etc., is one Physical Unit. A Physical Unit is usually designated to be one logical partition, but it may be subdivided into two or more partitions. In some cases it is necessary to partition a large Physical Unit into smaller parts so that the Real Disc Address does not exceed sixteen bits (the Real Disc Address for Logical Unit zero must not exceed fifteen bits). Although IRIS will operate with two or more small Physical Units combined into one logical partition, this is not recommended because a human error could result in the union of two cartridges that are not a true pair; eg, not part of the same Logical Unit, or from backups at different dates. An attempt to install such a mismatched set would likely result in the loss of many if not all files on the discs.

Each entry in the disc driver table starts with four words as follows:

word 0:	number of disc drivers in driver table
word 1:	address of LUFIX (system disc driver) in CONFIG file
word 2:	address of RBLK entry to BZUP disc driver in CONFIG
word 3:	n = number of partitions on this driver

SIR replaces the value in word 0 with a pointer to the driver's actual location in core; ie, the LUFIX pointer. The table is terminated by a -1 (octal 177777) where word 0 of the next table entry would be.

The next two words point to the proper system driver and BZUP driver in the CONFIG file, and the forth word must designate the number of physical partitions handled by this set of drivers. The word containing n is immediately followed by n partition entries, and each such entry consists of eight words as follows:

word 0:	0	(SIR puts LUVAR pointer here)
word 1:	p,g,u	Min priv,group,user to install
word 2:	g,u	Max group,user to install
word 3:	NC	Number of cylinders
word 4:	PART	Partitioning information
word 5:	PART1	Partitioning information
word 6:	m	Minimum block count
word 7:		(not currently used)

Word 1 has the same form as an account number. The top two bits specify the minimum privilege level any user must have to be allowed to install any Logical Unit in this partition. The remaining fourteen bits give the first group/user number that will be allowed to install, and word 2 gives the last group/user that will be allowed to install. Any user whose account number is within this range or who has at least the minimum privilege specified will be allowed to install a Logical Unit in this partition.

Word 3 specifies the number of disc cylinders in this physical partition, and words 4 and 5 are the partitioning constants for the driver. Word 6 specifies the minimum number of available blocks on a Logical Unit in this partition to allow building a new file.

```
**************************************************
*                     CAUTION                    *
*   The configuration of physical partition 0.0  *
*   must be such that a Real Disc Address on     *
*   Logical Unit zero can not exceed 15 bits.    *
**************************************************
```

The information for Logical Unit zero (partition 0.0) is supplied by the SysGen Overlay (SOV) tape and is repeated in CONFIG for reference only. Words 1, 2, and 6 of Logical Unit zero's information may be modified in the REX file at the locations shown on the SOV listing.

2.7 How to Move Logical Unit Zero

In some cases it becomes desirable to move Logical Unit zero (the system disc) from one Physical Unit to another. One example is when a new IRIS system is received on a cartridge and is configured to operate from the removable cartridge, and it is desirable to move it to the fixed disc. To move the system from one Physical Unit to another, follow these steps carefully in the order given:

1) Use DDCOPY (see Section 8.5) to copy the system disc to what is to be Physical Unit 0.0.

2) Do an IPL from new Physical Unit 0.0 (see Section 5.1) with switch one up to get into BZUP. Set the switches to 025000 octal, and press the colon (:) key (or press CONTINUE if the computer was halted) to start an IPL and give control to DBUG.

3) Use DBUG to set the partitioning constants in the BZUP driver to reference new Physical Unit 0.0. The command sequence is:

```
24300:part   Enter value of PART
24301:part1  Enter value of PART1
W0,24000     Write BZUP on disc
```

4) Locate the partitioning constants in the system LUVAR table in the Sysgen Overlay (SOV) listing, and use DBUG to enter the proper values in core. If the SOV listing is not available, find the system LUVAR table by dumping the address in location 576; the partitioning constants are at that address plus 1 and plus 2.

5) Set switch one up, and give DBUG a J14000 command to initialize the system with a minimum configuration.

6) Use DSP (see Section 3.3) to modify the SOV partitioning constants in the "REX" file the same as in step 4 above.

7) Use DSP to modify all partitioning constants in the disc driver table in CONFIG as necessary (see Section 2.6).

8) Do a normal IPL to bring up the system, and install other Logical Units as desired.

9) Back up the new system disc once it is determined that it is working properly.

2.8 How to Change Business BASIC's Program Area

The system's response time must be considered when increasing the maximum Business BASIC program size. Responses will be degraded if there are not enough partitions to keep swapping to a minimum, especially if an active file larger than one cylinder is required to hold a maximum size program. On most 10 megabyte (or smaller) discs, each cylinder holds 24 blocks. One block is used for the header, leaving 23 blocks, which will handle a 5470 word program size. A partition size of octal 13400 words is required for a 5470 word program, and at least 24K words of core are required for an effeciently running system with one 13400 word partition. See page 2-16 for the method of determining the maximum partition size for a given configuration. It is recommended that a partition size larger than 12400 words be used only if the application programs can not be divided into segments of 4958 words or less. Many application programs supplied by EDSI will operate in a 4958 word program area, thus allowing their use on a system with only 16K words of core, a 12400 word partition size, and a 22 block active file. To change the partition size:

1) Make a backup copy of the system disc.

2) Determine the amount of free space in lower core by subtracting the minimum number of free nodes (at NNOD in the CONFIG file) from the actual number of free nodes (at NNOD in core) and multiplying the result by 12 octal. Examine NBBP in the INFO table in core to determine the number of buffers actually in the buffer pool. If this number (octal) seems excessive (more than ten times the number of interactive ports) then multiply the excess by 260 decimal and add this to the free node space. Divide this total by the number of lower core partitions to determine the maximum amount that the partition size may be increased. See NNOD and NBBP definitions on page A5-2.

3) Select a suitable partition size (PSIZ) from the table in Figure 2.2. Once an acceptable value is selected, enter it in the PSIZ cell in the CONFIG file (see page 2-10), and do an IPL to make this value effective.

4) Enter BASIC, and type SIZE to check the program size. Return to step 2 above if the desired size has not been achieved.

5) Calculate the value of PSIZ/400. This is the required active file size (number of blocks, including the header) for a maximum size program. Bear in mind the comments at the beginning of this Section.

6) Check the default active file sizes in each interactive port driver's port definition table (see Appendix 6). If any active file sizes must be changed then do another IPL to make the new values effective.

7) Back up the newly configured system.

Figure 2.2: Partition Size Selection Table

PSIZ	# partitions in upper core		BASIC Program
(octal)	in 48K total	in 64K total	size (decimal)
2000	16	32	606
2400	12	25	862
3000	10	21	1118
3400	9	18	1374
4000	8	16	1630
4400	7	14	1886
5000	6	12	2142
5400	5	11	2398
6000	5*	10*	2654
6400	4	9	2910
7000	4	9*	3166
7400	4	8	3422
10000	4*	8*	3678
10400	3	7	3934
11000	3	7*	4190
11400	3	6	4446
12000	3	6	4702
12400	3*	6*	4958
13000	2	5	5214
13400	2	5	5470
14000	2	5	5726
14400	2	5*	5982
15000	2	4	6238
15400	2	4	6494
16000	2	4	6750
16400	2	4	7006
17000	2	4	7262
17400	2	4	7518
20000	2*	4*	7774
20400	1	3	8030
21000	1	3	8286
21400	1	3	8542
22000	1	3	8798
22400	1	3	9054
23000	1	3	9310
23400	1	3	9566
24000	1	3	9822
24400	1	3	10078
25000	1	3*	10334
26000	1	2	10846
27000	1	2	11358
30000	1	2	11870
31000	1	2	12382
32000	1	2	12894
33000	1	2	13406
34000	1	2	13918
35000	1	2	14430
36000	1	2	14942
37000	1	2	15454
40000	1	2	15966

*Partition sizes marked with an asterisk are recommended for optimum use of upper core for user partitions but all sizes are practical because all remaining core will be used for the buffer pool. The odd multiples of 400 octal are not shown for PSIZ > 25000, but all are allowable.

2.9 How to Cause a DISCSUB to be Core-Resident

Selected DISCSUBS are brought into core during an IPL by SIR, as specified by a list of DISCSUB numbers starting at location 1000 in the CONFIG file. The list is terminated by any value exceeding 777 octal. To cause a DISCSUB to be core-resident, use DSP to modify CONFIG. Suppose you wish to cause the square root function to be core-resident:

```
   #DSP EkeyE CONFIG
   D1000
   1000:    1    3    15    30    177777    [escape]
   E1004
   1004:    52   [number of PSQRF subroutine]
   1005:    177777
   1006:    [escape]
```

Start an IPL, and SIR will attempt to load all core-resident DISCSUBS. SIR will print a trap message if a core overflow occurs, in which case a minimum IPL is done automatically to bring up the system with only the master terminal active. DSP may then be used to modify the core-resident DISCSUB list or to make other changes to make room for the needed subroutines.

See page 2-15 for a list of DISCSUBS which are commonly made core-resident. A list of all DISCSUB numbers may be found in the Software Definitions listing or in Appendix 7 of this manual. Only the lower three digits of a DISCSUB number must be entered in the list in CONFIG; eg, 20 for BUILD, not 40020. DISCSUBS that are flagged "included with another if core-resident" must not themselves be entered in the core-resident list. LBSA must be at least 31400 to allow any DISCSUBS to become core-resident (see Section 2.4).

2.10 How to Add Devices to the System

PLOAD may be called to load a device driver. The driver must conform to the driver specifications described in Section 11, and must be loaded as a $Filename (the first character of the Filename must be a dollar sign). File type 77001 is used for a system subroutine, system device driver, or interactive port driver such as $DEC, $DAU, $MTAS, $RTC, $TTY, $MMUX, $MTI, $ALU, $DGMX, $BBMX, or any $TERM. File type 00036 is used for a peripheral device driver such as $MTAO, $LPT, $CRD, $PLT, $PTP, $PTR, or $PTM.

After loading the tapes, an IPL is required for SIR to load the drivers into core and link them to the system. If there is not enough available core then a core overflow will occur, in which case a minimum IPL will be done, and the manager must restructure core allocation in the CONFIG file or delete the driver loaded (see Section 4.6). The CHANGE command may also be used to change a driver's Filename, dropping the dollar sign so that SIR will not bring it into core.

2.11 Log-On Messages

In some instances, it is desirable to type additional messages when each user logs on, such as notices about programming seminars or changing account ID's, or a notice that the system will be down at a certain time for periodic maintenance or for use in stand-alone mode. To do so, create a data file with the name LOGONMSG, and format it as a single string of 75 characters on Logical Unit zero. All strings entered in this file will be printed after each user logs on. Although the user may suppress all or part of his account status type out, these log-on messages can not be suppressed. A null string will terminate the output of messages. The following Business BASIC program may be used to create such a file:

```
10 DIM A$(75)
20 BUILD #1,"<33> LOGONMSG!"
30 INPUT "\215\? "A$
40 WRITE #1,R;A$
50 LET R=R+1
60 IF LEN(A$)>0 GOTO 30
70 CLOSE #1
```

The same program may be used later with line 20 changed to

```
20 OPEN #1,"LOGONMSG"
```

to enter new log on messages. Also, line 30 could be changed to

```
30 INPUT "\215\LINE#, LINE? "R,A$
```

to allow changing specific lines of the log on messages. Also, see Paragraph 2.3.15 for changing the initial welcome message.

2.12 Log-On Charges

If it is desired to assess a fixed charge each time a user logs on, then change line 20 in the above program to

```
20 BUILD #1,"$dd.cc <33> LOGONMSG!"
```

where dd.cc is the desired charge. CHANGE may also be used at any time to change the cost of the LOGONMSG file. Since the LOGONMSG file is opened for the user by BYE each time a user logs on, the user's account is charges its cost for that access.

2.13 Log-On Restrictions

Selected users (identified by account group-user number) may be restricted to log on only on certain ports and/or during certain times of day. This is controlled by a table starting at location 16000 (octal) in CONFIG. This block of CONFIG does not normally exist, so before entering any log-on restrictions it is necessary to allocate and zero out this block by giving DSP the commands:

```
F CONFIG
A16000
K16000,16377,0
```

The log-on restrictions table has four words per entry as follows:

Word 0 has an account number in the lower 14 bits (group-user number). The top 2 bits define a mode as follows:

00 Entry applies only to the group-user number given.

01 Entry applies to all users in given group with user number greater than the user number given.

10 Entry applies to all account numbers greater than the account number given as a 14-bit number (ie, group = G and user ≥ U, or group > G, where G-U is the group-user number given).

11 Same as mode 10, but if match is found and it does not allow log-on, then continue scanning table. In all other cases, scan stops with first match.

Any restrictions on a user are determined by the first table entry where a match occurs: if no match is found then there are no restrictions on the particular user. When a match is found, then words 1 through 3 are used as follows:

Word 1 has the form nnnppp in octal, where nnn ≤ 177 (octal). Indicates that any account selected by word 0 may log on only on ports ppp, ppp+1, . . . ppp+nnn. Note: nnn = 177 ==> all ports ≥ ppp.

Words 2 and 3 each have the form 00aabb, where aa<bb and bb≤60 (octal), and each of aa and bb is a half-hour since midnight (in octal). Any account selected by word 0 may log on only if the current time t is in the range aa ≤ t < bb. The two words allow two time ranges for each day.

The table is terminated by a zero where word 0 of the next entry would be, unless the block is full (64 entries) in which case a terminating zero word is not used. The value 100000 (octal) in word 0 of an entry means "any account", and 177000 (octal) in word 1 means "any port".

Copyright 1978
Educational Data Systems

For word 2 or 3, the time of day values are as follows:

Time	aa or bb	Time	aa or bb
00:00	00	12:00	30
00:30	01	12:30	31
01:00	02	13:00	32
01:30	03	13:30	33
02:00	04	14:00	34
02:30	05	14:30	35
03:00	06	15:00	36
03:30	07	15:30	37
04:00	10	16:00	40
04:30	11	16:30	41
05:00	12	17:00	42
05:30	13	17:30	43
06:00	14	18:00	44
06:30	15	18:30	45
07:00	16	19:00	46
07:30	17	19:30	47
08:00	20	20:00	50
08:30	21	20:30	51
09:00	22	21:00	52
09:30	23	21:30	53
10:00	24	22:00	54
10:30	25	22:30	55
11:00	26	23:00	56
11:30	27	23:30	57
12:00	30	24:00	60

The value 000060 (octal) in word 2 means "any time of day", and word 3 will be ignored. Words 2 or 3 are ignored if word 1 indicates that the user is not on an allowable port.

The table is terminated by a zero where word 0 of the next entry would be, unless the block is full (64 entries) in which case a terminating zero word is not used. The value 100000 (octal) in word 0 of an entry means "any account", and 177000 (octal) in word 1 means "any port". For example, the entries

```
000314
002004
004160
000016
041140
177000
002030
003242
000000
```

would allow users to log on to account group 3 user 12 (decimal) only on ports 4 through 6 and only between 4:30 PM and 7:00 AM, and it would allow users to log on to any account in group 9 with user number ≥32 to log on to any port but only during the hours 8:00 AM to noon or 1:00 PM to 5:00 PM.

2.14 Automatic Program Start

Selected users (indentified by account group-user number) may have a specified BASIC program started automatically after log-on, and the program to be started can be dependent on which port the user is on. This is controlled by a table starting at 16400 (octal) in CONFIG. This table may be up to three blocks long (16400 through 17777 octal), but the blocks do not normally exist. The DSP commands to allocate these blocks are similar to those for the log-on restrictions block (see above). Each block of the table holds up to 16 entries of 16 (octal 20) words each. Words 0 and 1 have the same form as in the log-on restrictions table; if these words indicate that a selected user is on a selected port then words 2-11 are assumed to be a BASIC program Filename string, and that program is started running. The string may be in the form lu/Filename; otherwise, the user's assigned Logical Unit is assumed. For example, if DSP were used to make the entries

```
F CONFIG
A 16400
K16400,16777,0

16400:000201
16401:000000
I16402:MENU

16420:140000
16421:001003
I16422:INVENTORY

16440:0
```

would cause the program MENU to be automatically started if user 2,1 logs on to port zero, and it would cause the program INVENTORY to be started if any user logs on to port 3 or 4. The program is started running on the port where the user logs on as soon as all log on information and messages have been printed. See Section 2.3.14 if it is desired to suppress some or all of the accounting information at log on time.

3. DEBUGGING AND SERVICE ROUTINES

Three utility programs are available for debugging and patching IRIS. BZUP is a small (one disc block) utility program which includes the initial portion of the IPL sequence. DBUG is included in the REX disc file for use in debugging core-resident code such as SIR, discsubs, drivers, and the interrupt service routines. DSP is the most used debugger, since it is a processor which can be used from any interactive port while the system in in normal use.

3.1 How to Use BZUP

The Block Zero Utility Package (BZUP) is a debug package that occupies a single block at disc address zero on Logical Unit zero. When in core, BZUP currently occupies locations 24000 through 24377 and uses locations 24400 through 24777 as a disc block buffer area if disc transfers are performed. In addition, BZUP contains the Initial Program Load start-up sequence for REX. The disc driver in BZUP is also used by DBUG, SYSL, INSTALL, CONVERT, CLEANUP, and SHUTDOWN. BZUP is loaded initially during the SysGen procedure, and it may be reloaded at any time (see Section 4.2).

Some disc controllers require a software driver that exceeds the available space in BZUP; therefore, the BZUP for such a device must be stripped down to become a disc driver only, and it will not respond to any of the commands on the next page. Disc controllers currently known to be in this category include:

Minicomputer Technology TDC-802
Diva DD-14 and 4231
Data General 4046 and 4234
Xebec XDF-70
Ball 3300

To get BZUP from the disc, initiate an IPL as described in Section 5.1 or 5.2, except set switch 1 up before pressing the START switch (the IPL sequence will be executed if switch 1 is down). A space will be printed if BZUP actually contains the debug facilities, or the computer will halt if it is a stripped down version; if it halts, CONTINUE may be pressed to resume the IPL sequence. Note: switch 1 should usually be down before resuming the IPL sequence because having it up causes SIR to initialize the system for minimum configuration (see page 5-2).

BZUP contains a simple disc driver for one particular disc controller, thus necessitating a different BZUP object tape for each different disc controller. The partitioning constants at locations 300 and 301 in BZUP (locations 24300 and 24301 in core) determine to which disc drive and platter the Real Disc Addresses will refer. The form of the partitioning constant depends on the driver itself and is usually documented along with the code for the driver in the BZUP or CONFIG listing. See Section 11.9.

Each command to BZUP consists of a single letter or the RETURN key, and the command character may be preceded by an octal parameter as shown below.

Command	Description
a:	Open cell at address a.
a/	Display and open cell at address a.
lf	Display and open next cell. An error will be indicated if no cell is open.
n lf	Store number n in open cell, then display and open next cell. Error if no cell is open.
cr	Return carriage (no operation).
n cr	Store number n in open cell, then open next cell. Error if no cell is open.
aC	Copy cells a through a+377 into disc buffer area.
aM	Move the contents of the disc buffer area into locations a through a+377.
R	Set up for addresses to refer to real core.
dR	Read disc block d into disc buffer area, and set up for addresses 0 through 377 to refer to this block.
W	Write block in disc buffer back where it came from on disc (d of last dR or dW command).
dW	Write block in disc buffer onto disc at address d.
:	Start IPL sequence and bring up IRIS; or, if the switches are set to the starting address of DBUG (x25000), then control will be transferred to DBUG after the REX file is loaded into core, rather than to SIR for system initialization.

where: a is any core address (octal 177776 maximum)
d is any Real Disc Address
n is any octal number
lf is a LINE FEED
cr is a Carriage RETURN

BZUP acknowledges execution of a legal command by printing a space. Illegal commands cause a question mark to be printed. If a disc read or write error occurs, the disc status word is printed followed by a question mark. A command may be aborted by the user by pressing the ESC key before pressing the command character.

3.2 How to Use DBUG

DBUG is a complete position-independent debug package for any Nova-type computer. There is a copy of DBUG in the REX disc file. To use DBUG, do an IPL with switch 1 up to enter BZUP, then set the data switches to the DBUG starting address (currently 25000), and type a colon. After REX, SIR, and DBUG are loaded into core, control will be transferred to DBUG instead of to SIR. DBUG occupies octal 3000 words of core.

All operations can be performed from the master terminal, including transfer of control to a user's program and back to DBUG. All operations are executed by typing one letter followed by octal parameters as required (except ":" which is also preceded by an octal parameter) and ending with a RETURN (see also "Alternate Terminators" on page 3-10). The following functions are provided; the number in parentheses is the number of parameters required for that particular function.

A Type accumulators, carry flip-flop, and ION. (0)
B Insert or remove Breakpoint(s) in user program. (0, 1 or 2)
C Change accumulator, carry flip-flop, or ION. (2)
D Dump (octal, word or byte). (1 or 2)
E Enter octal or symbolic. (1 or 2)
F Set up an address offset. (0, 1 or 2)
G Get (read) a block from disc. (2)
H Halt with accumulators, carry, and ION restored. (0, 1 or 2)
I Input ASCII string on master terminal. (1 or 2, plus text)
J Jump with accumulators, carry, and ION restored. (0, 1 or 2)
K Store a constant in a block of core. (3)
L List program (octal and symbolic). (1 or 2)
M Move block in core. (3)
N Search memory for not-equal (with mask). (3 or 4)
O Output ASCII string on master terminal. (1 or 2)
P Punch paper tape on TTY or PTP. (0, 1 or 2)
Q Query location of breakpoints. (0)
R Read paper tape from TTY or PTR. (0 or 1)
S Search memory for constant (with mask). (3 or 4)
T Trace through user program. (1 or 2)
U (not used)
V Verify paper tape (TTY or PTR). (0 or 1)
W Write a block on disc. (2)
X Compute a checksum for a section of core. (2)
Y (not used)
Z Search for relative address reference. (1)
: Enter one value (octal or symbolic). (1 plus value)

These functions are described in detail on the following pages. All parameters must be entered in octal. The letters x, y, z, a, m, and n are used on the following pages to represent octal parameters. Press the RETURN key after entering any command.

RE-ENTRY TO DBUG:

To re-enter DBUG manually, RESET and START at 25000 or 25001. DBUG's normal starting address is 25000, which saves the CPU status; to preserve the previously saved CPU status, start at 25001 (this will also permit a return to a previous breakpoint via H, J, or T command). The use of some processors, however, will destroy the core copy of DBUG.

DBUG may also be brought into core, either by itself or with a stand-alone file, by use of SHUTDOWN (see page 8-6). BZUP is not brought into core by SHUTDOWN, however, so the G and W commands in DBUG can not be used in this case.

ADDRESSING MODES:

For many commands, DBUG allows either word or byte addressing, using either real core addresses or "offset" (virtual) core addresses based on an offset that has been previously entered (by an F command). DBUG also is designed to allow addressing up to 64K words of core memory. This is accomplished by having two word addressing modes (real and virtual), and three byte addressing modes (virtual plus two real modes: lower 32K and upper 32K). These modes are invoked by the optional second parameter "a" shown for commands D, E, H, I, J, L, and O (except that H and J do not permit byte addresses).

a	Meaning
omitted	word address, including offset
0	word address, absolute
1	byte address, using offset
2	byte address, lower 32K absolute
3	byte address, upper 32K absolute

For those commands where no "a" parameter is shown, the addressing mode (if any) is the same as "a omitted" above; ie, word address including offset (if any).

WARNING: If using the G and W commands, BZUP must be in core. Both DBUG and BZUP are position independent, but BZUP must be 1000 words (octal) ahead of DBUG to supply the disc driver for the G and W commands. A backslash is printed if a read or write error occurs, or if DBUG is able to determine that BZUP is not present. The partitioning constants in BZUP determine which Physical Unit is to be used.

CAUTION: If using the Iomec disc, do not use the G command to read a disc block into page zero of core, since cell 20 is used by the disc controller as a word counter. Exceptions: BZUP and page zero of REX each contain -357 octal in cell 20; therefore, these blocks can be read into core in page zero without disturbing the word count.

DETAILED DESCRIPTION OF COMMANDS FOR DBUG:

A

Type the contents of registers A0, A1, A2, A3, the carry flip-flop, and interrupt status as they were at the time DBUG was entered. The interrupt status is typed as an E for enabled or D for disabled. If DBUG was entered from a breakpoint, the type-out is preceded by that breakpoint location and a colon.

Bx,n

Insert breakpoint #n (n = 0 or 1; 0 assumed if n omitted; see below for larger n) in the user program at address x. If a previous breakpoint #n had been established (and was not in the meantime modified), it is restored to its original state before this new breakpoint is inserted. The breakpoint itself is a JMP @17 (for breakpoint #0) instruction, and DBUG puts a pointer to its breakpoint routine in location 17 octal. For breakpoint #1, location 16 is used. If control later reaches address x, then x is typed followed by a type-out of the registers, carry flip-flop, and interrupt status as in A above. Each breakpoint requires its own page zero cell. If enough such cells are available, up to four breakpoints may be used (numbered 0 through 3). To create additional breakpoints or change their page zero cells, simply insert the desired page zero addresses at locations 10, 12, 14, or 16 relative to the beginning of DBUG. A zero at any of these locations marks the end of the breakpoint list. DBUG itself can be used to do this; then use Q to confirm the new values.

BO,n

Remove breakpoint #n (#0 if n is omitted), restoring the instruction at that location. Note that a breakpoint cannot be put at location zero.

B

Remove all breakpoints that have been established.

Cx,y

Change accumulator, carry flip-flop, or interrupt status. If x is 0, 1, 2, or 3, then y is stored as the saved value for accumulator x. If x is 4, then the saved value of the carry flip-flop is set to 0 or 1 depending on whether y is 0 or not. If x is 5, the interrupt enable status (ION) is set to 0 (disabled) or 1 (enabled) depending on whether y is 0 or not.

Dx,a

Dump memory in octal, beginning at location x, using addressing mode a. Eight words (or bytes if a byte address mode is used) are typed per line, with the address of the first word (byte) at the beginning of each line.

Ex,a — Enable entry at address x, using address mode a. The address (changed to a word address if it was a byte address) is printed, followed by a colon; a value (octal or symbolic) may then be entered, followed by a RETURN. The next address (x+1) will then be printed and opened for entry, and entry continues into sequential cells until ESC is pressed to terminate entry. Relative addresses may be entered either as .±n or as absolute. Absolute addresses less than 400 (octal) are interpreted as page zero rather than relative. DBUG understands all standard assembler symbols and the arithmetic skips (SGR, SGE, SLS, SLE, SEQ, SNE, SKZ, SNZ, SSP, SSN, SGZ, SZN, SKE, and SKO), in addition to the following special CPU instructions:

```
IOR (62677 = IORST)        RDS n (DIA n,CPU = READS n)
HLT (63077 = HALT)         ITA n (DIB n,CPU = INTA n)
IEN (60177 = INTEN)        MSK n (DOB n,CPU = MSKO n)
IDS (60277 = INTDS)
```

Note: If no entry is typed in before the RETURN, the present content of the opened location is typed out in octal and symbolic form to allow examining the cell before entering. If another RETURN is then typed, again without an entry, the next address will be printed and opened for entry. If "↑" is typed instead of RETURN, the previous address is typed and opened. If "/" is typed instead of RETURN, the same address is typed and opened. This last feature is convenient for confirming an entry just typed in, and for examining it in both octal and symbolic form.

Fx,y — Establish an address offset; ie, a fixed difference between real absolute core addresses and virtual addresses as entered and listed in DBUG. The difference x-y (where x is the real core address and y is the virtual address on the listing) is added to each address entered and subtracted from each address printed. If y is not entered then x is used as the offset. An F is printed at the beginning of each line whenever a non-zero offset is in effect. Type FO to revert to direct core addressing.

F — Save the current offset value, and reinstate previous offset that was in effect before the current one was established. Types the offset being reinstated. This allows the user to alternate between two different offsets (or between one offset and real core).

The offset can be used to convert a real address to virtual or vice versa. For example, while in virtual mode, a temp cell containing a return address (which is real) is dumped. To convert this to virtual, put a breakpoint at that address, then revert to real mode by typing F, and query the breakpoint be typing O. The address will be printed in real form. Return to virtual mode by typing F again, and get rid of the breakpoint by typing B.

Gx,y

Get a block from disc. Real Disc Address x is read into core locations y through y+377. Gx will read into page zero, and G will read block zero (BZUP) into page zero. See WARNING on page 3-4! A backslash "\" and the disc controller status word indicating the error will be printed if any disc error is detected.

Hx,a

Halt with registers and carry restored. The instructions after the halt will restore the interrupt status and then execute a jump to location x, using word addressing mode a. INST STEP may then be used to step through the user's program.

H

Same as Hx,a except will return to the breakpoint from which DBUG was entered. See "J" below.

Ix,a
text

Input ASCII. Characters following the RETURN are stored in memory, using address mode a, starting at location x, two characters per word, (similar to .TXTF pseudo-op in the assembler with left-right packing). Input is terminated by pressing ESC, which causes a zero byte or word to be stored.

Jx,a

Jump to location x, (using word addressing mode a) with registers, carry, and interrupt status restored. Same as Hx except that it does not halt before jumping.

J

Return to user program at the breakpoint from which DBUG was entered, after restoring accumulators, carry, and interrupt status. Does not remove the breakpoint. May be used after setting a new breakpoint (same # or different), in which case control is still passed to the old breakpoint location from which DBUG was entered. Types a backslash if DBUG was not entered from a breakpoint.

Kx,y,z

Store the octal constant z in absolute locations x through y, inclusive.

Lx,a

List program, both octal and symbolic, starting at location x, using address mode a. To terminate listing, press ESC. Relative addresses more than 7 away are listed as absolute.

Mx,y,z

Move block in core. Absolute locations x through y, inclusive, are moved to the area starting at location z. The source and destination areas may overlap in either direction without bad effects. May be used to move DBUG as long as the destination area does not overlap the source area.

Nx,y,z,m

Search for not-equal. Same as Sx,y,z,m except that it searches for a not-equal condition.

Ox,a

Output ASCII. The contents of core starting at location x (using address mode a) are typed out as text, two characters per word. Output is terminated if a zero byte is encountered.

Px,y

Punch paper tape from core locations x through y, inclusive. Will punch on high-speed punch (device code 13) if available and turned on, else punches on TTY (device code 11). To punch on the TTY, type in the command up to but not including the carriage return, then turn on the punch, and press return. When the punching is complete, turn off the punch before entering the next command. Punches about 2 feet of leader before the data if this is the first P command since DBUG was started or since an end block plus trailer were punched.

Px

Punch an end block with starting address x, followed by about 2 feet of trailer.

P

Punch an end block without starting address, followed by 2 feet of trailer.

Q

Query breakpoints. Types the page zero cell corresponding to each available breakpoint and the core address (if any) where that breakpoint is currently set.

Rx

Read punched paper tape, from the master Teletype if x is omitted or is zero, or from the high-speed paper tape reader (device code 12) if x=1. If a checksum error occurs, or if an attempt is made to write into non-existent core or to overwrite DBUG itself, further reading is stopped, and the address where the error occurred is typed out. If the tape contains an end block with a starting address, the computer will halt with the starting address in A2. If CONTINUE is then pressed, it will jump to the starting address.

Sx,y,z,m

Search locations x through y, inclusive, for the constant z. Each word is first ANDed with mask m before comparison with z. If m is not entered, it is assumed to be 177777; ie, a search is made for an exact match with z. The use of the mask is best explained by an example: the command Sx,y,60025,160077 will search locations x through y for any I/O instruction for device 25. When a comparison is found, its address and contents are typed in both octal and symbolic forms.

Tx

Trace through user program for x steps, beginning where the last breakpoint was encountered or where a previous trace left off, whichever occurred last. Types a backslash if no such starting point exists. If x = 0 or 177777, traces forever. If x is omitted, traces one step. To start tracing at a certain location, first put a breakpoint at that location, then jump there using the J command, which will of course immediately hit the breakpoint, and finally give the desired T command. For every program step that is traced, types the core address, the instruction in symbolic form, and the number and new content of any accumulator which was involved in this instruction. Note: trace works by pushing breakpoint #0 ahead of itself. Therefore breakpoint #0 is not independently available when using T.

Tx,y

Same as Tx except suppresses intermediate type-out unless location y is written into by the instruction being traced; ie, the instruction is a STA, ISZ, or DSZ, and it addresses location y, regardless of addressing mode. Can be used in the form T0,y to determine if location y is ever written into by the user program.

Vx

Verify paper tape from TTY (x = 0 or none) or PTR (x = 1). If a verification error is found, its address is typed out.

Wx,y

Write a block on disc. Locations y through y+377 are written at Real Disc Address x. Wx will write page zero on the disc. See WARNING on page 3-4! A backslash "\" and the disc controller status word will be printed if any disc error is detected.

Xx,y

Compute and type a "rotating" checksum over core locations x through y. The checksum is produced with an ADDL instruction in order to detect a change if two words in core are swapped. Useful for testing if a change has occurred anywhere in a section of core.

Zx

Search for relative addressing reference. The 256 words centered on location x (using the "a omitted" addressing mode) are searched for any memory reference instruction that references location x using relative addressing. Any such instruction is listed in octal and symbolic form.

x:value

Enter octal or symbolic. The value given (either octal or symbolic) is stored at location x, using the "a omitted" addressing mode. If "value" is omitted, types the present contents of location x followed by a colon, after which a new value may be entered. See the "E" command on page 3-6 for more information.

REMARKS:

The carry light flashes while DBUG is waiting for an input character to be entered (except in I mode). This is a signal that DBUG is active and will respond to input.

If an error is made while entering control information, three choices are available for correcting it.

1. Press ESC (or CTRL D or ALT MODE) to delete the type-in and enable a new type-in.

2. Press CTRL A (or CTRL H or RUBOUT) to backspace the last character typed in, as in IRIS. CTRL H echoes itself; the other two echo the character being deleted.

3. If the error was in entering an octal value (not part of a symbolic instruction), type a few zeroes followed by the correct octal number, as DBUG only uses the last six octal digits typed in for an octal word.

DBUG normally occupies core locations 25000 through 27777. However, DBUG may be moved at any time by use of its Move instruction (even into upper 32K in a 64K system). After moving, the P command may be used to punch a tape of DBUG for the new location if desired. DBUG cannot punch itself in its own location, however, because it changes certain cells in core between the time it punches the checksum and the time it punches the data, thus producing a checksum error.

If it is desired to use DBUG with the EDSI Mighty-Mux (TTY option) at a Baud rate other than 110 Baud, enter the desired PCB and PCON in words 2 and 3 relative to the beginning of DBUG. PCB is the port control block address to be used for setting up the Mighty-Mux, and PCON is the port control word for the desired Baud rate and parity mode (eg., 50057 for 9600 Baud, even parity). To disable Mighty-Mux setup, put a 0 in word 2 of DBUG.

A return delay may be caused by entering a non-zero value in word 4 of DBUG. Smaller values cause longer delays.

ALTERNATE TERMINATORS:

Instead of RETURN, two other terminators may be used in certain cases, as already described under E above.

1. Slash (/) allows putting multiple commands on one line, as it converts the usual RETURN into a slash. (Do not use with L, N, S, or Z, as it will not increment the operand address.) Example:

```
B1234/ B1400,1/ J1234/
```

2. Up-arrow (↑) causes addresses to count backwards for the E, L, N, S, and Z commands. Can be used with S or N to find the last location below a given point where the search conditions are met.

Copyright 1978
Educational Data Systems

3.3 How to Use DSP

The Disc Service Processor (DSP) is an on-line interactive utility package for use in servicing and debugging processors and other files under IRIS. Any location in core or any file on the disc can be easily examined and modified by use of DSP; therefore, the use of DSP is restricted to the system manager and EDS personnel. To initiate use of DSP, log on to the manager account at any terminal, and enter the system command

```
          DSP Ekey
```

or

```
          DSP EkeyE Filename
```

where "key" is the password assigned to DSP (see Section 4.9), and Filename is the name of a file to be selected (see "F" command on page 3-13). After initially entering DSP in this manner, it may be re-entered from the same terminal without the password. To prevent unauthorized persons from using DSP, be sure to exit with an X command, and always log off when leaving a terminal.

Each time DSP is entered, the identity of the currently selected file (or core) is printed, and any of the commands described on the following pages may be given. DSP may operate on core or on any disc file as "virtual core". Press the RETURN key after typing any command. All commands which examine and/or modify memory operate on either real core (assumed on initial entry) or on the file or disc block last selected by an F, G, or H command.

REMARKS:

Any address x may be given as a byte address by appending a hyphen to the address. For example, D3025- will dump bytes starting with the right-hand byte of word address 1412, and E17000- will allow entry of bytes starting at the left hand byte of word address 7400. No value may exceed 377 octal when using byte addressing. If a byte address is given when a driver file is selected, then that byte address in real core is referenced; this eliminates the need to select real core to examine the driver's buffers.

The core-resident copy of a discsub or driver may be addressed by appending an apostrophe to any address x, where x is the virtual address (as shown on the listing). DSP does all address conversions, and the user may examine or modify the core-resident copy as if it were at the location shown on the assembly listing.

DSP will accept lower case command letters in place of the upper case letters shown on the following pages, except that the "N" in the "LxN" command must be entered as upper case.

DETAILED DESCRIPTION OF CAMMANDS FOR DSP:

x:v

Insert the value v at address x. Very useful for entering into a single memory location. The value v may be either a symbolic instruction or an octal number. See the "E" command for more information.

Ax

Append the block which will contain address x to the file selected by the last F command. The first core address and the Real Disc Address of the appended block will be typed out. The block is filled with 077377 halt instructions.

Bx

Insert a breakpoint at address x. This command is meaningful only if the specified file is a processor. If that processor is then used on the same port, and the breakpoint is encountered, then control will revert to DSP, and a print out of the registers and carry flip-flop will occur. The breakpoint is cleared when it is encountered, and it is also cleared by any F, G, H, or X command.

Bx,[conditions]

Insert a conditional breakpoint at address x. A breakpoint may be conditional on a register containing a specified value (indicated by "Ar=v", where r is a register number 0 to 3, and v is an octal value), and/or conditional on a memory cell containing a specified value (indicated by "x=v", where x is a memory address), and/or the breakpoint may be activated only after executing the instruction at the breakpoint location a specified number of times (indicated by an octal value by itself). For example:

```
B7235,A1=260,225=16003,4
```

will breakpoint the fourth time that location 7235 is reached with the value 260 in register A1 and the value 16003 in memory location 225. The conditions may be given in any order, and the memory location may be specified indirectly: eg, @37422=177723 means that the contents of location 37422 is used as a pointer to a cell that is to be checked for the value 177723. Note: the BREAK discsub must be core-resident before using conditional breakpoints.

C command

The "command" given is passed on to SCOPE as a system command. This is equivalent to pressing CTRL C and then entering the same command.

Dx

Dump octal starting at address x. The contents of storage starting at location x are printed in octal, eight words per line. The address of the first word of the line is printed at the beginning of each line. Listing may be terminated by pressing ESC.

Ex
Enter octal or symbolic starting at address x. Each entry must be followed by a RETURN. Press ESC to terminate entry mode. Machine instructions may be entered in symbolic form, but device addresses must be given in octal rather than using the device name in I/O instructions (eg, 10 rather that TTI). Labels may not be used, but absolute addresses will be converted to relative if possible.

F Filename
Select the file identified by Filename to be examined and/or modified by other commands. Logical Unit zero is assumed unless given in the form u/Filename, where u is the Logical Unit number in decimal. Note: if an extended random file is selected then any address x given will refer to a location in the header extenders rathar than to the data blocks.

F@
Select this port's active file to be examined and/or modified by other commands. The form F@n will select the active file of port number n to be examined and/or modified by other commands. CORA in the active file file header is ignored, and all addressing is relative to the beginning of user storage in the partition.

F
Select real core to be examined and/or modified by other commands.

Gu/x or Gx
Select the disc block at Real Disc Address x on Logical Unit u (where u is in octal) to be examined and/or modified by other commands. In this mode, only addresses less than 400 octal will be accepted. The simple form Gx assumes Logical Unit zero.

H
Select the header block of the currently selected file to be examined and/or modified by other commands. In this mode, only addresses less than 400 octal will be accepted.

Ix:text
Input ASCII string, where "text" is any string of characters, starting at address x. Result is identical to use of assembler pseudo-op .TXTF with reverse packing (preceded by .TXTM 1). A CTRL Z will be entered in the string as a RETURN code.

Jx,y
Search for potential address errors. Scans from address x-200 through x+177 for all relative reference instructions spanning address x that are less than y words from maximum relative displacement; ie, any place that an address error would be caused by inserting y lines of code at location x.

Kx,y,z Store the octal constant z in locations x through y, inclusive.

Lx List as symbolic instructions starting at address x. Listing may be terminated by pressing ESC.

LxN Same as Lx except each value is also printed in octal.

Mx,y,z Move the contents of locations x through y, inclusive, to the locations starting at z. The source and destination areas may overlap if desired without bad effects or loss of information.

Nx,y,z This command and the similar Nx,y,z,m command are the same as the respective S commands except that a not-equal comparison (rather than equality) causes the contents of a cell to be listed.

Ox Output ASCII string starting at address x. Output terminates on any byte less than 200 octal or if ESC is pressed. A CTRL E code is printed as a colon.

Px,y Punch locations x through y, inclusive, on the high speed paper tape punch in binary loader format. If the system does not have a high speed punch (no $PTP driver) then DSP attempts to use the master terminal ($PTM driver). Note: leader is automatically punched when the first Px,y command is given.

Px Punch an end block with a starting address x, which must be non-zero, then punch trailer. Must be preceded by at least one Px,y command.

P Punch an end block with no starting address, then punch trailer. Must be preceded by at least one Px,y command.

Qx Query cell continuously. Repeatedly prints the contents of address x in octal, allowing a swap after each print out. May be used from one terminal to monitor changes to a cell, either in core or in a disc file, while executing tasks from another terminal to cause such changes. Terminate by pressing ESC.

R — Read binary format paper tape on the high speed reader into the destination selected by the last F command. Each tape record (about four inches) is read into a buffer and checksummed before data are stored. The first 21 words octal of the last breakpoint snapshot will be lost because the same buffer area is used. If the system does not have a high speed reader (no $PTR driver) then DSP attempts to use the master terminal ($PTM driver). See "How to Load a Text File" in the IRIS User Reference Manual for restrictions on using $PTM.

Rx — Same as R except that all addresses on the tape are displaced the same amount so that the first word on the tape goes into address x, which must be non-zero.

Sx,y,z — Search locations x through y, inclusive, for the octal constant z. If found, the location is printed, and the contents of that location are listed as a symbolic instruction.

Sx,y,z,m — Same as Sx,y,z except that the contents of each cell are ANDed with mask m before being compared with constant z. For example, the command

```
S400,1120,53,101777
```

will search locations 400 through 1120, inclusive, for any instruction referencing location 53.

T — (Not used)

Ux — Print snapshot yanked into FMAP cells of active file at last breakpoint. Start print out (in octal dump format) at virtual address x where y <= x <= y+100 and y is the snapshot address set by the last Y command. Caution: the addresses will be wrong if a different Y command has been given since the breakpoint was encountered.

V — Verify paper tape. This command and the Vx command are the same as the respective R commands except that information from the tape is compared with the contents of the selected file (or core) instead of being stored. If a difference is detected, the address and the word from storage are listed.

Wu/x or Wx — Write the disc block selected by the last G or H command on disc at Real Disc Address x of Logical Unit u. This command will be rejected if u/x is not a legal Real Disc Address or if a single disc block has not been selected. The simple form Wx assumes Logical Unit zero.

X — Exit from DSP, clear any existing file selection or breakpoint, and prevent re-entry to DSP without the password.

Yx — Set first address of 101 word (octal) core area to be yanked into the FMAP cells of the active file header as a core "snapshot" when a breakpoint is encountered. If x=0, don't yank any area of core.

Zx — Search for relative reference. The 256 words centered on location x are searched for any storage reference instruction that references location x using relative addressing. Any such instruction is listed in symbolic form.

Zx,y — Same as Zx except a search is done for each address x through y.

; — Comment. Any line starting with a semi-colon will be ignored by DSP. This is used mainly to include comments on patch tapes.

4. SYSTEM UPDATES

Before making any system update it is best to have a current backup of Logical Unit zero and to REMOVE all other Logical Units. Also, in some cases, it is necessary that no other users are active on the system. Back up the updated system as soon as it has been determined that the update is satisfactory.

There are five catagories of system updates which are entered as follows:

1) A new processor (eg, a new language processor or a new system utility processor) may be loaded by use of PLOAD (see Section 4.1).

2) A new version of an old processor may be loaded in either of two ways:

   a) CHANGE the name of the old processor (or KILL it), then use PLOAD to load the new version, or

   b) Use DSP's "R" command to overlay the old version as described for DISCSUBS in Section 4.3.

3) A new version of a system file must be loaded by overlaying the old version as described in Section 4.3 or by doing a SysGen.

4) Patches and patch tapes are entered by use of DSP (see Section 4.10).

5) The system configuration may be changed at any time as described in Sections 2.4 through 2.10.

4.1 How to Use PLOAD

To use PLOAD (the system Program Loader) you must be logged onto the manager's account on the master terminal. Then type the system command PLOAD, and press the RETURN key. Once called, PLOAD disables interrupts and in effect stalls the system for the duration of loading tapes, but the active files are not disturbed. Upon exit, the system will resume all operations that were running before PLOAD was called.

PLOAD first asks which reader you wish to use to load the tapes. Two choices are available: (0) master Teletype reader, or (1) high-speed tape reader. Press either the zero key or the one key to select the desired reader.

When asked "NAME?" enter the Filename under which to build the new file, or press CTRL C to exit from PLOAD. Existing files cannot be replaced by PLOAD. To replace an old file, either delete it or change its name before calling PLOAD.

Copyright 1978
Educational Data Systems

A "TYPE?" will be requested for each file being loaded. Refer to Section 2.3.6 for a table of processor types, and to Section 8.7 if making up a new file type.

If PLOAD prints "RDR OK?" the tape has aborted due to a reader time out. Try loading the tape again from the beginning. A second failure to load indicates either a bad tape or a hardware problem in the tape reader.

After loading any file, be sure to make any necessary patches. Multiple part processors such as BASIC/RUN/RUNMAT should not be replaced while the system is in use since the parts refer directly to each other.

A new version of a language processor (such as BASIC) might not be compatible with saved files in that language. In such a case, all such program files must be DUMPed to text files, and the saved versions must all be deleted before replacing the processor. The new version of the processor is then used to LOAD and reSAVE the programs.

4.2 How to Replace BZUP

BZUP may be replaced by use of the binary paper tape loader. SHUTDOWN may be used to get the loader into core (see Section 8.4). After the BZUP tape has been loaded into core, the computer should halt at address 24403; if not, RESET and START at 24400. Now press the CONTINUE switch to write BZUP on the disc, then press the colon key to do an IPL and bring up IRIS.

4.3 How to Replace DISCSUBS

This file occupies a fixed location on the disc, thus requiring a special procedure for replacement. DISCSUBS may be replaced, either in whole or in part, by use of DSP's "R" command (see Section 3.3) to read the new tape in, overlaying the current DISCSUBS file. To replace all or any part of DISCSUBS by this method, be sure that there are no other users on the system, put the tape to be loaded in the tape reader, and give DSP the commands

```
F DISCSUBS
R
```

for each tape to be loaded. If a checksum error occurs, back up the tape to a record gap at least one record ahead of where the error occurred, and repeat the R command (restart the tape from the beginning if unsure of how to read record gaps and addresses on the tape). See Section 10.2 for more information. If using the master terminal tape reader, see to "How to Load a Text File" if the IRIS <u>User Reference Manual</u> for restrictions on using $PTM.

WARNING! Do an IPL after loading all tapes and before using the system for any other purpose.

4.4 How to Replace DBUG

DBUG resides in the REX file. If it becomes necessary to replace DBUG, put the DBUG tape in the paper tape reader, enter DSP, and give an "F REX" command followed by an "R" command. The tape will be read into the REX file, overlaying the old version of DBUG.

4.5 How to Replace REX, SIR, and PLOAD

The preferred method of replacing these system components is to do a new System Generation. The following procedure will permit doing this without loss of users' files on the system disc:

1) Use COPY to copy all users' files on Logical Unit zero to any other Logical Unit (see "How to Use COPY" in the IRIS _User Reference Manual_).

2) Punch a tape of the current users' accounts by giving the system command

```
COPY $PTP = ACCOUNTS
```

or substitute $PTM for $PTP if the system does not have a high speed punch. Note: this step and step 4 are not neccessary if using the default accounts or if all new accounts are to be created.

3) Do a new SysGen on Logical Unit zero (see Section 2.3). Exercise the new system to be sure it is working properly.

4) Load the tape of accounting information punched at step 2 by giving DSP the commands

```
F ACCOUNTS
R
```

It will be necessary to append additional blocks if the old ACCOUNTS file exceeded 16 entries. To do this, first give DSP the commands

```
F ACCOUNTS
A 400
A 1000
```

etc. until enough blocks have been appended.

5) Use COPY to copy any desired users' files back to Logical Unit zero, or avoid steps 1 and 5 in future updates by not allowing users' files on Logical Unit zero.

An alternative method of replacing REX, SIR, and PLOAD without having to move user's files is as follows:

1) Do a backup of Logical Unit zero. Check that the REX and SYSGEN tapes to be used have the same assembly and punch dates.

2) Enter DSP and give an "F REX" command.

3) Put the REX tape in the paper tape reader, and give DSP an "R" command. This reads the REX tape into the REX file, overlaying the old version of REX.

4) Take the SYSGEN tape and visually locate the record that starts at 14000 octal (the current starting address of SIR). Put the tape in the reader starting at that record, and give DSP another "R" command. SIR will be read from the SYSGEN tape into the REX file, overlaying the old version of SIR. A "NO SUCH ADDRESS" error will occur near the end of the tape; this is okay since SYSL is also on this tape but is not contained in the REX file.

5) Put the proper SOV (SysGen Overlay) tape in the reader, and give DSP another "R" command. SOV will be read into the REX file, overlaying the proper locations as it does when it is read into core during a SysGen.

6) Take the REX tape and visually locate the record that starts at 200 octal. Put the tape in the reader starting at that record, and give DSP an "F PLOAD" command followed by an "R" command. A "NO SUCH ADDRESS" error will occur after reading a few feet of tape; this is okay since only 400 words octal of this tape are read into PLOAD.

7) Put the SYSGEN tape in the reader (starting at the beginning), and give DSP another "R" command. Since PLOAD is the first part of the tape, it will be read in and will overlay the old contents of the PLOAD file. A "NO SUCH ADDRESS" error will occur when the tape has been read beyond the portion containing PLOAD; this is okay because only the first part is needed.

8) Do an IPL by starting at location 4. Check out the system, including PLOAD, then do a backup.

It is necessary to replace all of REX, SIR, and PLOAD at the same time from tapes with the same assembly and punch dates. Failure to replace all of the components when any one is being replaced, or the use of tapes with different dates, could result in loss of the system.

Copyright 1978
Educational Data Systems

4.6 How to Replace SCOPE, BYE, DSP, or MESSAGES

To replace these files, first change the Filename (eg, to OLDSCOPE, OLDBYE, etc.), use PLOAD to load the new tape under the correct Filename (see Section 4.1), and then do an IPL immediately after loading the new tape. To do this, RESET and START at location 4 without first exiting from PLOAD. SCOPE, BYE, and DSP are system processors, however, which cannot be changed or deleted without first changing their file type. Use DSP to change the type by giving the commands:

```
        F SCOPE (or F BYE, or F DSP)
        H
        10:33401
```

When finished, the old processor (OLDSCOPE, etc.) may be deleted by giving the command

```
        KILL EkeyE Filename
```

where "key" is the password to KILL for deleting a processor (see Section 4.9) and Filename is the name of the processor to be killed (eg, OLDSCOPE).

4.7 How to Replace BASIC, RUN and RUNMAT

These processors may be replaced as described in Section 4.6 except that special action must be taken to link these processors properly to each other. After exiting from PLOAD, use DSP to put zero in location 200 in BASIC (not necessary if BASIC was replaced). Exit to system command mode, type BASIC, and press the RETURN key twice to cause BASIC to link itself and RUNMAT properly to RUN.

4.8 How to Replace Other Processors and Drivers

Other processors and peripheral device drivers may also be replaced as described in Section 4.6. After deleting the old file or changing its name, PLOAD may be used to load the new tape with its normal type. New processors and drivers for new peripheral devices may be also loaded by use of PLOAD. A new peripheral driver will not be available for use, however, until an IPL has been performed.

4.9 How to Change Processor Passwords

The SHUTDOWN, DSP, PORT, CLEANUP, and KILL processors each have a password in location 570 of the processor to provide a double protection against unauthorized use. The object tapes have all passwords assigned as "X". Any time the tape is loaded, the password should be changed. DSP may be used to examine and/or modify a password. For example, if the password to DSP is to be XR2Z then DSP may be given the following commands:

```
   F DSP        to select DSP
   0570         to examine the current password
   I570:XR2Z    the password becomes XR2Z
```

Up to 15 ASCII characters (letters and digits only) may be specified as the password.

4.10 How to Enter Patches and Patch Tapes

Minor modifications or patches may be made to the system at any time by use of DSP. It must be realized, however, that someone may be using a processor at the same time you are modifying it. Do not enter a patch in such a way that the code may be used when only partially complete. Enter the entire patch first in a free area, and then enter the jump to the patch. Of course, the processor to be patched must first be selected by use of DSP's F command.

Many updates supplied by EDS are in the form of patch tapes. Each tape contains all necessary instructions to DSP to make the patch, including selecting the file (or files) to be patched. To use a patch tape it is only necessary to log on to the manager's account and select DSP. Then place the patch tape in the terminal's reader, and press its START switch momentarily. The tape will make the patches and exit from DSP. In most cases, such patches may be made while the system is in use; in some cases, however, the cover letter sent with the tape will instruct that all other users must log off before the patch is entered. The PORT ALL EVICT command may be used to insure that all users are off of the system (see Section 8.10).

Patch space is available at the end of REX for patching the operating system itself. The PATSP (Patch Space) and ENDP (End of Patch Space) cells in the INFO table in the REX file (not in core) indicate what space is available for such patches. A new patch should be entered starting at the address in PATSP, and it must not extend beyond the address given in ENDP. When the patch is entered, PATSP (in REX) must be changed to point to the next location after the end of the patch. The Software Definitions listing gives the displacements of PATSP and ENDP in the INFO table.

Copyright 1978
Educational Data Systems

5. START-UP AND SHUT-DOWN PROCEDURES

Under most circumstances, it is recommended that the system be left running 24 hours a day. However, some users find it necessary to shut down at night or on weekends. Also, if the Power Fail Auto Restart option is not in the computer, it is necessary to start up the system after a power failure. The system must also be restarted after use in stand-alone mode.

5.1 System Start Up Procedure

Turn the computer switch to the ON position, then turn on the master terminal and the disc (in some installations the disc power may go on with the computer). Wait for the disc to reach full speed; some disc drives have a "ready" light to indicate operational speed.

Set the switches to address 000000 and press RESET, then START to restart the system only if it was shut down by use of a simple SHUTDOWN command (see Section 5.3) or if a power failure occurred and the computer does not have the Power Fail Auto Restart option or the power switch was not in the LOCK position. Do not attempt to start at location zero if the system was used in stand-alone mode, or if system operation was terminated in any other manner.

Set the switches to address 000004 and press RESET, then START to do an IPL and restart the system only following a temporary shut down without the use of SHUTDOWN or if a stall occurred.

If either procedure fails to start the system on the first try then use the proper IPL bootstrap procedure as given in Section 5.2. Once the computer is started, turn the computer power switch to the LOCK position and remove the key. Note: a halt with 67077 in the data lights indicates a disc read error that could not be corrected with 16 retrys. Register A1 contains the Real Disc Address of the block which could not be read.

The IPL (Initial Program Load) sequence brings a fresh copy of the operating system into core from the disc. Then the System Initializing Routine (SIR) takes over to perform many start up tasks, one of which is to examine all files on the system disc, checking for several types of errors. This may take a minute or more on a large disc. If an error is detected then a message is printed identifying the file and the type of error. In most cases, it is desirable to delete the file, which is done by pressing the space key (or any key other than "@" or RUBOUT). If the error message indicates that the file was being built or deleted, then the file may be saved (and the build and delete status bits cleared) by pressing the "@" key. If nothing is done within 20 seconds then SIR examines location 201 of REX; if location 201 contains 177 octal then SIR halts, and CONTINUE may be pressed to transfer control to DBUG. If location 201 contains any other value then SIR deletes the file and continues.

The file can not be saved by pressing the "@" key in response to other types of errors. However, if it is desired to save the file, and you know enough about the system to attempt it, press the RUBOUT key to halt the system; then press CONTINUE to transfer control to DBUG (see Section 3.2), and type an "A" command to print the register contents; A1 contains the header disc address, and A2 is the location of the header in core. After any such procedure, the IPL sequence should be restarted from the beginning. If the problem appears to be in a driver, core-resident DISCSUB, etc., then the initializing sequence may be minimized by starting SIR with switch 1 up; to do this, IPL into BZUP (see Section 3.1), leave switch 1 up, and press the colon key (or press CONTINUE if the system halted). This will attempt to bring up the system with no ports other than the master terminal, no peripheral drivers, no core-resident DISCSUBS, etc., so that DSP can be used from the master terminal to investigate the problem.

SIR then asks for bad blocks. Press the RETURN key in response to the "BAD BLOCK?" message if there are no new bad blocks to report on the system disc, or type the address (as given in register A1 in a disc check error trap) and press RETURN. The message BLOCK IN USE will be printed if the address entered is allocated to any existing file. Certain disc blocks are required for operation of IRIS, and such blocks may not be marked as bad; otherwise, the file may be deleted, and another IPL performed to mark the bad block. The DSPS cells and FMAP cells of the DMAP header are used for the "bad blocks" list, which is terminated by a zero word. Up to 80 blocks may be listed as bad on each Logical Unit.

Finally, SIR asks for the date and time which it uses to set the real time clock. A typical date/time type-in might be:

```
78,2,10,16,33
```

Note that the hours must be based on a 24 hour clock. The above example, therefore, represents February 10, 1978, at 4:33 PM. If the time entered is earlier than the last access time of the most recently accessed file, then the computer will ask "TIME RUNS BACKWARDS?". Check over your type-in, and press the Y key (for yes) if it is correct. Otherwise, press N and enter the date and time again. If the system is left running 24 hours a day, then the time must be set (see Section 8.9) on the first of each month since the real time clock assumes 31 days in every month.

A stable light pattern with only lamp 14 bright and a flashing carry light indicate that the system is up and ready for use. The carry light should flash at exactly one cycle per second; a different flash rate or lack of flashing altogether indicates a problem in the Real Time Clock. Note: most multiplexers supply the clock which is used for system time; a high clock rate may be due to the clock being supplied by two devices.

Only the system disc (Logical Unit zero) is activated by SIR. If there are other Logical Units on the system then they must be "installed" before they will be accessible. See "How to INSTALL a Logical Unit" in the IRIS User Reference Manual. Also, see Section 8.8 in this manual.

5.2 IPL Bootstrap Programs

If the procedure given in Section 5.1 fails to start the Initial Program Load sequence then it is necessary to do an IPL Bootstrap.

If the computer has the Program Load option and the disc controller will respond properly to the Program Load function, then set 1000xx in the data switches (where xx is the controller's device address), press RESET, then PROGRAM LOAD.

In many cases either the computer does not have the Program Load option or the disc controller will not respond properly. In such cases it is necessary to key in a bootstrap program. For most head-per-track discs, such as with the Data General 4019 controller, or for the Ampex "Megastore" or the Dataram "Bulk Core", the following procedure will suffice:

set data switches	then press
000376	RESET, EXAMINE
0601xx	DEPOSIT
000377	DEPOSIT NEXT
000376	START

where xx is the controller's device code (usually 20).

For a Telefile disc controller, the same bootstrap program shown above may be used, but the device code is usually 33.

For any disc on a Data General 4046 or similar controller, one of the following bootstraps may be used:

If Logical Unit zero is on platter zero (usually the removable cartridge):

set data switches	then press
000376	RESET, EXAMINE
060133	DEPOSIT
000377	DEPOSIT NEXT
000376	START

If Logical Unit zero is on platter one (usually the non-removable disc):

set data switches	then press
000376	RESET, EXAMINE
040000	DEPOSIT ACO
061133	DEPOSIT
000377	DEPOSIT NEXT
000376	START

For any disc on a Data General 4234 or similar controller, one of the following bootstraps may be used:

If Logical Unit zero is on platter zero (usually the removable cartridge):

set data switches	then press
000372	RESET, EXAMINE
000000	DEPOSIT AC0
000000	DEPOSIT AC1
001400	DEPOSIT AC2
071333	DEPOSIT
063077	DEPOSIT NEXT
062233	DEPOSIT NEXT
067033	DEPOSTI NEXT
061133	DEPOSTI NEXT
000377	DEPOSIT NEXT
000372	START, CONTINUE

If Logical Unit zero is on platter one (usually the non-removable disc):

set data switches	then press
000372	RESET, EXAMINE
000000	DEPOSIT AC0
001000	DEPOSIT AC1
001400	DEPOSIT AC2
071333	DEPOSIT
063077	DEPOSIT NEXT
062233	DEPOSIT NEXT
067033	DEPOSIT NEXT
061133	DEPOSIT NEXT
000377	DEPOSIT NEXT
000372	START, CONTINUE

Copyright 1978
Educational Data Systems

For any disc on a Ball (Decision) 3150 or 3170 controller one of the following bootstraps may be used:

If Logical Unit zero is the non-removable disc:

set data switches	then press
000376	RESET, EXAMINE
0601xx	DEPOSIT
000377	DEPOSIT NEXT
000376	START

If Logical Unit zero is the removable cartridge:

set data switches	then press
000376	RESET, EXAMINE
020000*	DEPOSIT AC0
0611xx	DEPOSIT
000377	DEPOSIT NEXT
000376	START

* 050000 for a 3170 controller

where xx is the controller's device code (usually 40).

For an Iomec disc, the bootstrap is longer since the disc controller uses cell 20 in core as a word counter. Use one of the following bootstrap programs for an Iomec disc controller:

If Logical Unit zero is the removable cartridge:

set data switches	then press
000020	RESET, EXAMINE
177401	DEPOSIT
000012	DEPOSIT AC0
000000	DEPOSIT AC1
000000	DEPOSIT AC2
000374	EXAMINE
061370	DEPOSIT
066270	DEPOSIT NEXT
073170	DEPOSIT NEXT
000377	DEPOSIT NEXT
000374	START

If Logical Unit zero is the non-removable disc:

set data switches	then press
000020	RESET, EXAMINE
177401	DEPOSIT
000012	DEPOSIT AC0
000200	DEPOSIT AC1
000000	DEPOSIT AC2
000374	EXAMINE
061370	DEPOSIT
066270	DEPOSIT NEXT
073170	DEPOSIT NEXT
000377	DEPOSIT NEXT
000374	START

The System Industries disc controllers also require longer bootstrap programs since they access some of the control parameters from core. Use the following bootstrap programs for a System Industries 3015 or 3045 controller:

If Logical Unit zero is the non-removable disc:

set data switches	then press
000365	RESET, EXAMINE
020405	DEPOSIT
024405	DEPOSIT NEXT
004405	DEPOSIT NEXT
000400	DEPOSIT NEXT
000000	DEPOSIT NEXT
000000	DEPOSIT NEXT
000100	DEPOSIT NEXT
0771xx	DEPOSIT NEXT
0650xx	DEPOSIT NEXT
0632xx	DEPOSIT NEXT
000377	DEPOSIT NEXT
000365	START

If Logical Unit zero is the removable cartridge:

set data switches	then press
000365	RESET, EXAMINE
020405	DEPOSIT
024405	DEPOSIT NEXT
004405	DEPOSIT NEXT
000400	DEPOSIT NEXT
000000	DEPOSIT NEXT
020000*	DEPOSIT NEXT
000100	DEPOSIT NEXT
0771xx	DEPOSIT NEXT
0650xx	DEPOSIT NEXT
0632xx	DEPOSIT NEXT
000377	DEPOSIT NEXT
000365	START

* 100000 for a 3045 controller

where xx is the controller's device code (usually 40).

For a System Industries "Kahili" controller (9500 series) the following bootstrap program may be used:

set data switches	then press
000371	RESET, EXAMINE, DEPOSIT ACO
000000	DEPOSIT
000400	DEPOSIT NEXT
000000*	DEPOSIT NEXT
000000	DEPOSIT NEXT
000000	DEPOSIT NEXT
0611xx	DEPOSIT NEXT
000377	DEPOSIT NEXT
000376	START

* 002000 for drive #1
004000 for drive #2
006000 for drive #3

where xx is the controller's device code (usually 50).

For a Digital Computer Controls 116446 disc system, the Program Load feature may be used if the system is on drive #0. Set the data switches to 100030 octal, press RESET, then PROGRAM LOAD, then press the colon (:) key on the master terminal. If the system is not on drive #0, then key in this bootstrap program:

set data switches	then press
000376	RESET, EXAMINE
0n0000	DEPOSIT AC1
065130	DEPOSIT
000377	DEPOSIT NEXT
100376	START

where n is the drive number, then press the colon (:) key on the master terminal.

For a Xebec XDF-50 system, the PROGRAM LOAD feature may be used if the system is on physical partition 0.0; otherwise use one of the following bootstrap programs:

If Logical Unit zero is the removable cartridge:

set data switches	then press
000376	RESET, EXAMINE
060130	DEPOSIT
000377	DEPOSIT NEXT
000376	START

If Logical Unit zero is the non-removable disc:

set data switches	then press
000373	RESET, EXAMINE
003400	DEPOSIT AC0
001000	DEPOSIT AC1
000400	DEPOSIT AC2
000000	DEPOSIT AC3
076030	DEPOSIT
073030	DEPOSIT NEXT
067031	DEPOSIT NEXT
061130	DEPOSIT NEXT
000377	DEPOSIT NEXT
000373	START

For a Minicomputer Technology TDC-802, use the following bootstrap program to start an IPL from drive zero:

set data switches	then press
000401	RESET, EXAMINE
060136	DEPOSIT
000400	DEPOSIT NEXT
000401	START

5.3 System SHUTDOWN Procedure

All users must be logged off the system before it is shut down. This is to ensure that all accounts have been properly updated and that there are no open files. For this reason, the system command

```
SHUTDOWN Ekey
```

where "key" is the password assigned to SHUTDOWN (see Sections 2.3.11 and 4.9) should be given to shut down the system. Normally, this command must be given by the system manager on the master terminal (port zero), but other accounts and/or other ports may be allowed to shut down the system (see Section 2.3.13).

SHUTDOWN will check all ports and, if any port is in use, the message

```
PORT #n IS IN USE
```

will be printed, where n is the number of a port where the user has not logged off.

The MAIL command may be used to send a message to port n to request the user to log off, or the PORT EVICT command may be used to force the port (or all ports) to be logged off (see Section 8.10). If all ports are inactive, SHUTDOWN will stop the multiplexer so that no one can log on, and will then chain to BYE and log off the master terminal.

The computer will halt after the master terminal has been logged off. Now turn off the disc and the master terminal, and turn the computer power OFF. A RESET and START at location zero (octal 000000) may later be used to restart the system without an IPL (see Section 5.1).

WARNING! Do not attempt to restart the system at location zero except after shutting down in exactly the manner described above. Use the procedure in Section 5.2 under any other circumstances or if the system fails to restart by this method on the first attempt.

6. PROBLEM ANALYSIS AND REPORTING

This section outlines a procedure for analyzing and reporting any bugs found in Educational Data Systems' hardware or software. The use of this procedure assures efficient processing of bug reports by EDS so that the problem can be fixed with a minimum delay.

6.1 Trap Messages

A trap message is printed when any error is detected that indicates a hardware or software fault. A hardware fault is most commonly caused by a disc error which, after sixteen tries, cannot be corrected. A software fault occurs if one of the cross checks built into the software detects an illegal condition, such as an illegal disc or core address.

The trap message tells the type of fault (see Appendix 5), the core location where the fault was detected, and the Filename of the processor that was in core at the time (in some cases there may be no processor in core). The next line of the message lists the STATUS; ie, the contents of registers A0 through A3 and the carry flip-flop. If the fault was detected by a disc-resident subroutine, a third line will be printed giving the address in the DISCSUBS assembly listing. If disc-resident subroutines were nested at the time, then the address in the DISCSUBS listing will be printed for each nested call. An asterisk preceding a discsubs address indicates that the trap did not actually occur within the discsub itself, and the address given is an absolute core address where the trap occurred -- usually by a core-resident routine (within REX) that was called by the discsub.

The message is printed on the terminal of the user that was in core at the time the fault was detected, and that user's active file may then be cleared. The reason for clearing the active file is that the fault may have been caused by an error in that file. If no user was in core at the time, then the message is printed on the master terminal and will be lost if the master terminal is not on at that time.

Time-sharing is inhibited during the few seconds it takes to type out the trap message. Any input or output in progress at the time may continue, but there will be no response after that input or output until the trap message is completed.

All trap messages should be saved until they have been analyzed to determine the cause of the fault. If a software problem is suspected, the trap message should be forwarded to Educational Data Systems along with a Bug Report (see Section 6.3) so that the problem can be corrected.

6.2 Errors and Other Problems

This category includes all software problems that do not cause a trap message, as well as all hardware problems. Any problem with hardware that is under an EDS warranty or service contract should be documented by running the diagnostic test routine and including a copy of its print out along with the bug report.

Software problems are harder to diagnose since they usually occur somewhere in a very complex application program. To make diagnosis possible, it is usually necessary to isolate the problem to a single statement or a very small group of statements that will produce the error.

Since this is more easily done by the person who wrote the application program, it is best to attempt to isolate the problem before sending the bug report. Include a complete listing and a sample run of the smallest program that will cause the problem.

Many "software" problems occur because of a transient hardware error or because the core copy of the operating system has somehow been changed. In such a case, the problem may not re-occur or may be cured by doing an IPL. For this reason it is best to try an IPL first, especially if the problem occurs in a program that previously ran without errors. Even if the problem is not immediately repeatable, the documentation should be retained in case the same problem occurs again at a later time. Information in core can be changed by a hardware failure, a power line transient, static electricity, or by another software bug which may be completely unrelated to the symptom.

6.3 Bug Reporting Procedure

Please report any bug found in any Educational Data Systems' product (software or hardware) by filling out an EDS BUG REPORT form. If none is available then a Rediform 4S474 three part form may be used, filled out as follows:

TO	Your company name
AT	The name and address of the facility where the person making the report can be contacted.
SUBJECT	The name of the EDS product. Include any date, revision number, or serial number. Examples: IRIS <u>User Reference Manual</u> (EDS 1017-10) FORMAT 8-17-77 EDS 310 MULTIPLEXER (Serial #405) BASIC 7-15-77
DATE	The date the report is made.

Copyright 1978
Educational Data Systems

Describe the problem in detail, including all symptoms. Include information about any concurrent events or sequence of events leading up to or needed to reproduce the problem. Be concise; give an example of the smallest program that will demonstrate the problem. Attach all applicable computer printout and programming examples.

Use a separate BUG REPORT for each problem encountered. If you feel that two or more bugs may be related, reference should be made to the related reports. Sign the report, keep a copy, and send the original to Educational Data Systems at the address shown on the title page of this manual.

An isolated TRAP message with no background data is useless, as is an out-of-context statement such as "string comparison doesn't always work." There are several types of software problems. Each should be reported as outlined below:

a) TRAP MESSAGE. A TRAP message results from either a hardware failure or from a cross-check in the software. Attach the TRAP print out and the six feet of paper from your terminal leading up to the trap. Describe any other symptoms noted and the effect, if any, on other users.

b) Incorrect Operation (system continues normal operation but behaves incorrectly or gives the wrong answer in a particular instance). Include hard copy of the error with the BUG REPORT form (see Section 6.2).

c) Computer Stalls (RUN light on, but no response to any user). Write down the state of the ION light and the DEFER light, then press RESET, and write down the values displayed in the ADDRESS, DATA, and CARRY lights, and examine and write down the contents of each register ACO through AC3.

d) Computer Stops (RUN light out). This may be any of several problems. In any case, write down the values displayed in the ADDRESS and DATA lights and the state of the CARRY and ION lights; then press RESET, and examine and write down the contents of each register ACO through AC3. The type of problem is indicated by the state of the DATA lights as shown on the next page.

In any case other than a power failure, state whether the problem is repeatable or occurred only once, describe the last action by the user at each terminal, and include any other possibly pertinent information. In cases (a) and (b) above, and the 067077 and 073077 halts on the next page, the problem will be self clearing, and the system sould continue normal operation except for the particular error. In all other cases it will be necessary to do an IPL (see Section 5.1) to restore system operation.

DATA lights	Meaning of Halt
077077	Double TRAP. A trap occured while attempting to print the trap message. After writing down all information, press CONTINUE, then again examine and write down the contents of all registers and the CARRY light. The second set of information is from the initial trap.
067077	Power fail restart. A Power Fail Auto Restart was being attempted, and one or more disc drives require operator intervention. Ready all disc drives, and press CONTINUE to resume system operation.
073077	Power fail halt. A power failure occurred, and there was a failure in the Power Fail Auto Restart hardware. Ready all disc drives, and press CONTINUE to resume system operation.
063377	No free nodes or interrupt stack overflow. If A2=0 then there were not enough free nodes available for task queuing; do an IPL, and reduce the amount of core-resident routines to allow more free nodes, then do another IPL. If A2>0 then the interrupt stack has overflowed; either there is an interrupt mask hardware problem, or some device's mask bit is specified incorrectly in its driver.
073377	Disc time-out. A disc drive did not go ready within the expected time after an initialize or recalibrate command.
067377	Interrupt not acknowledged. Some hardware device has interrupted but did not present its device code to an INTA instruction. This indicates a hardware problem, most likely improper routing of the interrupt priority line on the computer backplane (pins A95 and A96). A restart may be attempted by pressing RESET, then CONTINUE.
063277	INSTALL has encountered a bad file (see opposite).
077377	Unknown halt. Unused areas of core are usually filled with 077377 halt instructions. Such a halt, or any other halt not shown above (eg, 063077), indicates an abnormal jump in the software.
xxxxxx	Any value other than above in the data lights. Most likely, a power failure occurred, and either the computer's power switch was not set to LOCK, or the computer does not have the Power Fail Auto Restart option. Examine location zero; if it contains 062677 octal then this assumption is correct, and a START at zero should resume system operation (first be sure all disc drives are ready).

6.4 Bad File Recovery

If a "bad" file is encountered during an INSTALL, and location 200 of INSTALL contains zero (see paragraph 2.3.17), then the computer will halt (see 063277 opposite). Register A1 will contain an INDEX block Real Disc Address, A0 will contain the displacement into that block to the questionable file's entry, and A2 will contain an error indicator as follows:

2 = Disc address in header doesn't match INDEX entry
3 = Filename in header doesn't match INDEX entry
4 = Incorrect number of disc addresses in header
5 = File uses disc block already marked as in use

Write down the contents of these registers, then press CONTINUE; this will cause the file to be deleted, and INSTALL will continue installing the Logical Unit. Alternatively, press EXAMINE NEXT, then CONTINUE to abort the INSTALL without deleting the file and attempt to restore normal system operation. Then, at a time when the manager is alone on the system, do an INSTALL FAST and use DSP to examine the INDEX entry and file header and attempt restoration of the file.

WARNING -- Do not do an INSTALL FAST on a questionable Logical Unit at a time when any one may attempt to use that unit!

Copyright 1978
Educational Data Systems

7. ACCOUNTING SYSTEM

The operating system keeps track of the system resource usage by each user account. Disc block usage is maintained on a dynamic basis; ie, the user's account is immediately updated to show the change in disc usage each time the user deletes, creates, or expands any file. CPU and connect time usage and peak disc block usage are updated by BYE when a user logs off.

All accounting information is kept in a disc file named ACCOUNTS which may be examined and/or modified by a BASIC program via data file access. Only the system manager can run such a program successfully since the ACCOUNTS file is read protected and write protected against all users except the manager account (priv 2, group 0, user 1). This section gives information on how to write BASIC programs to perform the system accounting functions and how to use programs provided by EDS to set up and examine user accounts.

Only one privilege three (system) account is allowed on the system. The system account is the first entry in the ACCOUNTS file. The system account ID cannot be changed by the system manager. Only one manager account is allowed on the system. The manager account number is group zero, user one, at privilege level two (octal 100001). Do not change the manager's account number or its Logical Unit assignment!

Each block of the ACCOUNTS file will hold 16 account entries, and each entry occupies octal 20 words as follows:

Words	Contents
0-5	Account ID string (up to 12 characters)
6	Assigned priority level (0 to 377 octal)
7	Assigned Logical Unit
10	Account number (priv, group, user)
11	Connect minutes remaining *
12	CPU seconds remaining *
13	Maximum disc blocks allotted **
14	Disc blocks now in use
15	Peak disc block usage
16,17	File use charges (floating 2-word BCD)

* The values for connect minutes remaining and CPU seconds remaining must both be either positive or equal to 100000 octal (no limit), or the user will not be allowed to log on. A positive value will be reduced by the amount of time used each time a user logs off.

** The maximum disc blocks allotted is simply a limiting value for disc block usage by this account. The blocks are not reserved for this account but merely limit the number that can be drawn from the general pool. Thus, the total block allottment to all accounts may, and generally should, exceed the number of blocks available on a Logical Unit.

7.1 ACCOUNTS File Access from Business BASIC

The following statement forms may be used in a Business BASIC program to access an ACCOUNTS file entry:

```
     line number OPEN #x,"lu/ACCOUNTS"

     line number READ #x,R;A$,P,U,N,T1,T2,M,D1,D2,C

     line number WRITE #x,R;A$,P,U,N,T1,T2,M,D1,D2,C
```

where:

```
     x  = Desired channel number
     lu = Logical Unit to be accessed
     R  = Record number in ACCOUNTS file
     A$ = Account ID (empty entry if A$ is a null string)
     P  = Assigned priority level
     U  = Assigned Logical Unit
     N  = Account number (priv, group, user)
     T1 = Connect time remaining (minutes)
     T2 = CPU time remaining (seconds)
     M  = Maximum disc blocks allotted
     D1 = Disc blocks currently in use
     D2 = Peak disc block usage
     C  = Net file use charges in dimes
```

The account number "N" may be broken up as follows:

```
     Privilege level   L = INT(N/16384)            0 <= L <= 2
     Group number      G = INT((N-L*16384)/64)     0 <= G <= 255
     User number       U = N-(L*16384+G*64)        0 <= U <= 63
```

and it may be recombined as N=L*16384+G*64+U. Group zero, user zero is reserved for the system account.

The ACCOUNTS file is assigned to the manager's account and has 77 protection. Therefore, it can be accessed only from the manager's account.

Before creating a new account, a search should first be made for the intended ID string, and a second search should be made for the intended account number, N, to be sure neither is already in use. Then search for an entry with a null ID string, and write all items into that entry; the account ID should be written by the last statement executed to ensure that no user can log onto the account before it is complete. To delete an account, write into that account with a null ID string.

7.2 Account List Program

ACCOUNTLIST is a BASIC program that scans the ACCOUNTS file and lists all users' accounts and their current status. The items listed for each account are record number, privilege level, account number (group and user), assigned Logical Unit, assigned priority level, maximum disc blocks allotted, user ID, and account ID.

Record number zero is not an active account and is not listed. All other active records will be listed in order of ascending account numbers. Listing stops when 20 consecutive empty account entries have been scanned.

This program will run only from the manager's account because the ACCOUNTS file is protected.

7.3 Account Utility Program

UTILITY is a BASIC program that will allow the system manager to examine and modify existing accounts and to create new accounts. The program is self-descriptive. Merely give the command to RUN UTILITY, and it will give instructions for its use.

UTILITY is a very general account utility program, and as such, it is very wordy to prevent errors. All questions may be answered via a single letter. Some system managers may elect to write their own BASIC program to perform the functions in UTILITY as well as to perform other system accounting functions.

One additional data file named USERID is used by both the UTILITY and ACCOUNTLIST programs. This file must be created by the system manager by use of FORMAT, and it is usually formatted D2,S14. Item zero holds the date that the account was created as a six digit decimal number in the form MMDDYY where MM is the month (1=January, 2=February, etc.), DD is the day of the month, and YY is the last two digits of the year. Item one holds the name of the user to whom the account is assigned. The limit of 14 characters in the name is arbitrary and may be changed as desired by formatting USERID with a different string length in item one; also, items may be added to the file's format to carry other information about each user as desired.

Each record in the USERID file corresponds to the entry with the same record number in the ACCOUNTS file. Therefore, either may easily be found if the location of the other is known. If a user forgets his account ID, the system manager can search the USERID file for his name and then read the corresponding record in the ACCOUNTS file. The ACCOUNTLIST program writes NONAME into any empty USERID record for which a corresponding entry exists in the ACCOUNTS file.

7.4 Suggestions for System Accounting

A special Read File Header Information subroutine is also available for accounting. It may be used by a Business BASIC statement of the form:

line number CALL 97,U,R,F$,A,T,S,Q,C,I,D,L,H

where 97 is the file attributes access subroutine number, U, R, A, T, S, Q, C, I, D, L, and H are any numeric variable names, and F$ is any string variable dimensioned as 15 characters or longer. The value given in U specifies a desired Logical Unit, the value given in R specifies a starting record number in the INDEX file, and the other variables receive information about the file as follows:

F$ Filename. A string of up to 14 ASCII characters.

A Account number. Privilege level, Group number, and User number (see Section 7.1)

T File type:

```
  T1 = T-32*INT(T/32)
  P  = INT(T/512)
  P1 = INT((P-8*INT(P/8))
  P2 = INT(P-P1*8)
  X  = INT((T-512*P)/64)

where: T1 = file type (see Section 8.7)
       X  = R,L,I control digit
       P  = Protection digits
       P1 = Protection digit #1
       P2 = Protection digit #2
```

S File size. Number of disc blocks used by the file.

Q File status. A non-zero value indicates that the file is mapped or is being built, replaced, or deleted. See STAT in the Software Definitions for more information.

C File cost (charge for access to the file) in dimes.

I Total Income to the file in dimes. This is increased by the value of C each time a user on a different account accesses the file. Note: if the value of variable I was zero before the call, then the file's income will be cleared to zero by use of the call; otherwise, the file's income is not changed by the call.

D File creation Date (hours after 1-1-76). The age in hours may be calculated as Age = SPC(2)-D.

L   Last access date (hours after 1-1-76). The hours since last access may be calculated as HSLA = SPC(2)-L.

H   Disc address of file's header.

All values are returned in decimal. The T, Q, and H values must be converted to octal for comparison with a LIBR listing or DSP output. Variable U is zeroed by CALL 97.

If the INDEX record specified by R does not contain an entry then the next entry is automatically tried until an entry is found or the end of the INDEX is reached. If the end of the INDEX is reached, then R will be set to -1, and all other variables will be unchanged. When a valid entry is found, R is set to the record number of the next INDEX entry, and all variables are loaded from information in the file's header as indicated above.

The system manager can write a Business BASIC program using the statements discussed in this chapter to perform all accounting necessary for commercial or other controlled use of the system. A data file should be used to maintain records of each user's account and for invoicing purposes, or the format of the USERID file may be extended to accommodate the needed information. An itemized invoice could be produced automatically for each user. Examples of possible charges are:

$0.20	per CPU second
$0.10	per connect minute
$2.00	per disc block per week
$0.50	per week for maintaining account

Higher charges might be made for the higher privilege level and/or higher priority accounts. CPU and connect time used are computed by initially allotting a known amount and later determining the difference between the remaining allotment and the initial allotment. Disc usage may be charged either on current usage or on peak usage. If peak usage is used, the peak value should then be set back to the current level. A fixed charge may be made for each log-on by setting a non-zero cost on the LOGONMSG file.

The accrued net charges for access to other users' files must be added to the above total, and the user must be credited for access to his files by other users. (If a file is deleted, this credit is applied automatically by the system by subtracting the file's income from its owner's accrued charges. Thus, a user's accrued charges will be negative if his income exceeds the charges for use of other's files.) The income must be accumulated for each account when scanning the INDEX with CALL 97 statements. For each file add the income to an accumulator for the account to which the file belongs. The accrued charges item in the user's account may be used as this accumulator if desired.

Copyright 1978
Educational Data Systems

8. MISCELLANEOUS INFORMATION

This chapter discusses miscellaneous features not described elsewhere. It also gives additional information of interest only to the system manager on features that are described elsewhere.

8.1 More on LIBRary

A library listing printed for a privilege zero or one user gives only the information of interest to such a user. Additional information is printed when the system manager or other privilege two user requests a library listing. The additional columns are:

TYPE The last three octal digits of the file header's TYPE word. The first digit is the R,L,I control digit described in "How to CHANGE File Characteristics" in the IRIS User Referemce Manual. The other two digits are the file type (see Section 8.7) from which the first column of the library listing is derived.

PRIV The privilege level of the file. This is the same as that of the user who created the file.

HBA Header Block Address. The Real Disc Address of the file's header block in octal.

One additional library command is also available for the system manager. It is

```
LIBR ?          or          LIBR lu/?
```

which will give a summary of disc block usage by privilege level on Logical Unit zero or on a specified Logical Unit.

8.2 House Cleaning Procedures

If the disc is filled near maximum capacity, users will get a message such as "LOGICAL UNIT NEEDS 5 MORE BLOCKS, NOT SAVED" when an attempt is made to save a file. When this occurs, the system manager should "clean house"; ie, he should delete any files that are not currently being used. The procedure for doing this may vary depending on how the system is being used at a particular installation. However, in general, the manager should request a complete library listing by giving the non-selective LIBR@ command. By examining the Hours Since Last Access (HSLA) column of the listing, the manager can determine whether certain files have fallen into disuse. The ACCOUNT column tells whose files they are. The manager may choose to KILL certain files, or he may contact the file's originator to determine if the file is still needed. A LIBR @>h command may also be used to list only those files which have not been accessed recently.

8.3 How to CLEANUP a Logical Unit

CLEANUP is a system processor that will perform file compression and "garbage collection" on a Logical Unit. At the same time, all files and processors will be relocated in a special sequence to optimize system throughput. To clean up a Logical Unit, first back up all disc cartridges, then enter the system command

```
CLEANUP EkeyE x USING y
```

where "key" is the password assigned to CLEANUP (see Section 4.9), x is the LU (Logical Unit) to be cleaned up, and y is the LU on which a scratch file may be built to be used as a map to reorganize unit x. X and y must be different units, and both must be on line before CLEANUP is invoked. The number of disc blocks needed for the scratch file is one block for the header plus the number of total disc blocks on unit x divided by 256. Thus, for a 1000 block unit, the scratch file size would be 1+1000/256 = 5 blocks. For a 32000 block unit, the scratch file size would be 129 blocks.

When CLEANUP is invoked, it first analyzes the input command. Both x and y are verfied that they are on line. All ports are scanned to make sure all users are logged off. The size of unit x is calculated and then the scratch file is built on unit y using the Filename GARBACO. Now the DMAP (Disc Map) on unit x is zeroed out. The INDEX is then scanned on a file type priority basis, and the file's headers are relocated. The new and old disc addresses of the blocks are stored in the scratch file.

After all files have been processed, the scratch file is used to "juggle" the disc blocks into their new positions. When the "juggling" is completed, CLEANUP will "remove" the LU if it is an auxiliary unit, or do an IPL if it is the System Unit (LU #0).

Before running CLEANUP, a system backup should be performed. If a TRAP occurs during the running of CLEANUP, the unit can be considered useless since some of the blocks on the unit may have been moved to their new locations while others have not. If the unit is the System Unit, an IPL probably will not work. In this case, the back up cartridge must be copied back to the original one. Any known bad disc addresses of the unit to be cleaned up should be entered during an IPL or INSTALL sequence to prevent CLEANUP from using a bad disc block which would cause a TRAP during a read or write operation. If Logical Unit zero is being cleaned up, it is a good idea to have at least 100 available blocks on the unit before CLEANUP is invoked; if there are very few blocks available, CLEANUP may not be able to reallocate the active files or DISCSUBS correctly and will produce a TRAP.

Several error messages can occur. These are listed below:

ILLEGAL INPUT - the input command was incorrectly entered.

SAME LU - x and y were not entered as different Logical Unit numbers.

LU NOT ACTIVE - Logical Unit x or y is not INSTALLed.

NOT ALL USERS LOGGED OFF - a port was found that was in use or not logged off.

"GARBACO" FILE ALREADY EXISTS - a file with the same name as the scratch file created by CLEANUP already exists on Logical Unit y.

OUT OF DISC SPACE - there are not enough disc blocks free on Logical Unit y to create the scratch file.

NO "FIXDIRECTORIES" DISCSUB ON THE SYSTEM - the subroutine used to fix the directories of indexed files is not in the DISCSUBS file.

BAD FILE DHDR IN INDEX - a file named in the index did not have a correct header address. (FATAL ERROR !!!)

GAP IN "ACCOUNTS" FILE BLOCKS - the list of blocks in the header of the ACCOUNTS file has a gap in it. (Do an IPL or INSTALL to use the unit).

The time required to perform the juggling of the disc blocks is the longest part of the program. It depends on the type of disc, size, number of files, and how much juggling was required. Typically, a 1K block fixed head drum will take 5-10 minutes, and a 5K block moving arm disc will take 10-20 minutes when 50% full and 1 to 2 hours when 80% full. Once CLEANUP has been run on a Logical Unit, it will take less time the next time it is run since the files will be almost in order instead of in random order.

OPTIMIZED DISC ARRANGEMENT (After running CLEANUP)

Phase	LOC	Filename	Comments
4	0	BZUP	Fixed
4	1	INDEX header	Fixed
4	2	REX header	Fixed
4	3	ACCOUNTS header	Fixed
5		ACCOUNTS	
5		INDEX	
4	LRTC	DMAP	Fixed
4		Nesting Blocks	(Must follow DMAP)
5		INDEX	
6		DISCSUBS, MESSAGES	
7		SCOPE	
7		CONFIG	(Header, first block)
7		SAVE	
7		RUNMAT	
7		BASIC	
7		RUN	
8		Active Files	(Reserve space)
9		LIBR, ASSEMBLE, XASSEMBLE, EDIT, ASGOL	
10		Contiguous & Indexed files	
11		Formatted & Text files	
12		Other Processors	
13		Drivers ($ files)	
14		BASIC Programs, Miscellaneous	
15		CONFIG	(Rest of file)
15		REX	

The phase numbers are printed as CLEANUP runs to indicate progress in the cleanup process. Phases one through three are fast initializing steps, so their phase numbers are not printed. The main cleanup work is done during the following phases:

16	Update file addresses in INDEX
17	Shuffle blocks as determined in phases 5 - 15
18	Correct indexed file directories

8.4 How to Use the System in Stand-Alone Mode

Any machine code file (types "A" and "X" on a LIBR listing) may be used in stand-alone mode. The SHUTDOWN processor will reject any call by a user of privilege 0 or 1 or from any port other than the master terminal. This is to protect the system against inadvertent shutdown, since time-sharing cannot be supported in stand-alone mode. The command will also be rejected if there is a user logged on at any port other than the master, in which case that port's number will be printed.

To initiate use in stand-alone mode, the system manager must type a system command of the form

```
        SHUTDOWN EkeyE Filename
```

on the master terminal (port zero), where "key" is the password assigned to SHUTDOWN, and where Filename is the name of a type A or type X file. On some systems, the shutdown may be allowed by other users and/or from other ports (see Section 2.3.13). SHUTDOWN checks that there are no other users on the system (see Section 5.3), moves itself to the top of core memory, fills all of core below itself with a special halt instruction (077377 octal, which is a DOCP 3,CPU instruction), and finally loads the selected file into core, overlaying some of the halts. If a starting address is specified in the stand-alone program file, then control will be transferred to that location. The starting address may be assigned or changed by use of the CHANGE command as described in Section 8.7. If no starting address is given, the computer will halt (standard 63077 halt) after the file is loaded into core. A 77077 halt indicates that the object file was too big to be loaded, and a 67077 halt indicates a disc read error. In either case, registers A1 and A2 give the disc and core addresses, respectively, of the first block not loaded.

Up to eight stand-alone files can be brought into core together by giving the command in the form

```
        SHUTDOWN EkeyE Filename,Filename,...
```

where each Filename identifies a stand-alone file. All files must be on the same Logical Unit. The files will be loaded into core in the sequence given; therefore, it is best to list the files in order of increasing core locations so that the beginning of one file is not destroyed in core by being overlayed with a partially empty last block of another file. The starting address from the last file having a starting address will be used.

SHUTDOWN includes an absolute binary paper tape loader which will be left at the top of core as defined by TOPW. The binary loader is standard except that each record is checksummed before being stored in core. Also, when a tape is being loaded through the master terminal's reader, the printer on the master terminal will cycle (but not print anything) at the end of each data block (about every four inches). This gives an indication that the tape is being loaded properly.

SHUTDOWN also includes an absolute binary paper tape punch routine which will punch from core to either the high speed paper tape punch or to the punch on the master terminal. To punch a tape, start at TOPW-2 (usually x7775) with switch zero down to select the master terminal (be sure the punch is turned on) or up to select the high speed punch. Leader will be punched, and the computer will halt. Set the first address to be punched in the switches, and press CONTINUE, then put the last address to be punched in the switches, and press CONTINUE again; repeat as necessary if non-contiguous areas are being punched. When all areas have been punched, set switch zero up, and press CONTINUE; only a few frames will be punched. If no starting address is desired, press CONTINUE again (switch zero still up), else set the starting address in the switches (switch zero down) and press CONTINUE; in either case, the proper end block and trailer will be punched.

DBUG, the stand-alone debugging routine (see Section 3.2), may be brought into core along with a stand-alone program by including an @ symbol and an octal address following the Filename. For example, the command

```
        SHUTDOWN EkeyE Filename @6000
```

will bring DBUG into core at location 6000 after loading the selected file or files. All files must be on Logical Unit zero if DBUG is to be brought into core. DBUG will be loaded last if it is requested regardless of its position in the command line, but all files must be on Logical Unit zero to allow DBUG to be loaded.

The paper tape routines and DBUG may also be brought into core without loading a stand-alone program from the disc. To do this, the manager may type a command of the form

```
        SHUTDOWN EkeyE @address
```

which will cause all of the actions described above except loading a stand-alone file into core. DBUG will be brought into core at the specified address, and the computer will halt. Press the CONTINUE switch to start the binary loader, or RESET and START at location x7777. Switch zero must be down to select the master terminal reader and punch or up to select the high speed reader and punch.

It is necesary to do an IPL to bring up IRIS after using the system in stand-alone mode. Usually this may be done by a RESET and START at location 1x7774, where xx7777 is TOPW as defined in the CONFIG file, then set switch one down, and press the ":" (colon) key. If this does not start the IPL then it will be necessary to key in an IPL bootstrap routine as described in Section 5.2.

8.5 System BACKUP Procedure

To do a disc-to-disc copy to back up the system disc, log on to the manager's account on the master terminal, and enter the system command

```
        SHUTDOWN EkeyE DDCOPY
```

where "key" is the password assigned to SHUTDOWN, and press the RETURN key. Then mount a scratch cartridge in Physical Unit 0.1, and RESET and START at location 400 to start the copy process. The entire system disc (on Physical Unit 0.0) will be copied to the cartridge on Physical Unit 0.1.

A normal halt (63077 in the data lights) indicates successful completion; remove the backup cartridge, and press the CONTINUE switch to start an IPL and bring up IRIS.

A 77077 halt indicates a disc time out. A 67077 halt indicates an irrecoverable read error, and a 73077 halt indicates an irrecoverable write error. In any case, register A1 will contain the disc address, and A0 will contain the disc status word. For a disc that is too big for a 16-bit disc address, register A1 will contain the cylinder number, and A2 will contain the track and sector numbers. If the CONTINUE switch is pressed at this time then the copy will be resumed starting with the next cylinder of the disc. Note that a complete cylinder may be lost due to one bad block.

If it is necessary to copy from the backup cartridge to the system disc, the DDCOPY routine must be loaded from paper tape by use of the binary loader. Examine the contents of locations 401 and 402 and interchange the values in these two cells; this will cause the copy to be from Physical Unit 0.1 to Physical Unit 0.0. Put the backup cartridge in Physical Unit 0.1, then RESET and START at location 400. When finished copying, replace the backup cartridge with a blank cartridge or another Logical Unit, and press the CONTINUE switch to start an IPL and bring up IRIS.

8.6 Real-Time Interrupt System

The interrupt service routine in REX provides for true real-time interrupt handling. When an interrupt occurs, an interrupt acknowledge instruction determines the highest priority interrupting device, and a mask is picked up from the device's interrupt handler and ORed with the current interrupt mask to disable that device's interrupt and any lower priority interrupts. Control is then passed to the device's interrupt handler with interrupts enabled, thus allowing interrupts from higher priority devices. Notice that the determination of priority is under control of the programmer writing the device driver rather than arbitrarily assuming that a device has higher priority if its mask bit is to the left of another device's mask bit. When the driver returns from the interrupt, the previous mask and all registers are restored, and control is returned to the interrupted task. An interrupt handler may create a task of higher priority than the regnant task, in which case the new task goes at the head of the queue, and returning from the interrupt "restores" to that task.

8.7 More on CHANGE and SCOPE

The CHANGE processor is described in "How to CHANGE File Characteristics" in the IRIS <u>User Reference Manual</u>. There are four additional features, however, for manager level users.

First, CHANGE will print

```
R, L, AND I CONTROL = x
NEW CONTROL DIGIT?
```

The R, L, and I control digit is the third digit from the right in the file's TYPE word, and it is made up of three bits as follows:

Value	Bit	Meaning
R=4	8	Runnable processor
L=2	7	Load active file
I=1	6	Initiate input

This feature exists so that a processor may be assembled using ASSEMBLE and then changed to a runnable processor to be used on the system. CHANGE will allow the R bit to be set only if:

1) The first address in the processor is 200, and

2) The processor will not overlay locations 600 through BPS-1, and

3) The file type is 1, 3, or 4 (system, stand-alone, or executable file).

When a Filename is entered in response to the # prompt character, SCOPE reads the file's header and examines the control digit. If the R bit is set then the file is opened on channel -1 (minus one) and thus becomes the port's selected processor; SCOPE then examines the L and I bits. If the L bit is set then SCOPE looks for another Filename following the processor's Filename (eg, BASIC ACCOUNTLIST). If such a Filename is found then its header is read, and the file type compared with the type of the processor. A match causes SCOPE to load the program file into the port's active file. SCOPE then checks the I bit whether L was set or not, and enables input if I is set. When input is terminated (or immediately on the next time slice if I was not set) the selected processor is given control at its initial entry point. If the R bit is not set then SCOPE examines the file type. For a BASIC program, and for certain other types of files, SCOPE selects the proper processor; eg, "RUN Filename" is (in effect) substituted for "Filename" if Filename identifies a BASIC program.

Second, if the R bit is set, CHANGE will print

```
        PROCESSOR TYPE = x
        NEW TYPE?
```

This allows changing the file type (last two digits of the file's TYPE word) to match a processor with its program files. The disc files on an IRIS system can be grouped into several general catagories as described under "Disc Files" in the IRIS User Reference Manual. More specifically, however, the least significant five bits of each file's TYPE word define the actual file type for classification purposes. Type zero indicates a permanent system file, type one indicates a system utility file, a system device file, or a system command processor, and types two through 17 (octal) are used to match program files to respective application processors as described above. Types 20 through 27 (octal) are used for special files such as the relocatable image files, and types 30 through 35 (octal) are used for various types of data files. Type 36 (octal) is used for peripheral device drivers such as $LPT. Type 37 is not used to identify any particular type of file but is used as a "wild card" to allow opening any file of type 30 through 36.

Third, if the R bit is not set, CHANGE will print

```
        STARTING ADDRESS = x
        NEW STARTING ADDRESS?
```

If the file is now a type 3 or 4 then a new starting address may be entered in octal. The starting address may range from 0 to 77777 inclusive (1xxxxx indicates no starting address). Processors do not require a starting address; the system transfers control to a processor starting at BPS+4 (the forth location after Beginning of Processor Storage, which is defined in the IRIS Software Definitions).

Finally, if the file is a BASIC program, CHANGE will print

```
        PROGRAM IS NOT LOCKABLE
        LOCKABLE OR NOT (L OR N)?
```

or the first line may say PROGRAM IS LOCKABLE. In either case, no change will be made if only the RETURN key is pressed, or either L or N may be entered to set or reset the "lockable program" flag (bit 5 of the TYPE word of the file's header).

8.8 More on INSTALL

Refer to "How to INSTALL a Logical Unit" in the IRIS User Reference Manual for procedures on INSTALL. For the system manager, INSTALL provides four additional features.

If the Physical Unit is not formatted (ie, it has never been installed as an IRIS Logical Unit), or if it is desired to change the Logical Unit number of an existing unit, then the system manager may give the command to

```
        INSTALL d.p
```

and press the RETURN key. If the unit is not formatted, then the manager will be asked "DESIRED LOGICAL UNIT NUMBER?", and the unit will be formatted with the Logical Unit number given.

If the unit has been previously installed, then the current Logical Unit number will be printed, and the manager will be asked "INSTALL (Y/N)?". A "Y" response installs the unit as its current number. An "N" response will cause INSTALL to ask "CHANGE LOGICAL UNIT NUMBER (Y/N)?"; an "N" response will cause INSTALL to exit to system command mode. If a "Y" response is given then the manager will be asked "NEW LOGICAL UNIT NUMBER?", and if a number is entered then the unit will be installed with the new Logical Unit number without losing any files on the disc (see note below), and all further references to the unit must use the new number. For example:

```
        #INSTALL 0.1
        LOGICAL UNIT NUMBER = 4
        INSTALL (Y/N)?  N
        CHANGE LOGICAL UNIT NUMBER (Y/N)?  Y
        NEW LOGICAL UNIT NUMBER?  36
```

When the installation is completed, any file on the unit must be addressed as 36/Filename instead of as 4/Filename.

Note: if a copy of Logical Unit zero is installed by changing its Logical Unit number, all driver files (Filenames starting with a dollar sign) will be deleted. These files may be saved by first changing their names to omit the dollar sign (eg, MMUX instead of $MMUX).

The third additional feature allows the system manager to give the command

```
        INSTALL AND CLEAR d.p
```

which will clear the disc at location drive d, partition p, thus formatting a new Logical Unit. If the unit at location d.p is an IRIS Logical Unit, then INSTALL will ask for confirmation that the unit is to be cleared. INSTALL then asks for the desired Logical Unit number and formats a new (empty) Logical Unit.

The fourth additional feature allows the system manager to give the command

```
        INSTALL FAST d.p
```

if cell 201 of INSTALL is non-zero. Cell 201 normally contains zero, disallowing fast install; a 1 in location 201 allows only the manager to do a fast install, and any larger value allows anyone to install fast. The fast install is recommended only when changing disc packs during normal system operation since the housekeeping operation is bypassed; a normal install is strongly recommended in all other cases to build a new disc map.

8.9 How to Set System Time

The system time resides in the HRS (hours since 1-1-76) and the TSC (part of hour in tenth-seconds) cells in the system INFO table. The TSC cell is incremented each tenth of a second. When TSC reaches 36000 (decimal) it is reset to zero, and HRS is incremented. This counter can run for 2↑16 hours (approximately 7.5 years) before an overflow will occur. If the system time must be adjusted it may be done by the system manager by use of a CALL 99 in a BASIC program. For example:

```
        10 DIM A$[25]
        20 INPUT A$
        30 CALL 99,A$
```

This program will work only from the manager's account, and the string entered for A$ must represent the current time. The time may be represented as described in Section 5.1, or it may be of the form

```
        FEB 15, 1977  10:22:36
```

where the "seconds" portion (the last colon and last two digits) is optional. All leading zeroes and the first two digits of the year are also optional. CALL 99 may also be used to read the current system time (see SETTIME, example program one in Appendix I of the IRIS User Reference Manual).

8.10 More PORT Control Functions

Any user on an interactive port may change certain attributes of his own port as described under "How to Change BAUD Rate" and "How to Change Other PORT Characteristics" in the IRIS User Reference Manual. The system manager may direct any of those same port change functions to any active port by giving a port number after the word PORT. For example, the manager may give the command

```
        PORT 3 BAUD 110
```

to change the rate at port three to 110 Baud. This particular command is useful when a user has changed his port's speed to a rate that is not available on his terminal. In addition, the manager may give the word ALL in place of a port number. For example, the command

```
        PORT ALL DELAY 12
```

would cause a twelve character delay following a carriage return on all ports (0.12 seconds delay for any ports on an EDS-8 multiplexer).

In addition to the ability to direct a PORT command to any specified port or to all ports, the manager may cause a port to be logged off regardless of what the user on that port is doing. This is done by giving the command

```
        PORT EkeyE n EVICT
```

where "key" is the password assigned to PORT (see Section 4.9) and n is a port number that is to be logged off. All users may be logged off at once by giving the command

```
        PORT EkeyE ALL EVICT
```

which is a useful command for use before shutting down the system to back up the discs. The manager may also determine what is going on at a particular port by giving the command

```
        PORT n MONITOR
```

which will cause a print out of the account number of the user logged on to port n, the current Baud rate of that port, the processor currently in use by that port, and the program file (if any) being used on that port. All ports may be monitored by giving the command

```
        PORT ALL MONITOR
```

which will list all of the above information about each active port on the system. See Section 2.3.19 if it is desired to allow users other than the system manager to use PORT MONITOR.

8.11 Priority Task Queuing

A task is put on the priority task queue by the instruction

```
QUEUE
Task
TCB pointer
Initial priority
(Unused word)
Register A3
```

where "Task" is either the name of a system task defined in the Software Definitions (see Section 8.14) or the core address of a task's entry point. The TCB pointer word may be zero if no Task Control Block is used by the task, or -1 if the Regnant Task Pointer (RTP) is to be used. The Task Control Block is usually a Port Control Block (PCB). The TCB pointer is put in RTP when the task is given control.

The carry bit, registers A0, A1, and A2, and the "register A3" parameter are saved in the task node (see Figure 8.1), and all of these may be used to pass parameters to the task. Upon return from QUEUE, register A2 points to the new queue node.

CAUTION! The initial priority of the task being queued must not exceed the current priority of the task which calls QUEUE. An interrupt handler may queue a new task with initial priority up to 1777 octal.

The task being queued must be core-resident, and it must have a flag word in the cell preceding its entry point. The lower 14 bits of the flag word must be zero, and the top two bits are used as flags as follows:

Bit	Meaning
15	possible task conflict
14	task is a processor

Figure 8.1: Task Queue Node

```
           ---------------------
        0  ! Register A2       !  <-- Next higher priority queue
           !-------------------!      node's link pointer (or
        1  ! Register A1       !      TASKQ if this is highest
           !-------------------!      priority task) points to
        2  ! Register A0       !      cell zero of this node.
           !-------------------!
        3  ! Register A3       !
           !-------------------!      (See below for a discussion
        4  ! Address, carry bit !      of the use of each word in
           !-------------------!       the node.)
CPRI =  5  ! Current Priority  !
           !-------------------!
        6  ! Spare word        !
           !-------------------!      LINK points to the next
TASK =  7  ! Task Pointer      !      lower priority queue node.
           !-------------------!      The link pointer in the
TCBP =  8  ! TCB Pointer       !      lowest priority queue node
           !-------------------!      points to the fixed idle
LINK =  9  ! Link Pointer      !  <-- task node of priority zero.
           ---------------------
```

Words 0-3 Used to pass parameters to a task when it is created, and to save the registers if the task is interrupted.

Word 4 Holds the task starting address when the task is created, and is used to save the return address if the task is interrupted. The address is shifted left one bit, and the carry bit is saved in bit zero of the word.

Word 5 (CPRI) Holds the current priority of the task.

Word 7 (TASK) Points to the entry point of the task. Task attributes are at negative displacements from the entry point.

Word 8 (TCBP) Points to the Task Control Block (TCB) of the user for which the task is to be performed, or it is zero for a system task.

Word 9 (LINK) Points to the next lower priority task node. This word is zero in the idle task, which ends the queue.

8.12 Dequeuing a Task

When a task has been completed it must remove itself from the task queue by executing the instruction

DQUEUE

There is no return from a DQUEUE instruction. The regnant task node is removed from the task queue and released to the free node chain, and control is given to the next task on the queue. The queue is never empty; the idle task (priority zero) is always at the end of the queue and never dequeues itself.

Figure 8.2: Task Priorities

2021	Power fail interrupt handler.
2020 2001	Device interrupt handlers. These priority levels are determined by the interrupt mask.
2000	Character processing (in REX).
1777 1600	System tasks that do not make disc accesses may have any priority from 1400 to 1777. The Ten Hertz task is usually priority 1602.
1577 1401	System tasks which do make disc accesses may have any priority from 1401 to 1577.
1400	Task to queue a processor, and processor escape and signal tasks.
1377 0001	User tasks range in priority from 1 to 1377 and usually have dynamic priority. Tasks having priority below 100 are considered to be background. See Section 2.5 for details.
0000	Idle task has priority zero.

8.13 Form of a Task

A task is a core-resident assembly lanquage program written in the following form:

```
        x00000  ;FLAG WORD
NAME:   xxx     ;ENTRY POINT
          .
          .     ;BODY OF TASK
          .
        DQUEUE  ;TASK COMPLETED
```

The flag word must be in the word preceding the task's entry point. Bit 15 of the flag word must be set to one if the task does any disc access, if the task is not re-entrant, or if it may conflict with other tasks in any other way; otherwise bit 15 should be zero. Bits 14-0 must be zero for all tasks except the system's processor task which has a one in bit 14. The body of the task may include calls to any re-entrant system subroutines (not to DISCSUBS unless bit 15 of the flag word is set), and it may queue other tasks for work that is of lower priority than itself. The entry point (NAME in the above example) is used in the QUEUE instruction to queue the task, and its priority is determined at the time it is queued (see page 8-13). At the time the task is entered, reqisters A0, A1, A2, and the carry will contain the same values as when the QUEUE instruction was given, and A3 will contain the "Register A3" value given at that time.

8.14 Available System Tasks

Two system tasks are available at this time. They are:

The Ten Hertz task (TENHZ)
The Signal task (SIGNAL)

TENHZ is to be queued only by the real time clock driver, and there must be only one such driver on the system. TENHZ is queued at priority 1602 once each tenth of a second with a pointer to INFO in register A3.

SIGNAL may be queued by any task wishing to send a signal. It must be queued at priority 1400 with the sender's PCB pointer as the task's TCB pointer. Register A0 must contain the destination PCB ponter, A1 the first signal value, and A2 the second signal value. A3 is iqnored. If the signal buffer is full then a bell is echoed to the sender's port.

9. HOW TO WRITE A PROCESSOR

A processor is a machine language program written in a specific configuration for proper interaction with the Real-time Executive (REX). Each IRIS system command, such as SAVE, LIBR or BYE, is executed by a processor. Likewise, the user languages and services, such as BASIC, TUTOR, EDIT, and ASSEMBLE are provided by processors. New processors may be added to IRIS at any time by the system manager, either by using PLOAD to load a tape provided by Educational Data Systems, or by assembling it on the disc. Other users may write a processor, but only the manager can make it accessible as a system command. This section provides all information necessary to write a new processor, add it to the system, debug it, and make it accessible for general use.

9.1 Core Locations and Entry Points

Core locations 200 through 577 (octal) and locations BPS through MBUS-1 are available for use by a processor (see Figure 9.1). BPS (Beginning of Processor Storage) is defined in the Software Definitions tape, and MBUS (Minimum Beginning of User Storage) is in the INFO table along with many other pointers (refer to a listing of the Software Definitions). The available space may be used as desired by the processor, except that cells BPS through BPS+3 are reserved for pointers to the "swap-in", "swap-out", "escape" and "CTRL C" routine entry points, and the "initial entry" is at location BPS+4. The processor occupies the page zero area 200-577 and additional space starting at BPS, and it may use the space between the end of the processor and MBUS for temporary data storage. A user partition and the port's active file must be used for program and data storage which is to last longer than one time slice (see Sections 9.3 through 9.5). The disc block buffer areas (BSA, HBA, HXA, and ABA) may also be used if certain restrictions are observed.

The location of the user's I/O buffer and data file table, as well as other information, can be found in his PCB (Port Control Block) via RUP (the Regnant User Pointer) in REX page zero. RUP is set by the system before swap-in to point to the current user's PCB. Refer to a listing of the IRIS Software Definitions for more information about the Port Control Blocks.

Two each of the auto-increment cells and auto-decrement cells are available in page zero for use by processors. The auto-increment cells are labeled AI1 and AI2, and the auto-decrement cells are labeled AD1 and AD2. These cells are not saved between time slices unless they are copied into the user area by the processor's swap-in and swap-out subroutines.

Figure 9.1: Processor Core Locations

```
; THIS IS THE FORM OF A PROCESSOR FOR "IRIS"


.TXTM   1          ;REQUIRED FOR CORRECT PACKING OF TEXT
.LOC    INFO-400 ;ALL PROCESSORS MUST START AT 200

; CELL 200 MUST CONTAIN AN ASSEMBLED VALUE.
; (DO NOT START WITH A .BLK OR ANOTHER .LOC)

; USE LOCATIONS 200-377 FOR CONSTANTS, POINTERS, ETC.

; LOCATIONS 400 THROUGH 577 MAY BE USED FOR SWAP-IN AND
; SWAP-OUT SUBROUTINES OR FOR ANY OTHER DESIRED PURPOSE.

.LOC    INFO-. ;PAGE ZERO OVERFLOW TEST


.LOC    BPS    ;BEGINNING OF PROCESSOR STORAGE

        SWPI   ;POINTER TO SWAP-IN SUBROUTINE
        SWPO   ;POINTER TO SWAP-OUT SUBROUTINE
        ESCR   ;POINTER TO ESCAPE ROUTINE
        CTLC   ;POINTER TO CONTROL C ROUTINE

; INITIAL ENTRY IS AT BPS+4.

    .

    .

    .

.END    ;END OF THE PROCESSOR


; ALL REMAINING SPACE UP TO MBUS MAY BE
; USED BY THE PROCESSOR AS DESIRED.

; SOME SPACE SHOULD BE RESERVED IN THE PROCESSOR
; AREA FOR PATCHES DURING DEBUGGING PROCEDURES.
```

9.2 Sequence of Events

The sequence of events in a processor's operation is as follows:

1) A # symbol is printed by SCOPE (the System Command Processor) as a system prompt character. The user types in a command which consists of a processor's Filename and may include additional elements such as the Filenames of program files or text files or other information required by the processor. In some cases, SCOPE will process one such element (see Sections 8.7 and 9.9). In any case, SCOPE finds and selects the desired processor and loads the initial entry address (BPS+4) into the URA (User's Return Address) cell of the user's PCB (Port Control Block).

2) At the next time slice the system loads the selected processor into core if is not already in core, inserts a breakpoint jump if a DSP breakpoint has been set in this processor on this port, and does a JSR to the processor's swap-in routine via the pointer in location BPS. The swap-in routine must perform any intializing required (see Section 9.4) and return to the system, which in turn jumps to the address in URA. The initial entry (first time in since the command was given) will be at BPS+4 as set up by SCOPE. Subsequent entries will resume operation (after the JSR to "swap-in") at the point where execution terminated in the preceding time slice.

3) The processor performs its intended functions until its time slice is terminated for one of the following reasons:

   a) Processor starts input (STINPUT)
   b) Processor wants to do output and it already has an output in progress (CALL WONA)
   c) End of time slice (BUMPUSER due to RTL <= 0)
   d) A record locked condition is encountered in a READ or WRITE (BUMPUSER following a CALL SIGPAUSE)
   e) User presses ESC or CTRL C (see text)
   f) Processor completes or aborts its task (CALL SCOPE)
   g) The processor or a subroutine detects a hardware or software error (TRAPFAULT)
   h) A DSP breakpoint is encountered

   Any of the first four conditions will cause the return address to be saved in URA for re-entry at the next time slice. Any of the last three conditions will cause this to be the last time slice for this processor. Conditions f or g will select SCOPE as the user's processor. Condition h will select DSP and cause the registers, carry bit, and a 65-word area of core to be saved in the DSPS cells of the active file header for print out by DSP's U command. See BUMPUSER on page A1-16 for the proper manner of determining end of time slice.

The actions of the ESCAPE and CTRL C keys depend upon the current state of the processor as follows:

a) If the processor is in core for this user at the time ESC or CTRL C is pressed then the only immediate action is to terminate any output in progress and set the escape flag (ESCF in REX page zero). The processor should periodically check ESCF; if (ESCF) is non-zero then the processor should clear it and take whatever action is appropriate. If the escape flag is ignored by the processor then re-entry for the next time slice will be at the escape entry.

b) If the processor is not in core for this user at the time then the system's action consists of saving the URA address in ORA (Old Return Address) and setting URA to transfer control to the processor for the next time slice (after the JSR to "swap-in") via the pointer in location BPS+2 or BPS+3 for ESC or CTRL C, respectively.

4) After the time slice is terminated for any of the above reasons (except a TRAPFAULT or a DSP breakpoint) the system does a JSR to the processor's "swap-out" routine via the pointer in location BPS+1. The swap-out routine must perform any wrap up required to save information for the next time slice (see Section 9.5).

9.3 Use of Active File

The active file is a special file on the system disc reserved for interim storage of a processor's data between time slices. There is an active file associated with each interactive port. The size of each active file is usually configured to be the partition size, including a block for its header. REX provides facilities to read in and write out the active file; however, the processor's swap-out routine must define how much is to be written out.

The processor may not need to use the active file if it has little or no interim storage to save between swaps. If the processor has no interim storage requirements then it merely has pointers to a JMP 0,3 instruction in cells BPS and BPS+1 (see Figure 9.2). If the processor has 101 (octal) or less cells of interim storage required, it may use the FMAP cells in the active file header for interim storage. The processor must copy the interim storage itself, and it may have to read the active file header into HBA and write it out if it chooses this method. See Figure 9.3 for a programming example. The active file header's Real Disc Address is contained in AHA of the regnant user's PCB.

If a processor has more than 101 (octal) words to save between swaps then it must use a partition and the active file; see Figure 9.4 for a programming example.

9.4 Swap In Procedure

Each time a user's time slice begins, RUP is set to point to the user's PCB, the selected processor is brought into core (unless it is already in core), BBA is zeroed, and the system does a JSR to its swap-in routine via the pointer in location BPS with zero in register A1 and (RUP) in register A2. If the active file and or its header are used for storage between time slices then the swap-in routine must get it into core and perform any other initializing required. A "load user" subroutine is provided and may be reached by a CALL LOADUSER instruction sequence with zero in A1. LOADUSER selects a user partition, puts a pointer to that partition's entry in the partition table in RUS, puts a pinter to the beginning of the partition in BBA, and reads the active file header into the beginning of the partition. If the active file's type (lower five bits of TYPE word) matches the processor, the active file is also read into the partition, and LOADUSER does a skip return. LOADUSER also restores the six DA (Decimal Accumulator) cells and any save area specified to AFSETUP at last swap-out (see Section 9.5). LOADUSER does a non-skip return if the types don't match. Alternatively, the swap-in routine may read the active file header itself, may read any other file or header into core as required, or may be simply a JMP 0,3 instruction if no initializing is required.

In some cases the swap-in routine must know whether this is the first or a subsequent time slice. This can be determined by comparing the value BPS+4 with the address in URA; equality indicates that this is the initial entry.

9.5 Swap Out Procedure

After each time slice is terminated for any of the reasons given in Section 9.2 (except a fault or a breakpoint) the system will do a JSR to the processor's swap-out routine via the pointer in location BPS+1 with RUP in register A2. If no wrap up is required then location BPS+1 may point to a JMP 0,3 instruction; otherwise, the swap-out routine must save all information necessary for the next time slice. The swap-out routine may do a skip return (eg, JMP 1,3) if the partition is to be released without writing it to the active file. Typically, the swap-out routine will either:

a) Copy a temporary storage area from page zero into the FMAP through FMAP+100 cells of the active file header and write out the active file header (see Figure 9.3), or

b) Call AFSETUP (see below), which will copy the DA cells from page zero to the DASA cells in the active file header, will copy up to 101 (octal) temporary storage cells from anywhere in core to the FMAP cells of the active file header (optional), and will set ABLK in the active file header to the number of active blocks needed to swap out the active data in the user's partition.

A non-skip return to the system from the processor's swap-out routine will cause the system to retain the partition (if one is assigned) and to write out the active file if and when the partition must be assigned to another user. A skip return causes the partition to be released immediately without writing it to the port's active file.

The active file is a random file normally containing DSAF blocks, including the header, where DSAF (Default Size of Active File) is defined in the attributes table of the interactive port driver. The active file header contains the Real Disc Address of each of these blocks, and each cell in the last half of a header is "wired" to a particular core address relative to CORA as described in Section 1.4. Also, any number of these disc addresses may be "active" as specified by the ABLK cell in the active file header.

The AFSETUP subroutine should be used to do the setup of the active file header. Its calling sequence is CALL AFSETUP with register A0 containing a pointer into the user's partition to the last word to be swapped out, register A1 containing the beginning address of an area of core of up to 101 words octal to be saved in the active file header's FMAP cells (or zero if no temporary storage cells are to be swapped), and A2 containing the size of the temporary storage area (#words-1). The starting address and size indicator are saved in DASA+6 and DASA+7 of the header, or DASA+6 is zeroed and A2 is ignored if no save area is specified. In any case, the Decimal Accumulator (DA) is copied to the first six DASA cells.

Copyright 1978
Educational Data Systems

AFSETUP will set CORA to the first data address of the regnant user's partition (equal to ((RUS))+400) and will set the ABLK cell in the header so that the correct number of blocks will be transferred equal to INT[(AO)+377-((RUS))/400]. ALLOCATE will be called by AFSETUP if the active file is too small to hold the portion of the partition to be swapped. If the processor is either type 0 or type 1, then the active file header's TYPE word is set equal to the processor's type. In most other cases, the processor's swap-out routine must set the type word to match itself after calling AFSETUP.

AFSETUP will do a non-skip return unless the active file is too small and not enough additional disc blocks are available to be allocated from the system account, in which case AFSETUP will do a TRAPFAULT (trap #13).

Figure 9.2: Processor With No Swapping

```
; THIS PROCESSOR PRINTS "I AM A PROCESSOR" REPEATEDLY


        .TXTM   1
        .LOC    INFO-400

CR:     215
J03:    JMP     0,3

CEXIT:  CALL
        SCOPE


        .LOC    BPS      ;ENTRY POINTERS

        J03     ;SWAP-IN (NONE)
        J03     ;SWAP-OUT (NONE)
        CEXIT   ;ESCAPE
        CEXIT   ;CONTROL C


XYZ:    LDA     2,CR     ;STORE A RETURN CODE
        OUTBYTE
        OUTTEXT          ;STORE THE MESSAGE
        .TXTF  "I AM A PROCESSOR"
        STOUTPUT         ;START OUTPUT
        CALL
        WONA
        JMP     XYZ      ;REPEAT


.END    ; "I AM A PROCESSOR" SOURCE
```

Figure 9.3: Swapping Storage in Active File Header

```
SWPI:  STA    3,SWPR          ;SWAP-IN SUBROUTINE
       LDA    1,AHA.,2
       LDA    3,DFT.,2
       LDA    3,CBN+CHM1,3 ;PARTITION TABLE ENTRY POINTER
       LDA    2,0,3           ;POINTER TO BUS IF PARTITION ASSIGNED
       SKZ    3,3             ;IS A PARTITION ASSIGNED ?
       JMP    SWPI2           ;  YES, GET SAVED DATA FROM PARTITION
       SUB    0,0             ;  NO, READ ACTIVE FILE HEADER
       LDA    2,.HBA
       READBLOCK
SWPI2: LDA    0,DFMAP         ;DISPLACEMENT TO FMAP CELLS
       ADD    2,0
       LDA    1,LENSV         ;LENGTH OF SAVE AREA - 1
       ADD    0,1
       LDA    2,.BSAV         ;BEGINNING OF SAVE AREA IN CORE
       CALL
       MOVEWORDS              ;MOVE FMAP CELLS TO SAVE AREA
       JMP    @SWPR

SWPR:  0      ;RETURN ADDRESS
DFMAP: FMAP   ;DISPLACEMENT TO FMAP CELLS
.BSAV: [Beginning of core area to be saved]
LENSV: [Length of area to be saved - 1]
TS:    0      ;TEMPORARY STORAGE


SWPO:  STA    3,SWPR          ;SWAP-OUT SUBROUTINE
       LDA    1,AHA.,2
       LDA    3,DFT.,2
       LDA    3,CBN+CHM1,3
       LDA    2,0,3
       STA    3,TS
       SKZ    3,3             ;IS A PARTITION ASSIGNED ?
       JMP    SWPO2           ;  YES, USE IT
       SUB    0,0             ;  NO, READ ACTIVE FILE HEADER
       LDA    2,.HBA
       READBLOCK
SWPO2: LDA    0,DFMAP         ;MOVE SAVE AREA TO FMAP CELLS
       ADD    0,2
       LDA    0,.BSAV
       LDA    1,LENSV
       ADD    0,1
       CALL
       MOVEWORDS
       LDA    3,TS
       SKZ    3,3             ;IS A PARTITION ASSIGNED ?
       JMP    @SWPR           ;  YES, DONE
       SUB    0,0             ;  NO, MUST WRITE ACTIVE FILE HEADER
       LDA    2,RUP
       LDA    1,AHA.,2
       LDA    2,.HBA
       WRITBLOCK
       JMP    @SWPR
```

Figure 9.4: Swapping With Active File

```
        .LOC    INFO-400

PTYPE:  1       ;PROCESSOR TYPE

TS:     .BLK    20              ;TEMPORARY STORAGE
SP:     0                       ;STACK POINTER


SWPI:   STA     3,SWPO-1        ;SET UP AFTER SWAP-IN
        CALL                    ;LOAD USER'S ACTIVE FILE
        LOADUSER
        JMP     SWPI1           ;  FILE TYPES DON'T MATCH
        JMP     @SWPO-1         ;RETURN TO SYSTEM

SWPI1:  SUB     0,0             ;INITIALIZE (FIRST SWAP-IN)
        STA     0,TS
        STA     0,TS+1
        STA     0,TS+4
        STA     0,TS+6
        STA     0,TS+12
        LDA     2,@RUS
        LDA     3,C400          ;INITIALIZE STACK POINTER
        ADD     2,3
        STA     3,SP
        STA     0,COST,2        ;CLEAR COST OF ACTIVE FILE
        STA     0,NAME,2        ;ALSO CLEAR THE NAME
        LDA     3,PTYPE
        STA     3,TYPE,2        ;SET TYPE TO MATCH PROCESSOR
        LDA     3,RUP
        LDA     1,AHA.,3        ;WRITE OUT NEW HEADER
        WRITBLOCK
        JMP     @SWPO-1         ;RETURN TO SYSTEM


        TS
        17
        0
SWPO:   STA     3,.-1           ;WRAP UP FOR SWAP-OUT
        LDA     2,SWPO-2        ;SIZE OF TEMP STORAGE - 1
        LDA     1,SWPO-3        ;POINTER TO TEMP STORAGE AREA
        LDA     0,SP            ;LAST WORD IN PARTITION
        CALL
        AFSETUP
        JMP     @SWPO-1
```

9.6 Use of System Subroutines

All system subroutines listed in Appendix I may be used by a processor. The most commonly used subroutines are also described elsewhere in this manual. If the active file is used, or if the processor uses the disc block buffer areas for other purposes, then the programmer should be especially watchful for possible conflicts in the use of these areas. Also, it is illegal to use HBA for anything other than a file header block.

9.7 Input and Output

All I/O is via a one-line buffer for each port. Pointers in each port's PCB determine the location of the buffer and the next character position. It is illegal for a processor to examine or modify the I/O pointers. System subroutines are provided for all required I/O functions as follows:

Start Input is called by a STINPUT instruction. The user is bumped and input is enabled. The processor will be swapped in and control returned to the next instruction with (RUP) in A2 after the user presses a RETURN to terminate input. The processor may change the EOM character from RETURN to any desired code by putting that code in the lower half of the RDE cell in the PCB. STINPUT may be issued even when output is active.

Access Input Byte is then called by an INBYTE instruction to access each byte of input. The byte is returned in A2 with the top bit of the ASCII code set to "one" and zeroes in the top half of the register. Space codes (octal 240) are ignored. The EOM code (usually a RETURN, octal 215) indicates end of input.

Access Input String Byte which is called by an INSTBYTE instruction, is the same as INBYTE except that no characters are ignored. Every character typed by the user will be given to the processor. If A0 is zero when INSTBYTE is called then the byte pointer is not incremented, and the same byte will be again accessed by the next use of INBYTE or INSTBYTE.

Wait for Output Not Active must be called by a CALL WONA instruction sequence before the first use of any of the following output routines unless it is already known that output is not active (eg, on initial entry or following a Start Input). WONA will bump the user if an output is already in progress. This allows computation to continue during an output, but prevents a second output from overlaying one that is in progress. The processor will be swapped in and control returned to the next instruction with (RUP) in A2 after the output is completed.

Copyright 1978
Educational Data Systems

Output Byte is called by an OUTBYTE instuction to store the byte in the lower half of register A2 into the user's output buffer. The byte is also returned in A0 with the top half of the word zeroed. If the buffer is filled then OUTBYTE overlays the last byte in the buffer rather than incrementing the pointer beyond the end of the output buffer.

Text Message Output is called by an OUTTEXT instruction followed by a .TXTF "text" pseudo-op. Copies any given "text" string to the user's output buffer and returns to the next instruction following the text. Neither OUTBYTE nor OUTTEXT will increment the Output Byte Pointer if the byte being stored is a zero.

Canned Message Output outputs any available "canned" message whose number is given in register A1. Appendix 2 of this manual gives the calling sequence for MESSAGE, and lists the currently available messages.

Convert Integer to ASCII is called by a CALL CIA instruction sequence with an integer to be output in A1. Register A0 must contain the radix to which it is to be converted, and A2 must contain the minimum number of digit positions desired. Leading zeroes are suppressed, and the result is padded with leading spaces for a total of (A2) characters. Set (A2)=0 for no leading spaces. Letters will be used as digits if the radix exceeds ten; ie, A for ten, B for eleven, etc.

Start Output is called by a STOUTPUT instruction after using the above routines in any combination to store ASCII codes in the user's output buffer. The string of ASCII codes must be terminated by a zero byte by clearing A2 and executing an OUTBYTE before starting output (this is not necessary if the last output was generated by an OUTTEXT or a CALL MESSAGE).

All of the above routines destroy the contents of all registers except as noted in the subroutine description.

9.8 Data File Access

Data file transfers are handled by the processor via several system subroutines. These subroutines provide facilities for opening existing files, creating new files, and deleting files. A processor may access and update data files via system calls or may access data directly by use of the read block and write block subroutines.

Nearly all file access is done via channels. Channels allow the system to guarantee that a file will not be deleted by one user while being accessed by another. Channel I/O also allows devices to appear as data files to the processor, thereby requiring no changes to the application software when a device is added to the system.

A file may be opened on a channel in any of four ways. CHANNEL OPEN will open a Filename on a channel passed to it. If the file is not the regnant user's and there is a charge for it, the regnant user will be charged. If the file is write protected, this information is retained in core and the user will receive an error if he attempts to write to the file. The LDAT cells will be set equal to the current value in the system clock, and the NTAC cell will be incremented. CHANNEL OPENREFERENCE will open the file for reference only; the regnant user will not be charged for its use, the LDAT and NTAC cells will not be changed, and the channel will be marked as write protected. CHANNEL OPENUPDATE will do the same as OPEN except it will error if the file is write protected. CHANNEL OPENLOCK will do the same as OPEN except that all other users will be locked out of the file, but an error will be indicated if another user already has the file open.

A new file is created via a CHANNEL BUILD system call. The file Filename is built on the requested channel. Errors are provided for illegal Filename, out of disc space, etc. If the Filename exists, BUILD will mark the old file as being replaced only if the new name is of the form "Filename!" and both the types and account numbers are the same; otherwise, an existing file will not be replaced. The new file will be marked as being built until it is CLOSEd. Any new file being created must be closed by the processor by issuing a CHANNEL CLOSE instruction before exiting to the system. If the channel is CLEARed before it is CLOSEd then the file being built will be lost, and if the new file was replacing an old file then the old file will be restored to normal status. Any particular channel may be cleared (its contents aborted) by a CHANNEL CLEAR call. All channels may be cleared by a CALL ALLCLEAR system command. ALLCLEAR is seldom used by a processor since the system clears all channels after the processor exits.

Copyright 1978
Educational Data Systems

If the processor wishes to delete a file, it issues a CALL DELETE system call. DELETE will check to see if the file is open by any user. If it is not open, DELETE will credit the owner's account and release the disc blocks to the system. If the file is in use, it will be marked to be deleted and will be deleted when the last user has CLOSEd or CLEARed the channel where it is open.

Data may be transferred to a file in either of two ways. A highly structured means of transferring data to and from files is CHANNEL READITEM and WRITEITEM. They will read (write) from (to) a particular file location to (from) a supplied core address. If a device has been opened on the channel, the system will automatically cause the data to be tranferred to the device so that the processor does not need to recognize devices as being different from files.

For faster access, a processor might determine the data block of a file directly from the file's header and then use RBLK or WBLK to transfer 256 (decimal) words from or to the file.

A processor may use the system calls FOFI and FOFC (Find Open File Initialize and Find Open File Continue) to determine if a given file is open by any user. A processor may determine whether there is any file open on a given Logical Unit by supplying zero for the file address when calling FOFI.

9.9 Processor Type

Each file's header has a TYPE word which is described in Sections 1.4 and 8.7. The file type of a processor, however, has special significance not discussed in that section.

Protection - All processors should be write protected to prevent inadvertent replacement or deletion. Read protect a processor only if it is for private use. Copy protection will prevent QUERYing the processor's attributes.

File type - The file type is used to identify a program file with its related processor. Therefore, processors with incompatible program files must have different file types. The active file will not be loaded by LUSR if its type does not match the processor. The file type should be zero only for a permanent system processor. Otherwise, the type should be one if the active file is not used or if it is used only for temporary storage and is not to be saved as a program file.

Control bits - Bits 6 through 8 of the TYPE word are control bits that are examined by SCOPE as described in Section 8.7 when a new processor is selected. Bit 8 must be set to indicate that the file is a runnable processor; files not structured as a processor must have zero in bit 8.

9.10 Debugging Procedures

If a new processor was created by use of ASSEMBLE it will be necessary to change it into a runnable processor as described in Section 8.7.

DSP (the Disc Service Processor) is a powerful tool for use in debugging a processor. The breakpoint and core snapshot are especially useful for this purpose. See Section 3.3.

The recommended procedure is to first set a breakpoint early in the swap-in subroutine, and then issue a system command to use the processor (either exit to the system with a CTRL C and issue the command, or use the C instruction in DSP to issue the command). Note that encountering the breakpoint causes the normal JSR to the processor's swap-out subroutine to be inhibited, and control is returned to DSP.

Set breakpoints successively further along in the swap-in routine, checking the contents of the registers at significant points in the code, until the swap-in procedure is fully debugged. Then use the same procedure in the body of the processor, starting at location BPS+4. Use conditional breakpoints for checking out loops and special conditions. Note that the breakpoint is cleared and the processor must start over from scratch each time a breakpoint is encountered.

At some point early in the check out it will be necessary to debug the swap-out subroutine. This must be done before a point is reached in the main code where a swap-out might occur. A forced swap-out for this purpose may be used by temporarily entering a CALL SCOPE in the main body of the code.

In some cases it is desirable to temporarily enter a TRAPFAULT instruction in the code in case the processor takes an unexpected branch. Such a case may occur following a call to any subroutine that has two or more possible returns.

After the processor is fully debugged, its protection may be changed to 33 or 22 to allow it to be used by other accounts.

10. DISC-RESIDENT SUBROUTINES

The disc-resident subroutines are assembled in three groups to produce a set of three object tapes which are loaded onto the disc as the file DISCSUBS. To be executed, the subroutine must be brought into a 256-word core block called the Subroutine Swap Area (SSA). Provision is also made for a larger (extended) subroutine to be brought into HXA and SSA as a 512 word block.

The CALL routine, which is core-resident, performs the task of bringing the proper block(s) of DISCSUBS into core and giving control to the desired subroutine. Two lines of assembly code are required to call a disc-resident subroutine:

```
        CALL              or        CHANNEL
        Subroutine                  Subroutine
```

where "Subroutine" is the name of a routine in the DISCSUBS file and has been equated to that subroutine's number by the Software Definitions tape. The word CALL or CHANNEL is actually a JSR via a page zero pointer to a core-resident calling routine. The word CHANNEL is used only when calling a channel-oriented routine, as CHANNEL replaces the Logical Unit number given in register AO with a pointer to the selected channel and checks the selected channel before giving control to the subroutine.

One DISCSUB may call another, and subroutines may be nested up to six levels deep in this manner. When one DISCSUB calls another, SSA is written on the disc to save any temporary storage cells in the first subroutine. In the case of an extended subroutine, the first block is brought into HXA and the second block is brought into SSA. Caution must be exercised when nesting extended subroutines since only SSA is saved on the disc when nesting occurs. For the same reason, an extended subroutine should not call, nor cause nesting to, any subroutine which uses HXA. However, if the first block of an extended subroutine will not be used later, then a call may be made from its second block to another extended subroutine or to a subroutine that uses HXA.

Disc-resident subroutines are relatively slow since a disc access is required to get the subroutine into core. A nested DISCSUB call requires three disc accesses to (1) write the calling subroutine on the disc, (2) read the called subroutine into core, and (3) read the calling subroutine back into core when the called subroutine is finished. It is possible to eliminate some or all of these disc accesses and thereby enhance the system throughput by specifying that certain DISCSUBS routines are to become core-resident as described in Section 2.9.

10.1 How to Write a DISCSUB

Several restrictions are imposed upon a disc-resident subroutine due to the conditions under which it must operate:

1) It must fit within a single disc block (256 words) or, if extended, it may occupy two disc blocks (up to 512 words).

2) It must be intrinsically relocatable; ie, all storage reference instructions must use either relative addressing or page zero system pointers, or pointers must be generated locally as in the example in Section 10.2.

3) It must be self-initializing; ie, any cell which is changed by the routine must not be assumed to initially contain the value which was assembled into the cell.

Actually, since linkage information is required at the beginning of each block (see Section 10.2), a maximum of 253 words may be used by a subroutine, or 509 words in an extended subroutine. Most system subroutines may be used (access and store byte routines, STOUTPUT, OUTTEXT, READBLOCK, WRITBLOCK, etc.), but routines such as BUMPUSER, WONA, and STINPUT, which will or might bump the user, may not be called because discsubs are not re-entrant.

Arguments may be passed both to and from the subroutine in registers A0, A1, A2, and the carry bit. A3 may be used to pass information from the subroutine back to the caller. Control is returned to the caller by a JMP 0,3 or a JMP @.NRET instruction for a non-skip return, or by a JMP 1,3 or a JMP @.SRET instruction for a skip return. Most DISCSUBS use a non-skip return under error conditions and a skip return when the task is successfully completed. Provision is also made for a status value to be put into register A3 on a non-skip return by the two-word instruction

```
JSR @.NRET
n*K!NOP
```

where n is the desired status value (not exceeding 177 octal) to be returned in A3. Obviously, register A3 cannot pass any information other than the status value back to the caller in this case, but the other registers and carry may still be used. NOP has been defined such that the expression n*K!NOP will also be a no-op if executed as an instruction; therefore, it is acceptable for a test instruction just ahead of the JSR @.NRET to skip over it and "execute" the n*K!NOP.

The only legal exit from a DISCSUB other than a return to the caller is a TRAPFAULT instruction upon discovery of a hardware or software fault. This will cause all nested subroutines as well as the core copy of the calling processor (if it is a swapping processor) to be aborted.

Copyright 1978
Educational Data Systems

10.2 How to Add a DISCSUB to the System

Each block of DISCSUBS must begin at a zero modulo 400 (octal) address. The first thing in each block is a linkage table for all routines in the block. There are two words in the linkage table for each routine, and these pairs of words must be in the same sequence as the routines themselves. The first of each pair of words is the name of the routine; this name, which must be defined in the Software Definitions, will also be used with a CALL or CHANNEL instruction to call the routine. The second word is the displacement from the beginning of the block to the entry point of the routine. The first word of the linkage table is labeled DSBn, where n is the block number in decimal. The second word of each pair in the linkage table may, therefore, be coded as LABEL-DSBn, where LABEL is the label on the routine's entry point. This label should be similar to the name of the routine, but it should end with an X. For example, the entry point of the TRAPFAULT subroutine is labeled FALTX. The entry point must be the very first word of the subroutine.

The new subroutine must be assigned a number, and its name is equated to this number in the Software Definitions tape. The number must be less than the definition for NSUB in the CONFIG file. If necessary, increase NSUB to be greater than the last DISCSUB number (see Section 2.4). Discsub numbers 120 through 127 octal have been set aside for customers to assign to subroutines for their own use only. If a subroutine is to be used on other systems to be supported by Educational Data Systems, please call for a number assignment.

Higher order bits of the subroutine's assigned number are used as flags indicating various attributes of the routine as follows:

bit 15	subroutine is in REX (not in DISCSUBS)
bit 14	subroutine is extended (occupies two blocks)
bit 13	included with preceeding routine if core-resident
bit 11	alternate version for core-residency only
bit 10	(not used)
bit 9	(not used)
bits 8-0	subroutine identification number (777 maximum)

Note: bit 15 is the most significant bit of the word.

If a subroutine is extended, it must be the last one in the block in which it begins, and the extension must be in the next block of DISCSUBS. There is no linkage table in the extension block, thus allowing an extended subroutine to be up to 509 words long. There may be no other subroutines in the extension block, but an extended subroutine may be preceeded by small subroutines in the first block.

The linkage table must be terminated by a negated displacement to the last word occupied or used by the last routine in the block. This is used by SIR to determine the size of the last subroutine if making it core-resident.

It is also possible to replace or add a single block of DISCSUBS without replacing the entire file. Make up a source tape of the new block or blocks, and assemble it without the rest of the DISCSUBS source tapes. Put the object file on the disc temporarily (either ASSEMBLE it on the disc or use PLOAD or COPY to load it under a different name such as DSUB). Use the G and W commands in DSP to copy this new version into the DISCSUBS file, then do an IPL. Any block of DISCSUBS containing 177400 in word zero is a spare block into which the new subroutine may be loaded. If new blocks are being added, use DSP's "A" command to first append the required blocks to DISCSUBS, but be sure that an extended subroutine occupies two logically sequential disc addresses. See Section 4.3 also. When finished copying blocks, kill the temporary file, or leave it on the disc for backup. If increasing NSUB without a complete system generation, then NSUB in the CONFIG file must be increased.

WARNING! Do an IPL immediately after adding or replacing any DISCSUBS block to be sure that the new subroutines are properly linked to the system.

A completed DISCSUBS block would look something like this:

```
        .LOC     12400   ;"DISCSUBS" BLOCK #21

DSB21:  SINH                  \
        SINHX-DSB21            \
        COSH                    >   Linkage table
        COSHX-DSB21            /
        DSB21-DS21E           /

SINHX:  JSR   SINHI           \
        102663                 \
        015252                  )   This technique is
        135661                  )   recommended to get
        002447                  )   a table pointer,
         .                      )   yet maintain
         .                      )   relocatability.
         .                     /
SINHI:  STA   3,SADDR         /
         .
         .
         .

SADDR:  0

COSHX:  JSR   COSHI  ;HYPERBOLIC COSINE FUNCTION
         .
         .
         .

DS21E   =.     ;END OF "DISCSUBS" BLOCK #21

        .LOC    DSB21+400-.  ;BLOCK OVERFLOW TEST
```

Copyright 1978
Educational Data Systems

10.3 How to Debug a DISCSUB

DSP may be used to examine and modify subroutines in the DISCSUBS file the same as for a processor. Breakpoints may be set in the calling processor just ahead of or just after the suboutine call, but breakpoints cannot be set in the DISCSUB itself. Two alternatives are possible, however; if there are no other users on the system, then halts may be inserted in the routine, or the routine may be made core-resident and DBUG used to debug it. If the system is in use, insert a TRAPFAULT instruction in the routine where a breakpoint would be desired; although not as convenient, this will give the effect of a breakpoint except that the TRAPFAULT will destroy the contents of register A3, and it will affect anyone who uses the subroutine, whereas a DSP breakpoint affects only the user who set it. Other users should not be allowed to call a new routine, however, before it has been fully checked out in all possible modes, and any instructions that were modified for debugging have been restored.

The easiest way to debug a DISCSUB is to make it core-resident (see Section 2.9) and use DBUG while there are no other users on the system. Do an IPL to be sure of a fresh copy of DBUG in core, then enter a HALT at the DISCSUB's entry point in core by giving DSP the following instructions:

```
F DISCSUBS
entry':63077
```

where "entry" is the octal address of the subroutine's entry point as shown on the listing, and the apostrophy indicates that the entry is to be made in the core-resident copy of the subroutine, wherever it may be. Now run a small BASIC program that calls the DISCSUB. The computer will halt when the DISCSUB is called. Write down the address one less than that shown on the front panel lights; this is the actual location in core of the DISCSUB's entry point. Now RESET and START at location 25000 to enter DBUG, and type the instruction F[lights-1],[entry] and press the RETURN key. For example, if the entry address on the listing was 26403 and the computer halted with 75306 in the address lights, then the command to DBUG would be F75305,26403. DBUG will print the difference (46702) to indicate that this offset will be used henceforth for all addresses. Now restore the first instruction of the subroutine by typing [entry]:[instruction], and type an A command to get a print out of the contents of the registers at the time the subroutine was entered. Serious debugging may now begin by setting breakpoints further along in the subroutine and using the J command to re-enter the subroutine after each breakpoint has been encountered. The entry address must be given with the first J command because there was no previous breakpoint, but a simple J command without an address may be used thereafter if it is desired to continue through the code to the next breakpoint. An F command with no argument may be used to toggle between real core addresses and virtual (subroutine listing) addresses. See Section 3.2 for detailed information on using DBUG.

10.4 How to Write a DISCSUB for Business BASIC

Machine code subroutines written to be used by the CALL statement in Business BASIC must accept and return information in a specific format. Three parameters are passed to the subroutine in the registers as follows:

```
        (A0) = Pointer to first available core location
        (A1) = Pointer to last available core location
        (A2) = Pointer to argument pointer list
```

Registers A0 and A1 contain the first and last addresses of the currently unused cells in the BASIC user's storage area. This space is available for use as temporary storage by the subroutine.

At the time control is transferred to the subroutine, BASIC has analyzed the arguments supplied in the CALL statement and has placed pointers to these BASIC variables in the argument pointer list. Register A2 contains the address of the first cell of this list which can hold a maximum of twelve argument pointers. Each argument may be either a decimal number or a string. In the case of a decimal number, the pointer will point to the first of the words where that number is stored, and the next word after the pointer will contain the number type (1, 2, 3, or 4 words). The number may be one element of an array, but no provision is made for using an entire array. In the case of a string, the pointer will point to the word containing the first two bytes of the string, and the next word after the pointer will be the string dimension with a one in bit 15 (the most significant bit).

CAUTION! These are absolute, not relative, word addresses. However, in a system with more than 32K words of core, a word address requires the full 16-bit word, so it can not be simply shifted left for use as a byte address. Instead, convert to a relative byte address by subtracting the partition address and shifting left; eg, the sequence

```
        LDA    0,BBA
        SUBOL  0,1
```

will convert an absolute word address in A1 to a relative byte address.

For example, suppose a Business BASIC program contains the statement:

```
        120 CALL 5,R,B$,N[1,3]
```

The arguments will be passed to subroutine number 5 as follows:

```
          Register A2                  |--------------------|
      -------------------         X2:  |  Value of          |
      |    address X1   |              |    variable R      |
      -------------------              |--------------------|


     Argument pointer list                  Variable B$
      -------------------              |--------------------|
X1:   |    address X2   |         X3:  |  byte 1  |  byte 2 |
      |-----------------|              |--------------------|
      | 0 | number type |              |  byte 3  |  byte 4 |
      |-----------------|              |--------------------|
      |    address X3   |              |  etc.    |         |
      |-----------------|
      | 1 |  dimension  |
      |-----------------|
      |    address X4   |              |      Array N       |
      |-----------------|              |                    |
      | 0 | number type |              |--------------------|
      |-----------------|         X4:  |  Value of          |
      |        .        |              |    N[1,3]          |
      |        .        |              |--------------------|
      |        .        |              |                    |
```

The remainder of the argument pointer list will be filled with pointers to a dummy variable which is ignored be BASIC when the subroutine returns.

The subroutine must do a skip return if its operation is successful. A non-skip return will cause Business BASIC to print an error message.

To make the subroutine available to be CALLed by a BASIC procram, it must be included in the DISCSUBS file. Refer to Sections 10.1 and 10.2 for special considerations in writing a disc-resident subroutine and how to include a new routine in the DISCSUBS file.

Once the new routine has been included as a DISCSUB, an entry must be inserted in the call table (CALLT) in the RUN processor. There is a pointer to CALLT in location 200 (octal) in RUN. Use DSP to look through CALLT for the first minus one (177777 octal), and replace it with a word containing the desired BASIC subroutine number (not exceeding 177 octal or 127 decimal) in the lower (right hand) byte and the actual DISCSUB number in the higher (left hand) byte. Also, an extended discsub must be flagged by a one in bit 7 of the word (add octal 200). Be sure the next cell in CALLT is 177777 to terminate the table. CALLT may be extended through location 577 octal in RUN.

11. ADDING DEVICES TO THE SYSTEM

An input/output or mass storage device may be added to IRIS in any of three different ways depending on the characteristics of the device and its intended use. In general, a device may be:

1) An interactive port through which a user communicates with processors and runs application programs, or

2) A peripheral device which may be OPENed by any caller granted access; the caller then does input or output by doing a READ or WRITE to the device as if it were a data file, or

3) A Logical Unit having its own INDEX, thus allowing any user granted access to READ or WRITE to files on the device, and allowing any user given an account in the Logical Unit's ACCOUNTS file to BUILD files on the device.

User oriented devices such as typewriters, teleprinters, and CRT terminals are desirable as interactive ports, although a combination such as a line printer and card reader could also be used in this manner. Devices such as line printers, card readers, paper tape units, cassette tape drives, graph plotters, data acquisition devices, and communications channels are usually interfaced as peripheral devices. Disc and drum memories are usually interfaced as Logical Units, but a cartridge disc (especially a floppy disc) could be interfaced as a peripheral device if the cartridge is to be used to transfer data between IRIS and another computer system. A high performance magnetic tape drive is usually interfaced as a peripheral device, but if such a unit has the ability to rewrite a record without destroying the following record it could be set up as a Logical Unit. A multiplexer driver is usually written such that some of its channels are interactive ports and others are used to interface peripheral devices as determined by its attributes table. Each interactive port has its own active file on the system disc for saving the current state of a job between time slices, but non-interactive ports do not have active files.

11.1 Interactive and Peripheral Device Drivers

Each device driver is written as an independent module and loaded onto Logical Unit zero as a separate file by use of ASSEMBLE, COPY, or PLOAD. The Filename must begin with a dollar sign and should indicate the device type; eg, $LPT for a line printer, $CRD for a card reader, etc. Any dollar sign file must start at location BPS with pointers to its interrupt handler and attributes table, plus three other pointers dependent on its type. The five pointers must be followed immediately by the entry to the driver's initializing routine. Also, each file ends with an attributes table, a linkage pointer table, and a port definition table.

If the driver does not have an interrupt handler, then the INTH pointer must be -1. However, if it does have an interrupt handler, then its entry point must be immediately preceeded by a jump to the power fail restart subroutine or by a JMP 0,3 instruction if no special action is required for a power fail restart. This jump instruction must be immediately preceeded by a mask word containing a "one" bit to inhibit further interrupts from this device during execution of the handler since interrupts are enabled during this time. The mask word may also contain additional "one" bits as desired to also inhibit interrupts from lower priority devices. The interrupt handler returns control to the system as soon as possible with a JMP 0,3 instruction, or with a JMP @.INTR instruction if register A3 has been changed.

The purpose of the power fail restart subroutine is to re-initiate operation of the device following a power failure, and the restart should be done with minimal loss of data (with no data loss if possible).

When scanning the INDEX during an IPL, SIR sees the dollar sign Filename, brings the driver into core, and links it into the system as indicated by these pointers and tables. There are two categories of files given Filenames starting with a dollar sign. They are:

1) Peripheral drivers (file type 36 octal plus whatever protection is desired against use of the device). The driver has FINIS, WRITE, and READ subroutine pointers following the attributes table pointer (at BPS+1) or a -1 in place of the pointer if a subroutine is not included. This category includes only drivers for devices that are to be OPENed on a data channel and used for data input and/or output, such as a line printer, card reader, paper tape equipment, etc.

2) System subroutines and drivers (file type 77001). The WRITE subroutine and pointer are replaced by a SEND CHARACTER subroutine and pointer; SIR places an absolute pointer to this subroutine in the SND cell of each PCB. The READ pointer is not used, and the FINIS pointer is replaced by a pointer to the first word of the driver which is to be core-resident. This category includes:

   a) Interactive device drivers (eg, $TTY),
   b) Multiplexer drivers (eg, $MMUX, $BBMX),
   c) System subroutines (eg, $DEC, $TERMS), and
   d) System device drivers (eg, $DAU, $RTC).

CAUTION! The driver must be intrinsically relocatable (position independent) since SIR may put it anywhere in core. There must be no absolute pointers or references to absolute locations in the driver other than the five entry pointers and the linkage pointer table.

Copyright 1978
Educational Data Systems

The attributes table, which is at the end of the driver, consists of three words as follows:

0) This cell usually contains a zero. When brought into core, SIR puts a pointer to the first PCB for this driver (if any are assigned) into this cell. However, if the hardware requires a specific first PCB address, then that location rather than a zero should be put at ATRIB, and SIR will attempt to allocate core to accomodate this requirement.

1) This word would have a single "one" bit if desired to enable interrupts from the device, and that bit of the system's initial mask word will be zeroed by SIR. This word may be zero if no interrupts are to be enabled, but it may not have more than a single one bit.

2) If the driver has an interrupt handler then this word must contain the device address with which the device responds to an INTA instruction. SIR will generate an interrupt vector to the driver's INTH routine. A zero in this cell means no interrupt vector is to be generated.

The attributes table is followed by a linkage pointer table and a port definition table. The linkage pointer table, which starts at ATRIB+3, consists of two words per entry as follows:

0) Absolute core location for pointer (usually in page zero)
1) Assembled (virtual) location to point to in the file

The pointer table is terminated by a -1, which may be at ATRIB+3 if no pointers are to be generated, and this -1 is immediately followed by the port definition table which consists of eight words per entry as follows:

0) Number of ports with the following characteristics
1) PCW - Port Control Word (see next page)
2) Buffer size (number of bytes for biggest I/O)
3) RDE - Return delay and EOM code or terminal type code
4) 0 (this cell reserved for future use)
5) 0 (this cell reserved for future use)
6) Size of active file (number of blocks)
7) 0 (this cell reserved for future use)

When one word holds two parameters, the first parameter is in the top byte. The list of ports must also be terminated by a -1; the table may be empty, but the -1 terminator is required. A PCB is assigned for each port in this list; alternatively, the driver may supply its own I/O buffer rather than supplying a list of ports here. No active files or data file tables should be assigned for peripheral devices (ie, size of active file should be zero).

CAUTION! The attributes and these two tables must be entirely within the last block of the file. See the symbols BLK1 and BLK2 in Figure 11.1 for a test to ensure that this is the case.

The Port Control Word (PCW) in the Port Definition Table (PDT) and in the Port Control block (PCB) controls various characteristics of the port such as baud rate, modem control, parity checking, etc., provided that the hardware allows these parameters to be controlled by software. PCW should be zero for any device which cannot control any of these characteristics. The format of the Port Control Word is as follows:

bit*	meaning
15_	0
14	Port is on an EDSI multiplexer
13	0
12_	Initial device control output (1 = high, 0 = low)
11	Normal device ready status (1 = high, 0 = low)
10	Port is a phantom port
9_	Auto log-off is enabled
8	Auto frequency scan is enabled
7	Inhibit parity check & generation
6_	0 ==> one stop bit, 1 ==> two stop bits
5	\ Character length:
4	/ 3 = 8 bits, 2 = 7 bits, 1 = 6 bits, 0 = 5 bits
3_	parity (if not inhibited): 0=odd, 1=even
2	\ Baud rate: 0 = 110, 1 = 150, 2 = 300,
1	} 3 = 600, 4 = 1200, 5 = 2400,
0	/ 6 = 4800, 7 = 9600.

* Bit 15 is the most significant bit.

For easy reference, some of the most commonly used Port Control Words are listed below:

For CRT terminals:

40077 = 8-bit character with even parity at 9600 baud
40067 = 8-bit character with odd parity at 9600 baud
40057 = 7-bit character with even parity at 9600 baud
40047 = 7-bit character with odd parity at 9600 baud

For Teletype:

40360 = 8-bit char., no parity, 2 stop bits, 110 baud
40150 = 7-bit char., even parity, 2 stop bits, 110 baud

For Modems:

55054 = 7-bit character, even parity, 1 stop bit, 300 baud with modem control (Data Terminal Ready = high, Auto Log-Off and Auto Frequency Scan enabled)

55452 = 7-bit character, even parity, 1 stop bit, 1200 baud, Data Terminal Ready, Auto Log-Off, but no Auto Frequency Scan

For Line Printers:

44277 = 8-bit character, no parity, 9600 baud, printer "ready" status (on pin 20 on mux connector) high

40277 = same as 44277 but "ready" status low

11.2 How to Write a Peripheral Device Driver

Figure 11.1 shows the general form of a peripheral driver file. Everything from the pointer to ATRIB (location BPS+1) through the cell labeled ATRIB, inclusive, is brought into core by SIR, and the four pointers (ATRIB, CLOSE, WRITE, and READ) are modified to point to the actual resulting core locations. The entry to the initializing subroutine is immediately following the pointer to the READ subroutine, and the location of the INIT entry is written into the STAD cell of the file's header for use by OPEN and to allow the programmer to locate the driver in core for debugging.

The system will do a JSR to the INIT (initializing) subroutine with the channel address in A2 when a caller OPENs the device, and it will JSR to the CLOSE subroutine (if supplied) when the channel is CLOSEd or CLEARed. If either of these routines is very long then it may be written as a DISCSUB which is called by a short core-resident routine in order to conserve core space. The INIT routine is mandatory but may be only a JMP 1,3 instruction if initialization is not required. The INIT routine may do a non-skip return with a status value in register A3 (see Appendix 2, page 4) if the device cannot be opened because it is off line, etc.; a skip return indicates that the device has been successfully opened.

The READ and WRITE routines must look the same to the system as the READ ITEM and WRITE ITEM system subroutines (see Catagory #3 in Appendix 1); these routines must skip return if the operation is successful, or non-skip return with status in register A3 (see Appendix 2, page 4) if any error is detected. If a device does not have input capabilities then there must be a -1 in place of the READ entry pointer, if there are no output capabilities then there must be a -1 in place of the WRITE entry pointer, and if there is no wrap-up routine then there must be a -1 in place of the CLOSE pointer.

Interrupts are enabled when any driver subroutine (READ, WRITE, INIT, or CLOSE) is invoked, and should remain enabled when the driver returns control to the system. The driver may disable interrupts for brief periods if necessary. The driver's interrupt handler may not call any non-reentrant subroutine nor may it do a TRAPFAULT.

If a port control block is not needed then the -1 port definition table terminator must immediately follow the linkage pointer table terminator. If a port is defined then word 3 of the attributes table is put into the RDE cell in the port's PCB, and the two special character delay words are put into the PCB's SD1 and SD2 cells. It is up to the driver itself to implement these delays, or these words may be used for another purpose by the driver if the special character delays are not used. The top byte of word 6 is put into the PCB's TLL cell, and the lower byte of word 6, which is the "size of active file" value, must be zero.

The file type for a peripheral device driver is 00036. See Section 11.1 for more details.

Figure 11.1: Typical Peripheral Driver File

```
        .TXTM   1           ;FOR CORRECT TEXT PACKING
        .LOC    BPS         ;ALL DRIVERS MUST START AT BPS

        INTH    ;POINTER TO INTERRUPT HANDLER
        ATRIB   ;POINTER TO ATTRIBUTES TABLE
        CLOZ    ;POINTER TO CLOSE SUBROUTINE (OR -1)
        WRITE   ;POINTER TO OUTPUT SUBROUTINE (OR -1)
        READ    ;POINTER TO INPUT SUBROUTINE (OR -1)

        STA     3,TEMP      ;INITIALIZE (OPEN) SUBROUTINE
        ---
        JMP     @TEMP       ;ERROR RETURN (STATUS IN A3)
        ISZ     TEMP
        JMP     @TEMP       ;DEVICE SUCCESSFULLY OPENED

        403                 ;MASK BIT(S) TO DISABLE
        JMP     PFRST       ;POWER FAIL RESTART ENTRY
INTH:   ---     ---         ;INTERRUPT HANDLER
        JMP     0,3         ;(JMP @.INTR IF A3 CHANGED)

PFRST:  ---     ---         ;POWER FAIL RESTART SUBROUTINE
        JMP     0,3

CLOZ:   ---     ---         ;WRAP-UP (CLOSE) SUBROUTINE (OPTIONAL)
        JMP     0,3         ;(SKIP RETURN OK, NO DIFFERENCE)

WRITE:  ---     ---         ;DATA OUTPUT SUBROUTINE (OPTIONAL)
        JMP     1,3         ;(NON-SKIP WITH STATUS IN A3 IF ERROR)

READ:   ---     ---         ;DATA INPUT SUBROUTINE (OPTIONAL)
        JMP     1,3         ;(NON-SKIP WITH STATUS IN A3 IF ERROR)

BLK1    =.-BPS/400          ;FOR TEST BELOW

ATRIB:  0       ;PCB LOCATION (SET BY SIR)
        400     ;MASK BIT TO BE ENABLED
        XXX     ;DEVICE ADDRESS (RESPONSE TO INTA)
                ;LINKAGE POINTER TABLE HERE IF REQUIRED
        -1      ;LINKAGE POINTER TABLE TERMINATOR
        1       ;NUMBER OF PORT CONTROL BLOCKS TO BE ASSIGNED . .
        40057   ; WITH THIS PORT CONTROL WORD . . .
        200     ; THIS I/O BUFFER SIZE (NUMBER OF BYTES) . . .
      3*K+215   ; AND THIS RETURN DELAY AND EOM CODE.
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        0       ; NO ACTIVE FILE (NOT AN INTERACTIVE PORT)
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        -1      ;PORT DEFINITION TABLE TERMINATOR

BLK2    =.-BPS/400
        .LOC    BLK1-BLK2 ;TEST THAT ATRIB IS ALL IN LAST BLOCK

        .END    ;END OF DRIVER
```

11.3 How to Write an Interactive or System Device Driver

A driver for a system device or for an interactive device has the same form as one for a peripheral device (see Sections 11.1 and 11.2) with the following exceptions:

a) An interactive port is never OPENed; the initializing routine is not core-resident but is brought into core separately by the system's startup or recover routine,

b) The wrap-up routine is not used; the FINIS entry pointer is replaced by a pointer to the first word which is to be core-resident,

c) The WRITE routine is not used; the WRITE entry pointer is replaced by a pointer to a SEND CHARACTER subroutine,

d) The READ routine is not used; the READ entry pointer is replaced by a minus one,

e) Each port is assigned an active file and data file table (ie, it is an interactive port) if and only if the lower byte of word 6 is non-zero, and

f) The file type must be 77001.

Note that, in the case of a multiplexer, each port may have a different speed, I/O buffer size, line length, active file size, EOM character or terminal type, and return delay. The terminal type, return delay, and speed may be changed by the user after logging on to the port. In most cases, the active file size should be a number of complete tracks and at least as big as calculated at step 8 in Section 2.8, but AFSET will attempt to allocate more blocks if an active file is too small for a user's program. See Section 2.8 for more comments on active file size.

The SEND pointer is converted to an absolute pointer by SIR and stored in the SND cell of each PCB for later user by the system. The SEND routine must accept a character in register A0 and output it to the port whose PCB pointer is given in register A2, or put the character in the PCB's TOB cell if the device is busy (-1 in A0 means "start output", -2 in A0 means "start input", and -3 in A0 means "stop any input or output"). The SEND subroutine must not change register A2 in any of its modes of operation. The "start output" subroutine must return with -1 in A0 if the system is to process characters, or any other value in A0 if the device accesses characters directly from core. The "start input" subroutine normally returns with a one in the most significant bit of A0, but it must zero this bit if DMA input is being started and a character is waiting to be processed in single character mode.

CAUTION! For this type of driver, the entire initializing routine and the first core-resident cell must be within the first block of the file, and the INIT routine must always return non-skip.

Figure 11.2: System Device Driver File

```
        .TXTM   1           ;FOR CORRECT TEXT PACKING
        .LOC    BPS         ;ALL DRIVERS MUST START AT BPS

        INTH    ;POINTER TO INTERRUPT HANDLER (OR -1)
        ATRIB   ;POINTER TO ATTRIBUTES TABLE
        INTH-2  ;POINTER TO FIRST CORE-RESIDENT CELL
        SEND    ;POINTER TO "SEND CHARACTER" SUBROUTINE
        -1      ;(NOT USED)

        ---     ---         ;INITIALIZING ROUTINE
        ---
        JMP     0,3

        773                 ;MASK BIT(S) TO DISABLE
        JMP     PFRST       ;POWER FAIL RESTART ENTRY
INTH:   ---     ---         ;INTERRUPT HANDLER
        JMP     0,3         ;(JMP @.INTR IF A3 CHANGED)

PFRST:  ---     ---         ;POWER FAIL RESTART SUBROUTINE
        JMP     0,3

SEND:   ---     ---         ;"SEND CHARACTER" SUBROUTINE
        JMP     0,3

BLK1    =.-BPS/400 ;SEE TEST BELOW

ATRIB:  0       ;FIRST PORT'S PCB LOCATION (SET BY SIR)
        400     ;MASK BIT TO BE ENABLED
        XXX     ;DEVICE ADDRESS (RESPONSE TO INTA)
                ;LINKAGE POINTER TABLE HERE IF REQUIRED
        -1      ;LINKAGE POINTER TABLE TERMINATOR
        7       ;NUMBER OF PORT CONTROL BLOCKS TO BE ASSIGNED . .
        40057   ; WITH THIS PCW . . .
        79      ; THIS I/O BUFFER SIZE (NUMBER OF BYTES) . . .
      0*K+215   ; AND THIS RETURN DELAY AND EOM CODE.
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        30      ; ACTIVE FILE SIZE (24 BLOCKS)
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        1       ;NUMBER OF PORT CONTROL BLOCKS TO BE ASSIGNED . .
        40054   ; WITH THIS PCW . . .
        1000    ; THIS I/O BUFFER SIZE (NUMBER OF BYTES) . . .
      4*K+215   ; AND THIS RETURN DELAY AND EOM CODE.
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        0       ; NO ACTIVE FILE (NOT AN INTERACTIVE PORT)
        0       ; (THIS CELL RESERVED FOR FUTURE USE)
        -1      ;PORT DEFINITION TABLE TERMINATOR

BLK2    =.-BPS/400
        .LOC    BLK1-BLK2 ;TEST THAT ATRIB IS ALL IN LAST BLOCK

        .END    ;END OF DRIVER
```

11.4 How to Write a System Subroutine Module

Large system subroutines such as $DEC (the decimal arithmetic routines) may be written as a separate module and loaded as a dollar sign file (type 77001). The three words of the attributes table must be zero, and all linkage with the system must be set up by the linkage pointer table; otherwise, it is similar to a system device driver as described in Section 11.3. Figure 11.3 shows the form of a system subroutine file.

Figure 11.3: System Subroutine File

```
.TXTM   1       ;FOR CORRECT TEXT PACKING
.LOC    BPS     ;ALL $ FILES MUST START AT BPS

        -1      ;NO INTERRUPT HANDLER
        ATRIB   ;POINTER TO ATTRIBUTES TABLE
        DEC     ;POINTER TO FIRST CORE-RESIDENT CELL
        -1      ;NO "SEND CHARACTER" SUBROUTINE
        -1      ;(NOT USED)

        ---     ---         ;INITIALIZING SUBROUTINE
        ---
        ---
        JMP     0,3

DEC:    ---     ---         ;SUBROUTINE ENTRY
        ---
        ---
        JMP     0,3

BLK1    =.-BPS/400          ;FOR TEST BELOW

ATRIB:  0       ;NO PCB
        0       ;NO MASK BIT
        0       ;NO DEVICE ADDRESS
        .DEC    ;THIS PAGE ZERO POINTER...
        DEC     ; IS TO POINT TO THIS LOCATION
        .XXX    ;REPEAT POINTER PAIRS AS REQUIRED
        XXX
        -1      ;LINKAGE POINTER TABLE TERMINATOR
        -1      ;PORT DEFINITION TABLE TERMINATOR (NO PORTS)

BLK2    =.-BPS/400
        .LOC    BLK1-BLK2 ;TEST THAT ATRIB IS ALL IN LAST BLOCK

        .END    ;END OF FILE
```

11.5 How To Write a Multiplexer Driver

A multiplexer driver is a system device driver as described in Section 11.3, except that special consideration is necessary to allow some ports to be used for interactive terminals while others are used to interface peripheral devices. Which ports are to be interactive and which are for peripheral devices is indicated by a zero or non-zero value, respectively, in the lower byte of word 6 of each entry in the port definitions table at the end of the driver. Any non-zero value causes SIR to allocate an active file with that number of blocks and also to allocate a data file table in core for each port in the group. A zero value in the lower byte of word 6 means that no active file or data file table will be allocated; hence the port cannot be used interactively, but it can be used to interface a peripheral device if the multiplexer driver provides facilities for peripheral drivers as described below, and a suitable peripheral driver is provided (see Section 11.7).

To allow peripheral devices to operate through the multiplexer, the mux driver must include the following code in its output interrupt handler routine:

```
        LDA     3,AHA.,2
        SKZ     3,3       ;INTERACTIVE PORT ?
        JMP     .+7       ;  YES
        LDA     3,TON.,2  ;  NO
        SSP     3,3       ;CHAR. HANDLING REQUESTED ?
        JMP     .+4       ;  YES
        SKZ     3,3       ;PERIPHERAL SERVICE PROVIDED ?
        JSR     1,3       ;  YES
        JMP               ;SERVICE NEXT PORT
         .
         .      (service this port as an
         .         interactive terminal)
```

where register A2 contains the PCB pointer for the port being serviced, and A0 contains the character to be processed. Also, a similar code sequence must be included in the mux driver's input interrupt handler routine, except that the JSR 1,3 instruction must be replaced by a JSR 0,3 instruction, and the character (if any) will be returned in register A0. The peripheral device driver is not allowed to change register A2.

The mux driver's power fail restart subroutine must also include a similar code sequence in a loop that examines each port, except that the JSR 1,3 instruction is replaced by a JSR -1,3 instruction. This is necessary to allow each peripheral driver to restart its respective device. The multiplexer driver should then restart the port hardware if and only if the peripheral driver's restart subroutine does a skip return.

Copyright 1978
Educational Data Systems

The mux driver must accept commands from peripheral drivers via its SEND subroutine. The peripheral driver may do a JSR @SND.,2 with a port's PCB pointer in register A2 and a command in register A0 as follows:

(A0) ≥ 0 Supplies an output character (not exceeding 377) to be output in single character mode. May be used to echo each character in single character input mode and to echo control characters which cause termination of buffered input. The character is saved in TOB and automatically sent later if the mux is busy. Does not cause an "output done" condition.

(A0) = -1 Starts output (in auto buffer mode if mux is capable). The peripheral driver must set the "output active" bit in FLW before starting output. Mux driver will set LOB=OBP then set OBP=FBA before starting output unless (A1)=0. Output is in ASCII mode with even parity unless the "binary output mode" bit in FLW is set.

(A0) = -2 Starts input (in auto buffer mode if mux is capable). The peripheral driver must set IBP=FBA and LIB=LBA and set the "input active" bit in FLW before starting input. Input is in ASCII mode, and all input characters must have even parity unless the "suppress parity check" bit in FLW is set. Normally returns with a one in bit 15 of A0 (eg, A0 unchanged), but must zero bit 15 of A0 and not start auto buffer input if a single character mode input is pending.

(A0) = -3 Stops any input or output in progress.

(A0) = -4 Sends port's PCW (Port Control Word) to the mux port. Usually preceded by a change to PCW (eg, toggle device control output bit). Output must not be active when this command is given.

The mux driver must non-skip return to the peripheral driver with register A2 unchanged.

The peripheral driver must clear the "input active" and "output active" bits in FLW when an input or output is terminated for any reason and is not being immediately restarted.

The mux driver passes information about special conditions to the peripheral driver by doing a JSR via the pointer in TON. In each case, the registers will contain:

```
                (A0) = character or status
                (A1) = condition
                (A2) = port's PCB pointer
                (A3) = return address
```

Single character input will occur even if buffered input is not enabled. Characters entered may be dismissed, if desired, by simply returning without processing. If buffered input is not enabled, the condition indicators possible in register A1 are:

```
        (A1)=0 ==> Character entered
                (A0) = 0 if terminator character entered
                (A0) = ASCII character code if not terminator

        (A1)=1 ==> Special condition
                (A0) = 1 if parity error detected
                (A0) = 202 if break detected
                (A0) = 020 if CTRL P entered
                (A0) = 233 if ESC entered

        (A1)>1 (see below)
```

If buffered input is enabled, then the condition indicators possible in register A1 are:

```
        (A1)=1 ==> Special condition
                (A0) = 0 if terminator character entered
                (A0) = 1 if parity error detected
                (A0) = 2 if input buffer full
                (A0) = 202 if break detected
                (A0) = 020 if CTRL P entered
                (A0) = 233 if ESC entered
                (A0) = code < 240 if any other control character

        (A1)>1 (see below)
```

The following status conditions are possible in register A1 whether buffered input is enabled or not:

```
        (A1)=2 ==> Status line (pin 20) change
                (A0) = status (0 ==> negative, ≠0 ==> positive)

        (A1)=3 ==> Buffered output done
```

The peripheral driver's Process Special Condition subroutine (at the address pointed to by TON) must return non-skip in any case with register A2 unchanged.

11.6 Character and Interrupt Task Queuing

Input and output characters for interactive ports, as well as other interrupt tasks, may be queued by a driver's interrupt handler by executing a QCHARACTER instruction with the character or a task identifier in register A0 and a PCB pointer in A2. These two words are put on the character queue for later processing as determined by the value in A0 as follows:

(A0)	Taken as		
< 0	Output character request	)	(A2) = PCB pointer.
0-377	Input character	/	
400	Start input	\	
401	Terminate output	)	(A2) may be any
402-407	(not assigned)	)	value to be passed
> 407	Task starting location	/	to the task.

If (A0)<0 then the character in the PCB's TOB cell is returned in A0, and TOB is zeroed. The top bit of (A0) will be a "one" to indicate the presence of a character in the lower byte; zero in A0 indicates end of output. If TOB was non-zero, then a request for another output character is put on the queue. Any value greater than 407 octal in A0 must be the starting location of a task, and control will be transferred later to that location by a JSR instruction with interrupts enabled and the same value in A2 as when QCHARACTER was executed.

11.7 How to Drive a Peripheral on a Multiplexer

A peripheral driver for a device which is interfaced through a multiplexer is the same as any other peripheral driver (see Section 11.2) except that it has no interrupt handler or port of its own. The multiplexer driver supplies the port and handles all interrupts; therefore, the peripheral driver's INTH pointer must be -1, the attributes table must contain all zeroes (3 words), and the port definition table must be empty.

The multiplexer's interrupt handler passes control to the peripheral driver for character processing by means of a pointer in the PCB's TON cell as described in Section 11.5. The peripheral driver's INIT routine must check that the PCB's AHA cell is zero, generate an absolute pointer to its input character processing subroutine, and store that pointer in the TON cell. The output character processing subroutine must immediately follow the entry to the input processing routine entry, and the power fail restart entry must immediately preceed the entry to the input processing routine entry. The character processing subroutines must not change register A2 which contains the PCB pointer. Each subroutine must return non-skip as soon as possible.

The peripheral driver may give commands to the mux driver via its SEND subroutine as described in Section 11.5. Figure 11.4 shows the form of a driver for a peripheral device which is connected to the system through a multiplexer. No mask word is required.

Figure 11.4: Driver for Peripheral on Multiplexer

```
.TXTM   1       ;FOR CORRECT TEXT PACKING
.LOC    BPS     ;ALL DRIVERS MUST START AT BPS

        -1      ;NO DIRECT INTERRUPT HANDLER
        ATRIB   ;POINTER TO ATTRIBUTES TABLE
        FINIS   ;POINTER TO WRAP-UP SUBROUTINE (OR -1)
        WRITE   ;POINTER TO OUTPUT SUBROUTINE (OR -1)
        READ    ;POINTER TO INPUT SUBROUTINE (OR -1)

        STA     3,TEMP   ;INITIALIZE (OPEN) SUBROUTINE
        JSR     SETUP    ;GENERATE POINTER TO NEXT CELL

        JMP     PFRST    ;POWER FAIL RESTART ENTRY
        JMP     INPCH    ;ADDRESS OF THIS CELL PUT IN TON
        ---     ---      ;OUTPUT CHARACTER HANDLER
        JMP     0,3

INPCH:  ---     ---      ;INPUT CHARACTER HANDLER
        JMP     0,3

PFRST:  ---     ---      ;POWER FAIL RESTART (MUST PRESERVE A2)
        JMP     1,3      ;(NON-SKIP IF MUX DRIVER NOT TO RESTART)

SETUP:  INC     3,3      ;SET CHARACTER HANDLER ADDRESS
        STA     3,ATRIB  ;SAVE ADDRESS OF JMP INPCH INSTRUCTION
        ---     ---      ;USE CPNPP TO COMPUTE PCB ADDRESS
        LDA     3,ATRIB  ;PUT JMP INPCH ADDRESS IN TON CELL
        STA     3,TON.,2 ;(SET TOP BIT TO GET ALL CHAR. HANDLING)
        STA     2,ATRIB  ;SAVE THIS PORT'S PCB ADDRESS
        ---     ---      ;FINISH INITIALIZING
        ---
        JMP     @TEMP    ;ERROR RETURN (STATUS IN A3)
        ---
        ISZ     TEMP
        JMP     @TEMP    ;DEVICE SUCCESSFULLY OPENED

FINIS:  ---     ---      ;WRAP-UP (CLOSE) SUBROUTINE (OPTIONAL)
        JMP     0,3      ;(SKIP RETURN OK, NO DIFFERENCE)

WRITE:  ---     ---      ;DATA OUTPUT SUBROUTINE (OPTIONAL)
        JMP     1,3      ;(NON-SKIP WITH STATUS IN A3 IF ERROR)

READ:   ---     ---      ;DATA INPUT SUBROUTINE (OPTIONAL)
        JMP     1,3      ;(NON-SKIP WITH STATUS IN A3 IF ERROR)

ATRIB:  0       ;PCB SUPPLIED BY MUX DRIVER
        0       ;NO MASK
        0       ;NO DEVICE ADDRESS
                ;LINKAGE POINTER TABLE HERE IF REQUIRED
        -1      ;LINKAGE POINTER TABLE TERMINATOR
        -1      ;PORT DEFINITION TABLE TERMINATOR (NO PORTS)

        .END    ;END OF DRIVER (See Figure 11.1 for ATRIB test)
```

11.8 How to Write a System Disc Driver

All checks for legal disc and core addresses and the decision to retry on an error are handled by the system's read/write block routine. The only task of a disc driver is to issue the instructions to read or write one or more blocks of 256 words at the given disc and core addresses. Refer to the Glossary in Appendix 3 for definitions of terms used here.

The disc driver will be called with the registers containing:

A0	Pointer to LUVAR table
A1	First Real Disc Address
A2	First core address
A3	Pointer to block count and retry counter
C	Zero for read, or one for write

The block count at (A3) will be one or, if consecutive disc and core addresses are to be transferred, the number of such consecutive blocks. The driver must convert the Real Disc Address to the Physical Disc Address if they are different, issue the instructions to the disc controller to transfer one or more blocks, and return to the second location following the block count (equivalent to a JMP 2,3 instruction) with the number of blocks being transferred in register A1. The values returned in the other registers are immaterial. See page 11-20 for other notes, including power fail restart provisions.

The driver uses the information in its LUFIX table (Logical Unit Fixed Information) at the beginning of the driver (see Figures 11.5 and 11.6) and in the LUVAR table (Logical Unit Variable Information) at the location given in register A0 to compute the physical disc address. CRLA (see IDRV on page 11-17) may be used to assist in this conversion. The form of a LUVAR table is:

Disp.	Label	Contents
0	NCYL	Number of cylinders
1	PART	Partitioning information
2		Partitioning information
3		Reserved (do not use)
4	AVBC	Available block count
5	MINB	Min blocks for building new file
6		(spare)
7	FUDA	First unused Real Disc Address
10	ERRC	Data check error count
11		Address check (seek) error count
12		Data channel late error count

The partitioning information in words PART and PART+1 specifies the location of the Physical Unit, and the form of the information in these words is determined by the disc driver itself. Usually, the word at PART is used to determine the location of the Physical Unit (which drive and cartridge), and the word at PART+1 is used for a starting cylinder number if that Physical Unit is to be divided into two or more Logical Units. The same PART words must also work for the corresponding BZUP driver without change.

AVBC is set by SIR to be equal to the number of disc blocks currently available to be allocated (not currently in use) on the Logical Unit. Initially, AVBC = NCYL*NTRK*NSCT.

MINB is the minimum value of AVBC to allow building a new file. Thus, MINB blocks will be reserved for expanding old files.

FUDA is set by SIR to be equal to the first Real Disc Address beyond the end of this Logical Unit; ie, FUDA = NCYL*LRCC. It is used by the system to determine whether the Real Disc Address supplied by the caller is too large. CAUTION!! FUDA must not exceed 100000 octal for logical unit zero.

ERRC is the first of three error count words. The appropriate one will be incremented by the system whenever a corresponding error is detected.

The LUFIX table is assembled with the driver, just preceding the driver's entry point. Its contents are addressed by use of negative displacements from the entry point as follows:

Disp.	Label	Contents
-24	DINT	(reserved for future use)
-23	DMSK	(reserved for future use)
-22	DSIZ	Size of driver (# words)
-21	PFRD	Power fail restart delay
-20	EMSK	"Any error" status mask
-17		"Write protected" mask
-16		"No such disc" mask
-15		"Data channel late" mask
-14		"Disc address check error" mask
-13		"Illegal disc address" mask
-12	IDRV	"Initialize driver" subroutine entry
-11	SLUR	"Skip if LU ready" subroutine entry
-10	SKNB	"Skip if not busy" subroutine entry
-7	REDS	"Read status" subroutine entry
-6	SEEK	"Seek or recalibrate" subroutine entry
-5	NSCT	Number of sectors (blocks per track)
-4	NTRK	Number of tracks (per cylinder)
-3	LRTC	Logical-to-real track conversion
-2	LRCC	Logical-to-real cylinder conversion
-1	DFLG	Flag word (see below)
0		Driver's read/write entry point

DINT and DMSK are not used in the current version of IRIS; these cells should contain zeroes. DSIZ is used by SIR when bringing the driver into core from the CONFIG file. SIR will replace the value in DSIZ with a pointer to the LUFIX table; this is for use by the driver if it is necessary to call CRLA (Convert Real to Logical Address). There is a pointer to CRLA in A0 when the driver's initialize subroutine is entered.

Figure 11.5: Typical Driver for Fixed Head Disc

```
FD1      =20        ;DEVICE ADDRESS

         0          ;(RESERVED FOR FUTURE USE)
         0          ;(RESERVED FOR FUTURE USE)
     FD1RS-.+2      ;SIZE OF DRIVER
         5000.      ;POWER FAIL RESTART DELAY
         1          ;"ANY ERROR" STATUS MASK
         20         ;"WRITE PROTECTED" MASK
         4          ;"NO SUCH DISC" MASK
         10         ;"DATA CHANNEL LATE" MASK
         0          ;"ADDRESS CHECK ERROR" MASK
         0          ;"ILLEGAL DISC ADDRESS" MASK
     JMP 0,3        ;"INITIALIZE DRIVER" SUBROUTINE
     JMP 0,3        ;"SKIP IF LU READY" SUBROUTINE
     JMP FD1SN      ;"SKIP IF NOT BUSY" SUBROUTINE
     JMP FD1RS      ;"READ STATUS" SUBROUTINE
     JMP 0,3        ;"SEEK OR RECALIBRATE" SUBROUTINE
         10         ;NUMBER OF SECTORS (BLOCKS PER TRACK)
         100        ;NUMBER OF TRACKS (PER CYLINDER)
         10         ;LOGICAL-TO-REAL TRACK CONVERSION FACTOR
         1000       ;LOGICAL-TO-REAL CYLINDER CONVERSION FACTOR
      2000+FD1      ;DISC ALLOCATION INFO, DEVICE ADDRESS

FD1E:    DOBC       2,FD1     ;DRIVER ENTRY POINT
         MOV        0,2
         LDA        0,PART,2  ;FIRST REAL CYLINDER
         INTDS
         LDA        2,1,3     ;RETRY COUNTER
         NEGL#      2,2,SNC   ;HAS POWER FAIL OCCURRED ?
         JMP        .+6       ;  YES, DON'T START
         ADD        0,1,SZC   ;READ OR WRITE ?
         JMP        .+3
         DOAS       1,FD1     ;  READ
         JMP        .+2
         DOAP       1,FD1     ;  WRITE
         SUBZL      1,1       ;ONE BLOCK TRANSFERRING
         JMP        2,3       ;RETURN

FD1SN:   SKPBZ      FD1       ;SKIP IF NOT BUSY
         JMP        0,3
         JMP        1,3

FD1RS:   DIAC       0,FD1     ;READ STATUS
         JMP        0,3
```

PFRD indicates the amount of time that the disc drive requires after a power failure before it will again be ready for disc transfers. To calculate the value for PFRD, use the formula

PFRD = (T x 3500) / (R + 22)

where T is the typical start up time of the disc drive in seconds, R is the execution time in microseconds of the driver's "skip if ready" subroutine (excluding the JMP in the LUFIX) assuming a Nova 1200 or a D116 computer; the system will compensate for the speed difference if so indicated by the SPEED value in the INFO table. The values 3500 and 22 are decimal constants. The integer value of the result should be converted to octal and used as the value for PFRD. This formula provides approximately 50% extra delay to allow for variations in CPU speed and disc start up time. PFRD must be zero if operator intervention is required to restart the disc after a power failure, such as for a Diablo drive with the "write protect" option.

EMSK must contain one or more "one" bits to produce a non-zero result when ANDed with the status word returned by the REDS subroutine if an error of any type has been detected. The next five words are similar masks for specific types of errors; if an error is indicated by a non-zero result when EMSK is ANDed with the status word, and none of the other masks produce non-zero when ANDed with the status word, then a data check error is assumed.

IDRV must contain a jump to a driver's initializing subroutine if the pointer to CRLA (Convert Real to Logical Address) given in A0 must be saved or if any initializing is required on initial start up or after a disc error, a disc time-out, or a power failure. If no initialization is required then IDRV may contain a JMP 0,3 instruction.

SLUR must contain a jump to a subroutine that will test whether the Logical Unit (identified by the LUVAR pointer given in register A2) is on line and ready, and so indicate by a skip return. A non-skip return indicates that the unit is not on line, not up to speed, or the controller does not provide for a ready test. Ready may be indicated even if the drive is busy. The SLUR cell must contain a JMP 0,3 instruction if the disc controller does not provide for testing whether an individual drive is ready.

SKNB must contain a jump to a subroutine that does a skip return if the disc controller is ready and is not busy, or a non-skip if it is busy or not ready. SKNB must not change register A2.

REDS must contain a jump to a subroutine which reads the controller status into register A0; if two or more status words are provided by the controller then this subroutine must combine the significant bits into one word.

Copyright 1978
Educational Data Systems

SEEK must contain a jump to a "seek or recalibrate" subroutine which will initiate a seek to the cylinder identified by the Real Disc Address given in register A1 or, if (A1)=-1, do a recalibrate and wait for it to be completed. SEEK must not change register A2 which contains the LUVAR pointer. Only certain discs require this routine; in other cases, SEEK may contain a JMP 0,3 instruction.

The next four items define the physical configuration of the disc for mapping and allocation purposes. NSCT indicates the number of sectors (number of blocks per track), not to exceed 16 (octal 20) sectors. NTRK indicates the number of tracks per clinder (number of heads), LRTC indicates the Logical-to-Real Track conversion factor, and LRCC indicates the Logical-to-Real Cylinder conversion factor. LRTC must equal NSCT, and LRCC must equal NTRK*NSCT. For example, suppose a disc cartridge has 203 cylinders, two surfaces, and 12 sectors. The corresponding constants for the LUVAR and LUFIX tables would be (in octal): NCYL=313, NTRK=2, NSCT=14, LRTC=14, and LRCC=30. AVBC will be calculated by SIR as NCYL*NTRK*NSCT=11410, and FUDA will be calculated by SIR as NCYL*LRCC=11410. FUDA must not exceed 100000 octal for Logical Unit zero, but may be as large as 177776 octal for other Logical Units. Discs having more than 16 sectors must be specified otherwise; for example, a disc with five tracks and 32 sectors should be specified as ten tracks and 16 sectors.

DFLG is a flag word made up as follows:

Bit(s)	Meaning		
15	Changeable cartridge		
14	Use ALLOCATE for active files		
13,12	(unused)		
11	Skip sector between tracks within cylinder		
10	Same sector, next track	\	Next best block
9	Next sector, next track	)	if desired block
8	Next sector, same track	/	is not available
7	Skip sector between data blocks		
6	Skip sector after header block		
5-0	Device address		

Bit 15 should be a one if the Logical Unit is on a changeable cartridge. Bit 14 should be a one if the drive has a head per track and the heads do not move or if it is a large storage module which does not require active files to start on cylinder boundaries for efficient swapping. Bits 11-6 define allocation parameters so that files can be transferred in or out of core in the minimum time. Bit 11 should be set if the controller will not automatically cross track boundaries on multiple block transfers. Bit 10 would only be set for a head-per-track disc. Bit 9 might be set for a floppy disc drive where the track to track seek is faster than one sector latency. Bit 8 would be set for any other moving arm disc. One and only one of bits 8-10 should be set. Bit 7 should be set if the controller is incapable of transferring consecutive sectors. Bit 6 should be set if there is not enough time afer reading one block to use information from that block for transferring the next consecutive block.

Figure 11.6: Typical Driver for Moving Head Disc

```
      MD2L-.+1     ;SIZE OF DRIVER
          0        ;CAN'T RECOVER FROM POWER FAIL
          1        ;"ANY ERROR" STATUS MASK
          200      ;"WRITE PROTECTED" MASK
          100000   ;"NO SUCH DISC" MASK
          10       ;"DATA CHANNEL LATE" MASK
          4        ;"ADDRESS CHECK ERROR" MASK
          60       ;"ILLEGAL DISC ADDRESS" MASK
      JMP 0,3      ;"INITIALIZE DRIVER" SUBROUTINE
      JMP MD2SR    ;"SKIP IF LU READY" SUBROUTINE
      JMP MD2SN    ;"SKIP IF NOT BUSY" SUBROUTINE
      JMP MD2RS    ;"READ STATUS" SUBROUTINE
      JMP MD2SC    ;"SEEK OR RECALIBRATE" SUBROUTINE
          16       ;NUMBER OF SECTORS (BLOCKS PER TRACK)
          2        ;NUMBER OF TRACKS (PER CYLINDER)
          16       ;LOGICAL-TO-REAL TRACK CONVERSION FACTOR
          34       ;LOGICAL-TO-REAL CYLINDER CONVERSION FACTOR
      @500+MD2     ;DISC ALLOCATION INFO, DEVICE ADDRESS

MD2E:   DOBC    2,MD2      ;DRIVER ENTRY
        MOV     0,2
        LDA     0,0,3
        DOC     0,MD2      ;BLOCK COUNT
        LDA     0,PART,2   ;FIRST REAL CYLINDER
        ADD     0,1        ;ADD TO DISC ADDRESS
        SUBCR   0,0        ;READ OR WRITE COMMAND FROM CARRY
        ADD     0,1        ;COMBINE WITH DISC ADDRESS
        INTDS
        LDA     0,1,3
        NEGL#   0,0,SZC    ;HAS POWER FAIL OCCURRED ?
        DOAS    1,MD2      ;  NO, START TRANSFER
        LDA     1,0,3      ;ALL BLOCKS TRANSFERRING
        JMP     2,3        ;RETURN

MD2SR:  DIB     0,MD2      ;SKIP IF READY
        LDA     1,PART+1,2
        AND#    0,1,SNR
        JMP     0,3
        JMP     1,3

MD2SN:  SKPBZ   MD2        ;SKIP IF NOT BUSY
        JMP     0,3
        JMP     1,3

MD2RS:  DIAC    0,MD2      ;READ STATUS
        JMP     0,3

MD2SC:  INC#    0,0,SZR    ;SEEK OR RECALIBRATE
        JMP     0,3        ;  SEEK (NOT IMPLEMENTED)
        DOAS    0,MD2      ;  RECALIBRATE
        SKPBZ   MD2
        JMP     .-1
MD2L:   JMP     0,3
```

Just before issuing the final command to start the data transfer, a system disc driver must disable interrupts and test whether a power failure has occurred by examining the retry counter (see examples). If the retry count is ≤0, then a power failure has occurred, and the transfer must not be started; a two-skip return is performed without regard to the contents of any register. Note that the driver does not wait for the transfer to be completed, and all code must be position independent.

11.9 How to Write A Disc Driver for BZUP

BZUP requires a simple disc driver that will transfer one disc block on one particular Logical Unit, wait for the transfer to complete, check status, and skip return if no errors occur. It must do a non-skip return with the disc status in register A0 if any type of error is detected or with zero in A0 if a time-out occurred.

The driver will be given a Real Disc Address in register A1 and a core address in A2. These registers must not be changed by the driver. Each Logical Unit will have its own copy of BZUP with partitioning constants for converting the Real Disc Address to the corresponding Physical Disc Address. These partitioning constants must be identical to the PART and PART1 words in the disc driver table (see Section 2.6). BPCxx is at address 300 in BZUP (location 24300 when BZUP is in core at its usual location). See Figure 11.7 for a typical BZUP disc driver. The .DMR pseudo-op line is included to check that the driver does not exceed the available space in BZUP; there are 61 words (octal) available for the driver.

If the driver will not fit in 61 words then it may also occupy much of the space below BPART (see page 11-22), but in this case BZUP will not have any utility commands available. BZUP is still required, however, for the IPL and for use by DBUG and SHUTDOWN.

Figure 11.7: Typical Disc Driver for BZUP

```
BPCF1:  0   ;PARTITIONING CONSTANTS MUST BE SAME AS IN DRIVER TABLE
        0

        DOAS     0,FD1
BRBFI:  LDA      0,.-1      ;READ A DISC BLOCK
        JMP      BWBF1+1

        DOAP     0,FD1
BWBFI:  LDA      0,.-1      ;WRITE A DISC BLOCK
        STA      0,.+4
        DOBC     2,FD1      ;OUTPUT CORE ADDRESS
        LDA      0,BPCF1
        ADD      1,0        ;ADD PARTITIONING CONSTANT
        DOA      0,FD1      ;OUTPUT DISC ADDRESS AND START
        SUB      0,0
        INC      0,0,SNR
        JMP      0,3        ;TIME OUT !
        SKPBZ    FD1
        JMP      .-3
        DIAC     0,FD1      ;READ STATUS, CLEAR "DONE"
        MOVR#    0,0,SZC    ;ANY ERROR ?
        JMP      0,3        ;  YES
        JMP      1,3        ;  NO

.DMR    BZF10    =JMP BPCF1+BSIZE-. ;BZUP OVERFLOW TEST

        ************************************************
        *                                              *
        *                 CAUTION !!                   *
        *                                              *
        *   BPCxx must be at location 300 of BZUP      *
        *   BRBxx must be at BPCxx+3                   *
        *   BWBxx must be at BPCxx+6                   *
        *   All code must be position independent      *
        *   Registers A1 and A2 must not be changed    *
        *                                              *
        ************************************************
```

11.10 How to Write a Terminal Translation Module

Interface to System — A terminal translation module is the interface between terminal-independent IRIS terminal control functions and a specific type of interactive terminal. The driver is a standard IRIS System Subroutine Module as described in section 11.4 except that it does not have an initialization routine. Location BPS+5 must contain a JMP 0,3 instruction, and the first core-resident location must be at BPS+6.

Each terminal on a system has its own terminal type code (TTC). This is the number specified to the system when a particular driver is selected for use on a given port. This number ranges from 0 to 144 octal, where type 0 means no driver is selected and default character processing is desired.

A driver identifies itself to the system during an IPL by setting a pointer to its first core-resident location at location 400+TTC. This must be done using the pointer linkage table which starts at ATRIB+3 in a system driver. Terminal drivers are the only drivers allowed to put pointers in this area with the pointer linkage table. Thus the system can determine each terminal driver's location and its TTC.

There is a maximum of 15 (decimal) terminal drivers allowed to be activated on a system at a given time. A TRAP #136 will occur if more than the maximum are activated. Also, if any terminal drivers are activated, the system driver $TERMS must also be activated.

Each code that a terminal translation module will be required to translate on output is initiated with a single byte whose value is less than 200 octal. The majority of output codes consist only of the single byte, but a cursor positioning request consists of a 177 byte followed by a number of bytes (depending on the terminal), which are greater than 0 and less than 377, terminated by a 377 byte. Most codes when sent out have no special significance to the system except for the 'RD' (read cursor) code. This code sets the EC (expecting cursor) flag in the TTN word of the user's PCB when it is sent out.

The mnemonics and the octal values for all of the terminal control codes are defined in a file called "CRTDEFS". A number of flags in the TTN word that a terminal driver might need to concern itself with are also defined there. This file is included on every system, and a listing of CRTDEFS is included at the end of this section.

Internal Structure -- In addition to the fact that a terminal translation module has the external structure of a System Subroutine Module, it has additional structural features that must be closely followed. Internally, starting with the first core-resident location, a terminal translation module consists of an eight-word header followed by various translation tables and procedures. The structure of the tables and calling sequences for the procedures will be discussed under the appropriate sections on translations. The eight-word header will be discussed here.

Word 1 -- This word contains the TTC for the driver. In addition, the most significant bit of the word is a special parity bit which tells the system how to handle parity errors on input. If the bit is off, normal parity processing is done. If it is on, the 7 low-order bits of an input character will come in, regardless of whether a parity error occurred or not. The top bit of an input character will be 0 if a parity error occurred or 1 if no parity error occurred. This special processing allows the parity of an incoming character to be used as information.

Word 2 -- Input escape code. If there are any escape sequences generated on input, the escape code used to prefix these sequences must be put here with the 200 bit set. In most cases this code will be the standard 233 escape, but some terminals may use another code. This code must be less than 240 octal.

Word 3 -- Virtual pointer* to input translation table for characters following the input escape code.

Word 4 -- Virtual pointer to input translation table for characters that don't follow the input escape sequence code.

Word 5 -- Output escape character. If any control codes require a translation to an escape sequence for output, the code used to prefix these sequences must be put here with the 200 bit set. In most cases this code will be the standard 233 escape, but some terminals may use another code.

Words 6 & 7 — Special delay characters. If there are any characters which require a delay when output, they should be specified here. The character is in the right half of the word and the delay required is in the left half. The value of the delay is in fiftieths of a second if the port is through an EDSI multiplexer, or the number of nulls to be sent out otherwise. Note that the character specified is the actual code output, not the internal code before translation (if any). Thus, if a terminal requires a delay after an escape sequence (eg, after a clear screen), the delay must be handled by a procedure which inserts an appropriate number of nulls.

Word 8 -- Virtual pointer to output translation table.

* A "virtual pointer" is an absolute pointer as far as the source file is concerned. It is relocated to the real core address of the table at IPL time by $TERMS. If the table is empty, the pointer must point to a null table which is simply a zero entry.

Output Translation -- In general, translation is table driven. The output translation table consists of a single word giving the number of table entries that follow, followed by entries of the form:

```
|---|-----------|-----------|-----------|-----------|-----------|
| E |           TRANS           | P |           GIVEN           |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
```

The table must be sorted in ascending order by the 7-bit GIVEN field.

All bytes output which are greater than or equal to 200 are treated normally and are output without translation by the system. If a byte less than 200 is output, the system will then search the output translation table's GIVEN field for the byte. If the search fails, an "↑" is output instead of the byte. If the search succeeds, action depends on the value of the P-bit. If P = 0, the 7-bit TRANS is output with the high-order eighth bit unconditionally set. If the P = 1, the 8-bit TRANS + E field is interpreted as an offset from the table entry to a procedure. The procedure is then called as described below.

Output Translation Procedures -- When an output translation procedure is called, A0 contains the character to be processed and A2 contains a pointer to the caller's PCB. A3 contains the return address - 1 which also points to a routine to store bytes in the caller's I/O buffer.

So as the procedure needs to put bytes into the I/O buffer, it merely needs to do JSR's via the value that the procedure was called with in A3, with the byte to be output in A0. WARNING -- these calls to the store byte routine clobber all registers.

When the procedure is finished with it's translation, it should do a skip-return with respect to the value it was called with in A3. The procedure should return with a JSR instruction rather than with a JMP instruction (see below). If the procedure returns with A0 = -1, then an error was detected in the translation and an "↑" is stored in the I/O buffer. If A0 > 0 then the character in the right half of A0 is stored in the I/O buffer. If A0 = 0 then nothing is stored in the I/O buffer.

If the procedure requires more than one character to complete its translation (as with a cursor positioning request), it may do a "resume" operation instead of a "return" operation. One does this by setting the OTIP (output translation in progress) flag in the TTN word of the caller's PCB. This causes the system to resume operation of the procedure at the instruction following the return (hence the return with a JSR instead of a JMP) with the next character in A0 and a fresh PCB pointer in A2. It is up to the procedure to clear the OTIP flag when it is done with the translation.

Something to note about the return address that is given to a procedure when it is called is that it will always be the same for each call or resume to the procedure, no matter when it is called. Hence the return address may be preserved in a temporary cell that is local to the procedure on the first call and it will always be good thereafter.

Another small detail is that for output translation procedures, RTP will always be a valid PCB pointer, so the driver doesn't need to save it locally if it needs to preserve it.

Cursor Positioning -- A cursor positioning request is always a 177 byte followed by an X-coordinate (column) followed by a Y-coordinate (row) followed by an additional unspecified number (usually zero) of characters which depends on the terminal, terminated by a 377 byte. The X and Y coordinates will come to the driver as origin (1,1) where (1,1) is the topmost leftmost position on the screen (this corresponds to origin (0,0) in RUN). Hence most drivers will need to decrement the coordinates to origin (0,0) before sending them to the terminal. Every time a cursor position goes out from RUN, the OCC (output column counter) is updated to the value of the X-coordinate by the RUN processor.

The unspecified number of additional coordinates are used by certain terminals for control functions. Their use is discouraged as much as possible since any program that uses them becomes terminal dependent. Hence, most terminal drivers will only need to process two coordinates; but they should, however, give an error if the wrong number of coordinates is sent.

Input Translation -- Input translation is handled in a fashion quite similar to output translation. There are two input translation tables, however: one for characters that follow the input escape sequence character, and one for characters that don't. Each table consists of a single word giving the number of entries in the table followed by entries of the form:

```
|---|-----------|-----------|-----------|-----------|-----------|
| P |          TRANS         |                GIVEN               |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
```

The table must be sorted in ascending order by the 8-bit GIVEN field.

The input translation table for characters not following the terminal's escape sequence character will be searched any time an input character less than 240 is encountered. If an escape has been seen, the table for characters that do follow the terminal's escape sequence character will be searched no matter what the next character is. If the character is found, the action taken is based on the value of the P-bit. If P = 1, TRANS is interpreted as a 7-bit character to be translated to. It goes in with the high-order eighth bit unconditionally set to one. If P = 0, TRANS is interpreted as a 7-bit offset relative to the table of a procedure. The procedure is then called as described below. If the character is not found (whether or not an escape has been seen) it is passed on to the system as is.

Input Translation Procedures -- When an input translation procedure is called, A0 contains the character to be processed, and A2 contains a pointer to the PCB from which the input came. A3 contains the return address.

On return from the procedure, if A0 = 0, then nothing is given to the system to process. If A0 > 0 then the character in the right half of A0 is given to the system to process. If A0 = -1 then the character is to be treated as an EOM (end of message) and the system will terminate input. The procedure should be exited with a JSR via the value that was in A3 when the procedure was called.

If more than one character is required to complete a translation (as with a read cursor position command), the procedure may, as with output, "resume" the system instead of "returning" to it. This is done by setting the ITIP (input translation in progress) flag in the TTN word of the PCB before returning. When the next character comes in, the procedure will be resumed at the instruction following the return, with the next character to be processed in A0 and a fresh PCB pointer in A2. It is up to the procedure to clear the ITIP flag when it is done translating.

As with output procedures, the return address for input procedures will always be the same for everyone every time the procedure is called. Hence, one can save it locally once, and it will always be good. One should be cautioned, however, that RTP is not a valid PCB pointer. If one needs to preserve it, it must be saved locally.

Reading Cursor Position -- When a read cursor command ('RD') goes out to a terminal, the system sets the EC (expecting cursor) flag in the TTN word. A procedure to process the incoming cursor position from the terminal should make sure this flag is set before processing the position. This is to make sure the operator hasn't accidently hit the correct characters to generate an incoming cursor position. Once the driver has made sure this flag is set, it should clear it.

The system has input enabled when it expects the cursor position to come in. It expects the X-coordinate (column), followed by the Y-coordinate (row), origin (0,0) where (0,0) is the topmost leftmost position on the screen. These two bytes should be stored directly in the first two locations of the I/O buffer. The driver should resume the system with A0 = -1 to terminate input.

Reentrancy and Storage Requirements -- Since these drivers may be used by different ports at the same time, they must be mutually and individually reentrant. If any temporaries are needed to be preserved, they must (with the exception of the return addresses as previously mentioned) be saved on a port by port basis. A free node may be acquired by the FREENODE command and linked to the port's PCB by putting a pointer in the NLP cell which is reserved for this purpose. Multiple nodes may be linked together if more storage is required. The last word of a node is reserved for a linkage pointer to the next node in the chain (if more nodes are required) and must be zero in the last node of the chain. The remaining words in these nodes may be freely allocated between input and output procedures, but the input and output procedures must be mutually reentrant; ie, they must never share temporary storage cells.

```
;"CRTDEFS" -- DEFINITIONS OF CRT CONTROL CODES FOR "IRIS" 7.3
; 2-3-78

;                   All Rights Reserved
;     Copyright © 1978, Educational Data Systems
;    This document may not be reproduced without the
; prior written permission of Educational Data Systems


        ET =3    ;ETX code
        RB =7    ;ring bell
        ML =10   ;move left
        LF =12   ;line feed
        VT =13   ;vertical tab
        FF =14   ;form feed
        CR =15   ;carriage return
        MH =17   ;move home
        CS =20   ;clear screen
        MR =40   ;move right
        RD =41   ;read cursor position
        CU =43   ;clear unprotected
        CL =44   ;clear to end of line (unprotected)
        CE =45   ;clear to end of screen (unprotected)
        MD =52   ;move down
        MU =53   ;move up
        BB =60   ;begin blink
        EB =61   ;end blink
        BR =62   ;begin reverse video
        ER =63   ;end reverse video
        BD =64   ;begin dimming
        ED =65   ;end dimming
        BP =66   ;begin write protect
        EP =67   ;end write protect
        BU =70   ;begin underline
        EU =71   ;end underline
        BX =72   ;begin expanded print
        EX =73   ;end expanded print
        FM =74   ;enter format mode
        FX =75   ;exit format mode
        LK =76   ;lock keyboard
        UK =77   ;unlock keyboard
        BT =100  ;begin transmission from CRT memory
        MP =101  ;use memory pointer instead of cursor
                 ;  for next positioning.
        AT =177  ;"@" -- start of multi-byte sequence
                 ;  (usually a cursor positioning request)
                 ;  which is terminated by a 377 code.


 ; SOME FLAG DEFINITIONS IN TTN. WORD:
        EC   =20  ;EXPECTING.CURSOR
        ITIP =40  ;INPUT.TRANSLATION.IN.PROGRESS
        OTIP =100 ;OUTPUT.TRANSLATION.IN.PROGRESS
        ESCS =200 ;ESCAPE.SEEN

        .EOT     ; "CRTDEFS" R7.3 SOURCE
```

12. SYSTEM ASSEMBLIES

All components of IRIS should be assembled using the absolute assembler or (preferably) the ASSEMBLE processor. The SYMBOLS source tape must be used as the first source tape on pass one of the first assembly if ASSEMBLE is not used.

12.1 Software Definitions Tape (DEFS)

This tape defines such things as the structure of tables, control words, and file headers. It also includes various definitions and displacements which are used throughout the system. If this tape is changed, then all system components must be assembled with the new definitions. The Software Definitions tape is required only on pass one of an assembly.

12.2 Page Zero Definitions Tape (PZ)

Most of the pointers, constants and flags in page zero of REX are available for use by processors and subroutines. The Page Zero Definitions tape defines the locations of these cells when assembling system components other than REX. If any change which affects these definitions is made in page zero of REX then this tape must be modified accordingly, and all other system components must be re-assembled. The Page Zero Definitions tape is required only on pass one of an assembly.

12.3 How to Assemble System Components

To assemble any IRIS system component other than REX or BZUP, feed the source tapes to the assembler in the following sequence:

```
Pass 1:  Software Definitions (DEFS)
         Page Zero Definitions (PZ)
         Component source tapes

Pass 2:  Component source tapes
```

For REX or BZUP the sequence is the same except that the Page Zero Definitions tape is omitted. The Software Definitions and Page Zero Definitions are not necessary on pass 2 but may be included if desired for listings. An assembly of REX, PLOAD, SIR, and/or SYSL requires all of the DEFS, REX, PLOAD, SIR, and SYSL source tapes, in that order; PZ must not be included.

BASIC, RUN, and RUNMAT also require a BASIC/RUN Definitions tape (BRDEFS) which should follow the Page Zero Definitions on pass 1 and may be included on pass 2 if a listing of it is desired. The same definitions tape must be used for all three of these processors which, together with DISCSUBS Group 2, comprise Business BASIC. Note that the last two source tapes of RUN are also used as the last two RUNMAT source tapes.

Copyright 1978
Educational Data Systems

APPENDICES

Category #1 -- Interactive I/O Subroutines. These must be used to perform all input and output on an interactive port. Most are group 1, and are thus available on all IRIS systems.

ACCESS INPUT BYTE accesses the next non-space from the regnant user's input buffer. See Section 9.7 for calling sequence.

ACCESS INPUT STRING BYTE accesses the next byte from the regnant user's input buffer. See Section 9.7 for calling sequence.

CONVERT DRATSAB TO ASCII converts a string of bytes in DRATSAB code (compressed Hollerith) into the corresponding ASCII codes. DRATSAB and ASCII codes can be found in appendices of the IRIS User Reference Manual. The calling sequence is CALL CDTA with the byte address of the DRATSAB string in A1 and a pointer to an Item Control Block (ICB) in A2. The ICB is the same as described for Read Item in Category #3. A0 must contain zero if the cards are punched in Hollerith standard key punch codes or non-zero if the cards are marked as shown on the EDS BASIC Card Programmer. On return, A0 contains the number of bytes transferred, and A3 contains the status. Group 3.

CONVERT INTEGER TO ASCII outputs a binary number to the regnant user's output buffer after converting it to a specified radix. See Section 9.7 for calling sequence.

DECIMAL INPUT uses the Access Input Byte subroutine to scan ASCII codes from the user's input buffer, converts the string to floating point decimal, and leaves the result in the Decimal Accumulator (DA). The calling sequence is DECIMAL with 6 in A0 and 0 in A1. Normal return is skip with A0 containing the number of significant digits scanned, and the terminating code in A2. Return is non-skip with zero in DA and the terminating character in A2 if no digits are found. Group 2.

DECIMAL OUTPUT converts the value in the Decimal Accumulator (DA) into an ASCII string representing the value in standard form, and uses the Store Output Byte subroutine to store the string in the regnant user's output buffer. The calling sequence is DECIMAL with 7 in A0 and 0 in A1. The output string will consist of either a space if (DA) is positive or a minus sign if (DA) is negative, a string of up to 14 decimal digits, and an imbedded period if required. Defaults to floating form if the value in DA is less than 0.1 or is greater than 99,999,999,999,999 decimal. Group 2.

DECIMAL OUTPUT CLOSE SPACED is the same as Decimal Output above, except that there will be no leading space if the value is positive. The calling sequence is the same except that A0 must contain 10 octal. Group 2.

Copyright 1978
Educational Data Systems

DECIMAL OUTPUT, FORMATTED is the same as Decimal Output above except that A0 must contain octal 11, A1 must contain zero, and A2 must contain the byte address of a format string. The value in DA is output in the format specified by that string as described in the PRINT USING statement in the IRIS User Reference Manual. Normal return is skip, but a non-skip return will occur if an error is detected in the format string. In either case, the byte address of the first unused byte of the format string is returned in A2. Group 2.

ERROR outputs the message "ERROR #n" to the regnant user's output buffer, where n is any integer from zero to 177 octal. The calling sequence is CALL ERROR with the value n*K!NOP in register A0. The values K=400 and NOP=100010 octal are assigned in the Software Definitions. Group 2.

MESSAGE outputs a canned message from the MESSAGES file to the regnant user's output buffer. See Appendix 2 for calling sequence and available messages.

OUTTEXT outputs a given text string to the regnant user's output buffer. See Section 9.7 for calling sequence.

QUEUE CHARACTER queues an input character to be processed and entered in a port's input buffer, or requests an output character from a port's output buffer, or queues an interrupt task to be executed. See Section 11.6 for calling sequences. Core resident.

START INPUT enables input from the regnant user's terminal into the user's input buffer. See Section 9.7 for calling sequence.

START OUTPUT initiates output from regnant user's output buffer to the user's terminal. See Section 9.7 for calling sequence.

OUTPUT BYTE stores a byte in the regnant user's output buffer. See Section 9.7 for calling sequence.

WAIT FOR OUTPUT NOT ACTIVE assures that a previous output has been completed. See Section 9.7 for calling sequence.

Category #2 -- Mathematical Functions. These arithmetic and transcendental subroutines are available for use by a task or processor. Note that most are group 2, and thus are available only on IRIS systems including Business BASIC.

ADD DECIMAL INTEGERS adds two unsigned 4-digit BCD (Binary Coded Decimal) integers. The calling sequence is LDA 3,.DEC followed by JSR @-2,3 with the augend in A0 and the addend in A1. The sum will be returned in A0, and the carry out will be in bit 15 (the most significant bit) of A2. For multiple precision addition, the carry may be propagated by incrementing the addend in A1 (the least significant BCD digit of the addend may be ten). Group 2.

BINARY DIVIDE forms the 16-bit quotient of two unsigned 16-bit binary integers and also returns the 16-bit remainder. The calling sequence is BINDIVIDE with the divisor in A0 and the dividend in A1. The quotient is returned in A3 and will be equal to the integer portion of dividend/quotient. A1 will contain the remainder which will be less than the divisor, which is returned unchanged in A0.

BINARY MULTIPLY forms the 32-bit product of two unsigned 16-bit binary integers. The calling sequence is BINMULTIPLY with the multiplier in A0 and the multiplicand in A2. The 32-bit product is returned in registers A0 and A3 with the most significant half in A3. Registers A1 and A2 are unchanged.

BREAK DECIMAL NUMBER separates the floating point value in the Decimal Accumulator (DA) into its integer and fractional parts. The calling sequence is DECIMAL with octal 20 in A0. The fractional portion is left in DA and is normalized, and the integer portion of the value is copied into the Decimal Buffer area (DB) in six-word unpacked form. Group 2.

DECIMAL ADD adds the value from a specified location in core to the Decimal Accumulator (DA), and leaves the sum in DA. The calling sequence is DECIMAL with 3 in A0, the number type of the augend in A1*, and the core address of the augend in A2. Group 2.

* The number types are as follows:

A1	Number Type	Remarks
0	unsigned decimal integer	range 0 to 9999
1	signed decimal integer	range ±7999
2	2-word floating decimal	six digit mantissa
3	3-word floating decimal	ten digit mantissa
4	4-word floating decimal	14 digit mantissa
5	6-word unpacked decimal	15 digit mantissa

Add 10 octal to the number type for reverse divide or reverse subtract. Octal 10 may be added to the number type for other decimal operations without bad effect. See Figure 1.6 for details on the number types.

Copyright 1978
Educational Data Systems

DECIMAL DIVIDE divides the value in the Decimal Accumulator (DA) by the value at a specified location in core, and leaves the quotient in DA. The calling sequence is DECIMAL with 4 in AO, the number type of the divisor in A1*, and the core address of the divisor in A2. Group 2.

DECIMAL DIVIDE REVERSE divides the value at a specified location in core by the value in the Decimal Accumulator (DA), and leaves the quotient in DA. The calling sequence is DECIMAL with 4 in AO, the number type of the dividend plus 10 octal in A1*, and the core address of the dividend in A2. Group 2.

DECIMAL MULTIPLY multiplies the value in the Decimal Accumulator (DA) by the value at a specified location in core, and leaves the product in DA. The calling sequence is DECIMAL with 5 in AO, the number type of the multiplier in A1*, and the core address of the multiplier in A2. Group 2.

DECIMAL STORE stores the value in the Decimal Accumulator (DA) at a specified location in core. The calling sequence is DECIMAL with 0 in AO, a number type specifying the form of storage in A1*, and the core address in A2. Group 2.

DECIMAL SUBTRACT subtracts the value at a specified location in core from the value in the Decimal Accumulator (DA), and leaves the difference in DA. The calling sequence is DECIMAL with 2 in AO, the number type of the minuend in A1*, and the core address of the minuend in A2. Group 2.

DECIMAL SUBRTACT REVERSE subtracts the value in the Decimal Accumulator from the value at a specified location in core, and leaves the difference in DA. The calling sequence is DECIMAL with 2 in AO, the number type of the subtrahend plus 10 octal in A1*, and the core address of the subtrahend in A2. Group 2.

DECIMAL LOAD loads the Decimal Accumulator (DA) from a specified location in core. The calling sequence is DECIMAL with 1 in AO, the number type of the argument in A1*, and the core address of the argument in A2. Group 2.

DECIMAL WRAPUP may be called after other decimal operations to ensure that the last store into core by the hardware Decimal Arithmetic Unit (DAU) has been completed before computation is resumed. The calling sequence is DECIMAL with octal 14 in AO. Group 2.

FIX DECIMAL TO BINARY converts a floating-point decimal number to binary form. The calling sequence is FIX without regard to the contents of any registers. Skip return is normal with the integer portion of the value in DA in A1 as an unsigned 16-bit binary integer, and the sign in AO (0 for positive or 1 for negative). Return will be non-skip if the absolute value in DA exceeds 65535 decimal. Group 2.

* (see page A1-4)

Copyright 1978
Educational Data Systems

GET SIGN OF DA may be used to determine the sign of the value in the Decimal Accumulator (DA). The calling sequence is DECIMAL with octal 13 in A0. The sign of the value in DA will be returned in bit 0 (the least significant bit) of A1, and the rest of A1 will be zero. Group 2.

FLOAT BINARY TO DECIMAL converts a signed binary integer to floating-point decimal form. The calling sequence is FLOAT with a 16-bit unsigned binary integer in A1 and the sign in A0 (0 for positive or 1 for negative). The result is placed in the Decimal Accumulator (DA). Group 2.

LOAD DAU ACCUMULATOR will load the software Decimal Accumulator (DA in REX page zero) into the hardware Decimal Arithmetic Unit (DAU). The calling sequence is LOADDA without regard to the contents of any registers. Registers A0, A1, and A2 are not changed by LOADDA. This call must be made to load the contents of DA, DAS, and DAC into the DAU if the value in any of these cells has been changed. Group 2.

SET SIGN OF DA will force the sign of the Decimal Accumulator (DA) to either state desired without changing the absolute value in DA. The calling sequence is DECIMAL with octal 12 in A0 and the desired sign in A1 (zero for "+" or one for "-"). Group 2.

SET INFINITY IN DA will set "plus infinity" in the Decimal Accumulator (DA), where "plus infinity" is the largest value that can be represented in floating point decimal (9.99999x10↑62). The calling sequence is DECIMAL with octal 17 in A0. On return, A2 will be unchanged. The error flag (ERRF) in REX page zero is set to non-zero. Group 2.

SET ONE IN DA will set the value +1 in the Decimal Accumulator (DA). The calling sequence is DECIMAL with octal 16 in A0. On return, A2 will be unchanged. Group 2.

SET ZERO IN DA will set the value 0 in the Decimal Accumulator (DA). The calling sequence is DECIMAL with octal 15 in A0. On return, A2 will be unchanged. Group 2.

STORE DAU ACCUMULATOR stores the accumulator of the hardware Decimal Arithmetic Unit (DAU) in the software Decimal Accumulator (DA in REX page zero). The calling sequence is STORDA without regard to the contents of any registers. This call must be made prior to manipulation of DA, DAS, or DAC by any processor or subroutine, whether a hardware DAU is actually installed or not. On return, A0 will contain the first word of DA, A1 will contain the value in DAS (the sign of DA), and A2 will be unchanged. Group 2.

SUBTRACT DECIMAL INTEGERS subtracts two unsigned 4-digit BCD (Binary Coded Decimal) integers. The calling sequence is LDA 3,.DEC followed by JSR @-1,3 with the subtrahend in A0 and the minuend in A1. The difference will be returned in A0, and the borrow out will be returned in bit 15 (the most significant bit) of A2. That is, if (A1) did not exceed (A0) then A0 will contain (A0)-(A1) and A2[15] will be zero; if (A1) did exceed (A0) then A0 will contain (A0)+10000-(A1) and A2[15] will be one. For multiple precision subtraction, the borrow may be propogated by incrementing the minuend in A1 (the least significant BCD digit of the minuend may be ten). Group 2.

TRANSCENDENTAL FUNCTIONS provide for evaluation of the square root, natural log, exponential, sine, cosine, tangent, and arctangent functions. The calling sequence requires three words consisting of STORDA followed by CALL Function, where "Function" is one of the following seven subroutine names:

PSQRF	(square root function)
PLOGF	(natural log function)
PEXPF	(exponential function)
PSINF	(sine function)
PCOSF	(cosine function)
PTANF	(tangent function)
PATNF	(arctangent function)

The argument of the function must be in the Decimal Accumulator (DA in REX page zero or the hardware DAU accumulator), and the result will be returned in DA. The argument for the square root function may be any positive number; if the argument is negative then the error flag (ERRF) will be set, and the square root of the absolute value will be returned in DA. The argument for the log function may be any positive number; if the argument is ≤10↑(-16) then DA will be set to -9.99999x10↑62; if the argument is ≤0 then the error flag will be set also. The exponential function is valid for arguments in the range ±148; if the argument is greater than +148 then the error flag will be set and DA will be set to +9.99999x10↑62, or if the argument is less than -148 then DA will be set to zero with no error indication. The argument for the sine, cosine, or tangent functions may be any angle expressed in radians, but the result will be most accurate if the argument is in the range zero to two pi. All argument values are valid for the arctangent function; the result will be an angle in the range ±(pi/2) radians. Group 2.

Category #3 -- Disc and File Access. These subroutines are available to a processor or task and should be used for all disc and file access on an IRIS system. Note that many are group 3, and thus are not available on all systems. Section 9.8 contains more information on many of these subroutines.

ALLOCATE allocates blocks to a random disc file. The calling sequence is CALL ALLOCATE with the number of additional blocks desired in A0, the file header block in HBA, and a pointer in A2 pointing into the header where the first disc address is to be entered. If the file is extended then A2 may point into a header extender in HXA. Skip return is normal, indicating that the blocks have been allocated. A non-skip return indicates that the allocation was not successful, and A3 may contain status 10 or 11.

ALLOCATE CONTIGUOUS is called only by Build File when creating a contiguous disc file. Group 3.

BUILD FILE creates a new file, which may replace an old file by the same Filename if it is of the same type, on the same account, and the Filename given is terminated by an exclamation mark. The calling sequence is CHANNEL BUILD with a channel number in A0, the byte address of the Filename string in A1, and a pointer to a header information table in A2. The header information table consists of six words as follows:

Word #0:	TYPE	File type word
Word #1:	NBLK	Number of blocks ($1 \leq NBLK \leq 201$ octal)
Word #2:	STAD	Starting address (stand-alone file only)
Word #3:	COST	Charge to other users in dimes (BCD)
Word #4:	UNIT	Logical Unit, or -1 to use regnant user's
Word #5:	.FMAP	Pointer to format map, or zero if none

If TYPE=37 a formatted data file will be built unless the Filename string indicated a contiguous file. The FMAP pointer may be indirect. Any protection, cost, or Logical Unit number given in the Filename string will be used in preference to those given in the table. Normal return is skip with the Real Disc Address of the new file's header in A0, the byte address of the Filename string terminator in A1, and the address of the channel in A2. The following status values are possible in A3 on a non-skip return: 0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14. See Appendix 2 for the meaning of these errors. On any non-skip return, the file has not been built.

BUILD DEVICE FILE is identical to Build File except that it will allow a Filename to begin with a dollar sign. The calling sequence is CHANNEL BILDD with all other parameters identical to BUILD above.

CHARGE charges a user for access to another account's file. The calling sequence is CALL CHARGE with the file's Logical Unit number in A0, and its Real Disc Address in A1. The Last Access Date cell in the file's header is updated, its Number of Times Accessed Counter (NTAC) is incremented, and the COST (if any) is added to the total charges (CHGS) in the header and to the "net accrued charges" cells in the regnant user's entry in the ACCOUNTS file on Logical Unit zero.

CHECK "BSA CHANGED" FLAG must be called before new information is stored in BSA. The calling sequence is CALL CBSA without regard to the contents of any registers. The data in the Block Swap Area (BSA) will be written on the disc at the disc address contained in the DBSA cells if and only if the BSA Change Flag (BSACF in REX page zero) is non-zero, and BSACF will be zeroed. DBSA is set by the system whenever a block is read into BSA, and the calling routine need only put a non-zero value in BSACF to cause the block to be written back on the disc before BSA is used for some other purpose. Core resident.

CHECK CHANNEL determines whether a channel is in use. The calling sequence is CALL CHKCHANNEL with a channel number in A0, and that channel of the regnant user is examined. Normal return is skip if the channel is in use, with the disc address of the file's header in A1 and the address of the channel in A2. Non-skip return if channel not in use, with possible status values being 0 or 1. Core resident.

CHECK COPY PROTECTION determines whether a file is copy protected. The calling sequence is CALL CHKCP with the file's ACNT word in A0 and its TYPE word in A1. Access will be granted (indicated by a skip return) if the file's account number is the same as the regnant user's account number, if the regnant user is privilege level three or has a higher privilege level than the file, or if the protection specified in the file does not include copy protection. Non-skip return if access not granted. Core resident.

CHECK READ PROTECTION determines whether a file is read protected. The calling sequence is CALL CHKRP, but it is otherwise the same as Check Copy Protection. Core resident.

CHECK WRITE PROTECTION determines whether a file is write protected. The calling sequence is CALL CHKWP, but it is otherwise the same as Check Copy Protection. Core resident.

CLEAR ALL CHANNELS clears all data channels of the regnant user's port. The calling sequence is CALL ALLCLEAR without regard to the contents of any register. Clear Channel is used to clear each data channel on the regnant user's port. Return is non-skip, and there are no error indications. Core resident.

CLEAR CHANNEL clears a specified data channel. The calling sequence is CHANNEL CLEAR with a channel number in A0. This function is identical to Close Channel unless a file was being built on the channel and has never been closed; in this case, the new file will be deallocated, and any old file that it may have been replacing will be restored to normal status.

CLOSE CHANNEL closes a specified data channel. The calling sequence is CHANNEL CLOSE with a channel number in A0. If a file was being built on the specified channel then it will be made accessible to other users, and any file being replaced by the new file will be deleted at this time unless it is open elsewhere. Normal return is skip; non-skip returns are possible with error status 0 or 2.

CONVERT LOGICAL TO REAL ADDRESS converts a logical disc address to a Real Disc Address. The calling sequence is CALL CLRA with a LUFIX pointer in A0 and a pointer to a logical address table in A2. The logical disc address must consist of a logical cylinder number at (A2), a logical track number at (A2)+1, and a logical sector number at (A2)+2. The conversion factors in the LUFIX are used to combine the logical values into the corresponding Real Disc Address which is returned in A1. The logical address table is unchanged. Core resident.

CONVERT REAL TO LOGICAL ADDRESS converts a Real Disc Address to a logical disc address. The calling sequence is CALL CRLA with a LUFIX pointer in A0, a Real Disc Address in A1, and a pointer to a logical disc address table in A2. The conversion factors in the LUFIX are used to separate the given Real Disc Address into its logical components, which will be stored in the logical address table as described for input to Convert Logical To Real Address. On return, A1 contains the logical sector number, A2 is unchanged, and A3 contains the logical track number. No checking is done for an illegal address. Core resident.

DEALLOCATE deallocates blocks from a random disc file. The calling sequence is CALL DALLC with a file header in HBA, the desired number of blocks to be in the file in A0, and a pointer into the header in A2. Disc blocks will be deallocated starting at (A2) and working back toward the beginning of the file until (A0) blocks remain, and the file's owner is credited for the freed blocks. (A2) is ignored if (A0) is zero.

DELETE KEY is used only by the Search subroutine for the purpose of removing a key from a directory.

DELETE FILE deletes any file other than a processor, a driver, or a permanent system file. The calling sequence is CALL DELETE with a Logical Unit number in AO and a byte pointer to a Filename string in A1. If (AO) is -1 then the regnant user's Logical Unit will be assumed. The Filename string may not be in HBA. The Filename is immediately removed from the specified INDEX, but the file will not be deallocated until it is not open on any user's channel. Normal return is skip with the byte address of the Filename terminator in A1, and AO will be non-zero if the file is being replaced. Possible non-skip error status values are 7, 13, 15, 17, 22, and 24 (see Appendix 2).

DELETE PROCESSOR deletes any file except a permanent system file. The calling sequence is CALL PDELETE with registers as described for Delete. The operation is identical to Delete, except that a processor or driver file may also be deleted.

DIRECTORY is used to set up an indexed file and to maintain the free record list for the file. Used only by Business BASIC for the mode zero functions of the SEARCH statement. Group 3.

EXTEND FILE increases a random file's size to greater than 128 data blocks. The calling sequence is CALL EXTEND with a non-extended random file's header in HBA. Normal return is skip with the modified header in HBA, the header extender block in HXA, the file's Logical Unit number in AO, the Real Disc Address of the extension block in A1, and the address of HXA in A2. The calling routine must write the new extender and the modified header to the disc. Error status values 10 and 11 are possible on a non-skip return.

FIND FILE searches the INDEX of a specified Logical Unit for a given Filename. The calling sequence is CALL FFILE with a Logical Unit number in AO and a byte pointer to the Filename string in A1. If the Filename string is supplied in the form number/Filename then (AO) is ignored and only the Logical Unit specified by that number will be searched; otherwise, only the Logical Unit specified by (AO) will be searched. Return is skip if the Filename is found in the INDEX of the specified Logical Unit, or non-skip if the Filename is not found. Error status values 13, 14, and 16 are possible on a non-skip return, but a larger value in A3 means that the Filename simply was not found. On such a return (or on a skip return) the INDEX block containing the Filename (or entry for a new Filename) will be in BSA, AO will contain the Logical Unit number, A2 will contain the Real Disc Address of the INDEX block in BSA, and A3 will contain a pointer into BSA to the actual INDEX entry.

FIND LOGICAL UNIT TABLES locates the Logical Unit Table entry and the fixed and variable information tables (LUFIX and LUVAR, respectively) for a given Logical Unit number. The calling sequence is FINDLUT with a Logical Unit number in AO. Return is non-skip with 14 in A3 if the Logical Unit is not active; otherwise, skip return with the address of the Logical Unit Table entry in A1, a pointer to the LUFIX table in A2, and a pointer to the LUVAR table in A3. Register AO is unchanged in either case.

FIND OPEN FILE scans all channels on all ports to determine whether another user has a designated file or any file on a designated Logical Unit open. The calling sequence is a two-step process. First CALL FOFI with a Logical Unit number in A0 and the Real Disc Address of the file in question in A1. Return is non-skip with a pointer to FOFC in A3. Now CALL FOFC (or JSR 0,3 if A3 has not been changed) to look for a channel where the file is open. Return is non-skip if no such channel is found. If the file is found to be open anywhere, then return is skip with the port's PCB address in A0, the address of the channel less CHM4 in A2, and the FOFC entry address still in A3. If A1 was zero when FOFI was called, then FOFC will look for any file open on the specified Logical Unit. Core resident.

FLUSH BUFFER POOL assures that there are no dirty pages in the disc block buffer pool. The calling sequence is LDA 2,.FALT followed by JSR @-3,2 without regard to the previous contents of any register. FLUSH scans the entire buffer pool table, writes each dirty page to its respective disc address, and clears the dirty page flag. FLUSH is used by FAULT, SHUTDOWN, and REMOVE.

FIX DIRECTORIES adjusts the Real Disc Addresses in the directories of an indexed file after the file has been moved. Used only by COPY, CLEANUP, and $MTA. Group 3.

GET RECORD FOR READ locates a designated record in a file and brings the data block into core. Used only by Read Item and Read Contiguous.

GET RECORD FOR WRITE is identical to Get Record For Read, except that it will error return if the file is write protected. Used only by Write Item, Write New Item, and Write Contiguous.

Copyright 1978
Educational Data Systems

OPEN opens a file or a device on a channel. OPEN FOR UPDATE opens a file (but not a device) on a channel with the intent of writing or updating data. OPEN FOR REFERENCE opens a file or a device for reference only; writing will not be allowed. OPEN AND LOCK opens a file or a device and locks out all other users. For OPEN the calling sequence is CHANNEL OPEN with a channel number in A0, the byte address of a Filename in A1, and a pointer to a channel control block in A2. The channel control block consists of two words as follows:

Word #0: expected file type, or -1 for any type
Word #1: Logical Unit number, or -1 if the regnant user's assigned Logical Unit is to be used

If the desired file is found, is not read protected, and is of the expected type, then it is opened on the specified channel. A default file type 37 in word #0 will allow opening any file of type 30 through 36. Opening a type 36 file (a device driver) causes a JSR to the driver's INIT routine. If the file is write protected then the write locked status is set in the channel. CHARGE is called by OPEN to charge the user for access to the file. If a Logical Unit number is given in the Filename string then this overrides the unit number given in CCB word #1. Skip return is normal with the Real Disc Address of the file's header in A0, the byte address of the Filename terminator in A1, and the core address of the channel in A2. The following status values are possible in A3 on a non-skip return: 0, 1, 13, 14, 15, 17, 20, 21, 25, 34, 35, 36. See Appendix 2 for the meanings of the errors. On any non-skip return, the file has not been opened.

The calling sequence for the other modes of OPEN are the same as above except substitute the word OPENUPDATE, OPENREF, or OPENLOCK for the word OPEN following the word CHANNEL. All other aspects are the same except for the following:

OPENUPDATE will return an error 22 if the file is write protected. Also, OPENUPDATE will open a driver file itself rather than the device interfaced by the driver.

OPENREF ignores the read protection, does not call CHARGE, and does not JSR to a driver's INIT routine. The write locked bit is unconditionally set in the channel status word without regard to the file's protection.

OPENLOCK will return an error 22 if the file is write protected or will return error 25 if the file is open elsewhere.

READ CONTIGUOUS reads data from a contiguous disc file or a text file. The calling sequence is identical to that of Read Item except that a byte displacement into the record must be given in place of the item number. For a text file, the "record length" is assumed to be 256 words (512 bytes), and record number -2 functions the same as record number -1 for sequential access except that reading will not be terminated by a RETURN code but will attempt to fill the entire destination area. Group 3.

READ DISC BLOCK reads one block (256 words) from a Logical Unit into core. The calling sequence is READBLOCK with a Logical Unit number in AO, a Real Disc Address in A1, and a core address in A2. Return is non-skip with registers AO, A1, and A2 unchanged after the block has been read into core at the specified location. A trap will occur in the case of an inactive Logical Unit or an illegal disc or core address. Core resident.

READ FILE HEADER INFORMATION is used only by the CALL 97 statement in Business BASIC. Group 2.

READ ITEM reads an item from a file or from a peripheral device. The calling sequence is CHANNEL READITEM with a channel number in AO and a pointer to an Item Control Block (ICB) in A2. The ICB has the form

Word #0:	record number (origin zero)
Word #1:	item number (origin zero)
Word #2:	item type (see Sèction 1.5)
Word #3:	desired length (number of Words or bytes)
word #4:	destination pointer (byte address if string)

One item is accessed from the file or peripheral device open on the specified channel, and the item is stored at the location specified in word #4 of the ICB. The amount of data transferred will be the smaller of the specified destination size in word #3 or the data supplied by the file or the device. If the item length is smaller than the destination area then a non-string item will be padded with zeroes or a string item will be terminated with a single zero byte. If reading a string item then the byte address given in word #4 of the ICB must be relative to (BBA). Return is skip if successful with the number of words or bytes actually transferred in AO, and A3 will be zero if and only if a text file is being read. Return is non-skip with error status in A3 if not successful.

RELEASE BLOCK is used only by the SEARCH subroutine to release a block of a directory back to the free chain. Group 3.

SEARCH is used to search a directory of an indexed file or to insert or delete a key entry in such a directory. Used only by Business BASIC for the non-zero modes of the SEARCH statement. Group 3.

SHUFFLE is used only by the Search subroutine to reorganize a directory when an insertion overflows a block of a directory. Group 3.

UNLOCK RECORD unlocks a record that has been locked by a file access. The calling sequence is CALL UNLOCK with a channel number in A0. Normal return is skip, having unlocked any record which may have been locked on the specified channel. Non-skip return is possible if the channel number is illegal (A3=0) or if the channel is not open (A3=2). Core resident.

WRITE CONTIGUOUS writes data into a contiguous disc file or a text file. The calling sequence is identical to that of Write Item below except that a byte displacement into the record must be given in place of the item number. For a text file, the "record length" is assumed to be 256 words (512 bytes), and record number -2 functions the same as record number -1 for sequential access. Group 3.

WRITE DISC BLOCK writes one block (256 words) from core to a Logical Unit. The calling sequence is WRITBLOCK with the registers as specified for Read Disc Block above. Return is non-skip with registers A0, A1, and A2 unchanged after the specified block has been written to the disc. A trap will occur in the case of an inactive Logical Unit or an illegal disc or core address. Core resident.

WRITE ITEM writes an item into a file or to a peripheral device. The calling sequence is CHANNEL WRITITEM with the registers and ICB identical to that described for READITEM opposite. The number of words or bytes transferred will the the smaller of the source length or the file item size. If the source length is smaller, then a non-string will be padded with zeroes or a string will be terminated with one zero byte. Rteurns are the same as for READITEM except that the contents of A0 are meaningless.

WRITE NEW ITEM is used only by Write Item when automatically formatting a new formatted disc file by writing sequentially into the first record. Group 3.

Copyright 1978
Educational Data Systems

Category #4 -- Swapping and Tasking. These subroutines are available for use by processors and tasks. All swapping and tasking on an IRIS system must be done by use of these system calls. Most subroutines in this category are group 1 and are thus available on all IRIS systems.

ACTIVE FILE SETUP is used by a processor's swap-out subroutine to set up the active file header for the areas of core that are to be swapped out. See Section 9.5 for calling sequence.

BUMP USER bumps the regnant user from core. The calling sequence is BUMPUSER without regard to the contents of any registers. The regnant user will be bumped unconditionally, and control will later be transferred to the next location following the BUMPUSER instruction when another time slice is granted. An alternative calling sequence allows control to be transferred to any address x at the next time slice; it is LDA 3,x followed by JMP @.BUMP. The processor should do a bump whenever its time slice is used up; specifically, it should periodically execute the instruction sequence

```
        LDA     0,RTL
        NEGL#   0,0,SNC
        BUMPUSER
```

DEQUEUE TASK removes a task from the priority task queue. See Section 8.12 for calling sequence.

GET OR RELEASE A FREE NODE acquires a node from the free node chain or releases a node no longer needed back to the chain. Each node is ten words decimal. The callinq sequence is FREENODE with zero in A2 if a node is desired or with a pointer to the first word of the node if a node is being released. When acquiring a node, the node pointer is returned in A2. A1 is unchanged in any case, and return is non-skip. Core resident.

LINK PROGRAMS is used only by BASIC's CHAIN statement.

LOAD USER loads the regnant user's active file into core if its type matches the user's processor. See Section 9.4 for calling sequence. Core resident.

Copyright 1978
Educational Data Systems

PAUSE bumps the regnant user for a specified time duration or (optionally) until a signal is sent to the user in the pause state. The calling sequence is

```
        CALL
        SIGPAUSE
        BUMPUSER
```

with a delay value (in tenth-seconds) in A1. A0 must contain 3 for an unconditional pause or 4 if it is desired to terminate the pause if a signal is sent to this port. Group 2.

QUEUE CHARACTER (Category #1) may also be used to queue an interrupt task. See Section 11.6 for calling sequence.

QUEUE TASK puts a task on the priority task queue. See Section 8.11 for calling sequence.

REQUEUE is used only by the system to re-order the task queue as a result of dynamic priorities.

SCOPE exits from a processor. The calling sequence is CALL SCOPE without regard to the contents of any registers. Control is not returned to the processor except for one last JSR to its swap-out subroutine. Core resident.

START INPUT will cause a user to be bumped. See Category #1.

WAIT FOR OUTPUT NOT ACTIVE may cause a user to be bumped. See Category #1.

Copyright 1978
Educational Data Systems

Category #5 -- Miscellaneous. These subroutines are available for use by all modules added to an IRIS system. This is merely a catch-all catagory for subroutines that do not fit in any of the four previous catagories. Most are in group 1, and are thus available on all IRIS systems.

ACCOUNT LOOKUP finds a user's account entry in an ACCOUNTS file via the Account I.D., account number, or entry position (record number). The calling sequence is CALL ACNTLOOKUP with a mode designator in A0 and a pointer to a 256-word buffer area (may be BSA) in A2. A0 prescribes the mode as follows:

If (A0)=0 then (A1) = account entry position

If (A0)=-1 then (A1) = B(Account ID string)

If (A0)=1 then (A1) = account number (bits 0-13)

If (A0)>1 then (A1) = account number, and HBA must contain any file header; the UNIT cell of that header indicates the Logical Unit on which to look up the account number.

Return is non-skip if the account is not found. A skip return gives the account entry position in A0, the disc address of the ACCOUNTS block in A1, and a pointer to the account entry (in the specified buffer area) in A2, and status 40 (octal) in A3.

BREAKPOINT is used only by the system when a DSP breakpoint is encountered in a processor.

COMPARE STRINGS tests whether two strings are equivalent. The calling sequence is CALL CSTR with the absolute byte addresses of the strings to be compared in A0 and A1. Return is non-skip with the terminating byte addresses in A0 and A1, and an equivalance indicator in A3 (zero if equal or non-zero if not equal).

Copyright 1978
Educational Data Systems

CONVERT ASCII TO DATE scans an ASCII string, which must represent a date and time in standard format, and converts it to two binary words representing "hours since 1-1-76" and "part of hour in tenth-seconds". The calling sequence is CALL CNVAD with the byte address of the string in A1. The string may represent the date in either of two forms as follows:

```
    FEB 20, 1977  11:09:56
        or
    77,2,20,11,09,56
```

where the second string is the same form as requested when an IPL is performed. In either case, the first two digits of the year, the "seconds" value, and all leading zeroes are optional. Spaces, commas, and colons are interchangable as field separators, and the string may be terminated with a zero byte, a RETURN code, or any other character that is not acceptable as part of the date/time representation. Normal return is skip with the hours after 1-1-76 in A0, the part of hour in A1, the terminating code in A2, and the byte address of the next byte after the terminator in A3. Return is non-skip if an error is detected in the string, with the registers containing any partial results as described above.

CONVERT DATE TO ASCII converts the two word binary date into an ASCII string in standard format. The calling sequence is CALL CNVDA with a pointer to the pair of clock words (hours after 1-1-76, and part of hour in tenth-seconds) in A2, or zero in A2 if the system clock is to be used. A1 must contain the byte address where the string is to be stored, or zero if it is to be output to the regnant user's output buffer. If A1 is non-zero then A0 must contain the destination string size (not less than 27 octal). The string will be in the first form shown for CNVAD above. Return is skip if successful, or non-skip if an illegal time value is given (second word exceeds 35999 decimal).

CONVERT DATE & TIME is used only by the CALL 99 statement in Business BASIC. This subroutine uses Convert Date To ASCII and Convert ASCII To Date.

CONVERT PCB POINTER TO PORT NUMBER determines the number of a port from the location of its Port Control Block. The calling sequence is CALL CPPPN with a PCB pointer in A0. Return is non-skip if (A0) is not a valid PCB pointer; otherwise return is skip with the port number in both A0 and A1. Core resident.

CONVERT PORT NUMBER TO PCB POINTER determines the location of the Port Control Block for a given port number. The calling sequence is CALL CPNPP with a port number in A0. Return is non-skip if (A0) is not a valid port number; otherwise skip return with the port's PCB pointer in A0. Core resident.

DECIMAL INPUT CONVERSION is identical to Decimal Input described in Category #1 except that a byte address is given in A1 when calling the subroutine, and the ASCII string at that byte address is scanned rather than using Access Input Byte. Returns are also the same, except that A1 will contain the byte address of the first character not converted. If (BBA) is zero then (A1) must be an absolute byte address, but if (BBA) is non-zero then (A1) must be a byte displacement from (BBA). Group 2.

DECIMAL OUTPUT CONVERSION is identical to either Decimal Output or Formatted Decimal Output described in Category #1, except that a pointer to a two-word control table is given in A1 when the subroutine is called. The first word of the table is a starting byte address where the output string is to be stored (instead of using Store Output Byte). The second word of the table is the maximum number of bytes that may be output to that location. In the non-formatted case, if the byte count is exceeded then control returns to the caller with the output incomplete. In the formatted case, if the byte count is exceeded then a non-skip return is given with the output incomplete. In any case, on return, A1 will contain the next byte address after the last byte stored. If (BBA) is zero then (A1) must be an absolute byte address, but if (BBA) is non-zero then (A1) must be a byte displacement from (BBA). Group 2.

FLAGCHANGE allows changing and testing individual bits of a flag word in core. The calling sequence is

```
FLAGCHANGE
command+displacement+skip
mask
```

with a table base pointer in register A2. The word "command" must be replaced by either SET or RESET or TOGGLE indicating that the selected bits are to be set to ones, reset to zeroes, or complemented, respectively. The word "displacement" is either a number or a symbol equated to a number giving the displacement from the table pointer in A2 to the flag word. The word "skip" is either SKIPZ or SKIPO or is omitted, indicating that a skip is desired if all of the selected bits are zero or if any of the selected bits are one, respectively, as a result of the change, or that no skip is desired. The "mask" word contains ones in the bit positions of interest in the flag word. Note: for toggle, only one bit in the mask word may be a one.

GET BYTE accesses one byte from a given location in core. The calling sequence is GETBYTE with any absolute byte address in register A1. The byte is returned in register A2, and the top half of A2 will be zero. Register A1 is unchanged. See also XGETBYTE on page A1-22.

Copyright 1978
Educational Data Systems

IS (A2) A DIGIT? determines whether register A2 contains an ASCII code for a decimal digit. The calling sequence is ISA2DIGIT with an ASCII code in A2. Return will be non-skip if the ASCII code does not represent a decimal digit (not in the range 260 to 271 octal). Return is skip if it is a digit, and A1 will contain the ASCII code for a zero, so that a SUB 1,2 instruction will convert the ASCII code to its binary value. A2 is unchanged in any case. A0 is unused and unchanged.

IS (A2) A LETTER? determines whether register A2 contains an ASCII code for a letter. The calling sequence is ISA2LETTER with an ASCII code in A2. If the character is neither an upper case nor a lower case letter then return is non-skip with A2 unchanged. If the code is a letter then it is converted to a number; i.e. 1 for either "A" or "a", 2 for "B" or "b", etc., and return is skip. After a skip return, an ADD 1,2 instruction will restore the original ASCII code in A2.

MOVE WORDS moves the contents of a group of words in core to another area in core. The calling sequence is CALL MOVEWORDS with the first address of the source area in A0, the last address of the source area in A1, and the first address of the destination area in A2. Return is non-skip. Core resident.

MOVE BYTES moves a group of bytes in core to another area in core.

The calling sequence is CALL MOVBYTES followed by a mode word containing relative address flags in the top two bits (see below) and an alternate terminating code in the lower 8 bits. A0 is the byte address of the beginning of the source area, A1 is the byte address of the end of the source area, and A2 is the byte address of the beginning of the destination area. Bytes are moved from the source to the destination area until either a) the source is exhausted, b) a zero byte is transferred, or c) a byte equal to the alternate terminator code given in bits 0-7 of the mode word is transferred. Return is non-skip with the source byte address of the last byte transferred in A0, the number of bytes not transferred in A1, and the last byte transferred in A2. If bit 15 of the mode word is one then the source byte addresses (in A0 and A1) are relative to the Byte Base Address in BBA; if bit 14 of the mode word is one then the destination byte address (in A2) is relative to (BBA). If (BBA)=0 then all addresses are taken as absolute regardless of these flag bits. Core resident.

PASSWORD COMPARE tests whether the user supplied the correct password. The calling sequence is CALL PASSCOMPARE with the byte address of the correct password in A1. Access String Byte is used to get characters from the regnant user's input buffer for comparison with the string given. If the password entered by the user ends with a RETURN or CTRL E code at the same time that the given string ends with a zero byte then the return will be skip with the terminating character in A2. Return is non-skip with a CTRL E code in A0 if no password was entered, or with the first mismatched byte of the correct password in A0 if an incorrect password was entered.

PUT BYTE stores one byte at a given byte address in core. The calling sequence is PUTBYTE with any absolute byte address in register A1 and the byte to be stored in the lower half of register A0. See also XPUTBYTE on page A1-22. Core-resident.

RECEIVE SIGNAL receives a signal if any has been sent to the regnant user's port. The calling sequence is CALL SIGPAUSE with 2 in A0 and a pointer to a parameter list in A2. Return is non-skip if no signal was waiting for the regnant user, and the parameter list is unchanged. Otherwise, return is skip with the port number of the sender in word #0 of the parameter list and the two signal values in words 1 and 2 of the parameter list. Group 2.

RECOVER is used only by the system to start the input/output system after initialization or after a fault.

SEND SIGNAL sends a signal to a user on another port or to a later program segment on the same port. The calling sequence is CALL SIGPAUSE with 1 in A0, the sender's PCB pointer in A1, and a pointer to a parameter list in A2. The parameter list must contain either the port number or the PCB pointer of the addressee in word #0 and the two signal values in words 1 and 2. Return is non-skip if the signal buffer is full and the signal cannot be sent; otherwise return is skip. Group 2.

SPECIAL FUNCTIONS will access certain parameters such as system time, port number, amount of time used, etc. Used only by the SPC function in Business BASIC. Group 2.

START IPL aborts all system operations and performs an Initial Program Load. The calling sequence is JMP @.STPL without regard to the contents of any registers. There is no return.

SYSTEM COMMAND TRANSMITTER is used only by the CALL 98 statement in Business BASIC. Group 2.

TRAPFAULT aborts a process due to an illegal condition or a hardware failure and prints a trap message. The calling sequence is TRAPFAULT without regard to the contents of any registers. The TRAPFAULT may be followed by a value of the form n*K!NOP where n is an octal number not exceeding 177. The regnant user's task will be aborted, and a message of the form

    TRAP #n AT location IN processor

will be printed on the user's terminal, where n is the number supplied in the n*K!NOP word (zero if no such word is supplied), "location" is the location of the TRAPFAULT instruction (in octal), and "processor" is the Filename of the ragnant user's current processor. This will be followed by a second line giving the STATUS, which consists of the contents of registers A0, A1, A2, A3, and the carry bit. If the fault was in a discsub, then a third line will be printed giving the address in the DISCSUBS listing (several addresses if discsubs were nested). If the fault was in a core-resident subroutine called by a discsub, then the absolute address is given preceeded by an asterisk. If 200 octal is added to the n*K!NOP expression (ie, it is given as n*K!NOP!200) then the regnant user's active file will be cleared; this should be done in any case where it is likely that the fault condition was caused by garbage in the active file.

XGETBYTE (eXtended GET BYTE) is the same as GETBYTE (see page A1-20) except that the byte address given in A1 is taken to be relative to the Byte Base Address (the word address in BBA). If (BBA)=0 then XGETBYTE is identical to GETBYTE. Core-resident.

XPUTBYTE (eXtended PUT BYTE) is the same as PUTBYTE (see page A1-22) except that the byte address given in A1 is taken to be relative to the Byte Base Address (the word address in BBA). If (BBA)=0 then XPUTBYTE is identical to PUTBYTE. Core-resident.

Appendix 2: CANNED MESSAGES

The MESSAGES file contains a variety of "canned" messages that can be transferred to the regnant user's output buffer by use of the MESSAGE subroutine. The calling sequence is CALL MESSAGE with the message number (see list below) in register A1. The message will be preceeded by a carriage return if (A0)=-1, or by a carriage return, a question mark, and a space if (A0)=-2. If A0 contains any other value, then nothing preceeds the string from the MESSAGES file. There are two possible returns from the MESSAGE subroutine as follows:

> Non-skip if MESSAGES file has not been loaded on the system disc (indicated by A3=0), or if there is no message assigned the number given (indicated by A3<>0). In the former case the string `?? NO "MESSAGES" FILE' has been output to the regnant user's output buffer, and in the later case the string `?? NO SUCH MESSAGE NUMBER' has been output.
>
> Skip if the desired message has been transferred to the regnant user's output buffer.

The message may be appended to any previous output and may be followed by more output by use of OUTBYTE, OUTTEXT, CIA, etc., or the call to MESSAGE may be followed immediately be a Start Output.

Output must not be active when MESSAGE is called. Note that the messages are numbered in decimal, but the octal equivalent of the number must be given to MESSAGE in A1 to output the desired message. Messages one through 99 coincide with the respective BASIC error numbers, and messages 100 and up coincide with the error status on a non-skip return from most DISCSUBS if decimal 100 (octal 144) is added to the status returned in register A3. The status values shown on page A2-4 are in octal. The currently available messages are listed on the following pages.

Copyright 1978
Educational Data Systems

Appendix 2: CANNED MESSAGES (continued)

No.	Message
0.	?? NO SUCH MESSAGE NUMBER
1.	SYNTAX ERROR
2.	ILLEGAL STRING OPERATION
3.	STORAGE OVERFLOW (PROGRAM TOO LARGE)
4.	FORMAT ERROR
5.	ILLEGAL CHARACTER
6.	NO SUCH LINE NUMBER
7.	RENUMBER ABORTED BY ESCAPE, PROGRAM WAS LOST
8.	TOO MANY VARIABLE NAMES (LIMIT IS 93)
9.	UNRECOGNIZABLE WORD
10.	LINE NUMBER,"RUN" IS ILLEGAL BEFORE AN INITIAL RUN
11.	INCORRECT PARENTHESES CLOSURE
12.	PROGRAM IS LIST/COPY PROTECTED
13.	NUMBER TOO LARGE (9.9999999999999E+62 IS MAXIMUM)
14.	OUT OF DATA
15.	ARITHMETIC OVERFLOW (SUCH AS DIVISION BY ZERO)
16.	"GOSUB"S NESTED TOO DEEP
17.	"RETURN" WITHOUT "GOSUB"
18.	FOR-NEXT LOOPS NESTED TOO DEEP
19.	"FOR" WITHOUT MATCHING "NEXT"
20.	"NEXT" WITHOUT MATCHING "FOR"
21.	EXPRESSION TOO COMPLEX (TOO MUCH FUNCTION NESTING)
22.	NOT ENOUGH DISC BLOCKS AVAILABLE FOR SWAP-OUT
23.	ARRAY SIZE EXCEEDS INITIAL DIMENSIONS
24.	ONLY ONE DIMENSION ALLOWED FOR A STRING
25.	STRING OR ARRAY NOT DIMENSIONED
26.	LOGICAL UNIT NOT ACTIVE
27.	SYNTAX ERROR IN USER-DEFINED FUNCTION
28.	SUBSCRIPT, CHANNEL NUMBER, OR SIGNAL PARAMETER OUT OF RANGE
29.	ILLEGAL FUNCTION USAGE
30.	USER FUNCTION NOT DEFINED
31.	USER FUNCTIONS NESTED TOO DEEP
32.	MATRICES HAVE DIFFERENT DIMENSIONS
33.	ARGUMENT IS NOT A MATRIX
34.	DIMENSIONS ARE NOT COMPATIBLE
35.	MATRIX IS NOT "SQUARE"
36.	CALLED SUBROUTINE NOT IN STORAGE
37.	EXPRESSION IN ARGUMENT FOR CALL
38.	ERROR DETECTED BY CALLED SUBROUTINE
39.	FORMATTED OUTPUT EXCEEDED BUFFER SIZE
40.	CHANNEL IN USE
41.	ILLEGAL FILENAME
42.	FILE NOT FOUND
43.	SYNTAX ERROR IN COST OR PROTECTION
44.	NOT A DATA FILE (CAN'T OPEN OR REPLACE)
45.	FILE IS READ PROTECTED
46.	FILE IS WRITE PROTECTED
47.	LOGICAL UNIT DOES NOT HAVE ENOUGH FREE BLOCKS
48.	ACCOUNT'S DISC BLOCK ALLOTMENT IS INSUFFICIENT
49.	CHANNEL NOT OPEN

(continued on next page)

Appendix 2: CANNED MESSAGES (continued)

No.	Message
50.	FILE IS COPY PROTECTED
51.	ILLEGAL RECORD NUMBER
52.	RECORD NOT WRITTEN
53.	ILLEGAL ITEM NUMBER
54.	ITEM TYPES DON'T MATCH
55.	STATEMENT IS ILLEGAL FROM KEYBOARD
56.	CAN'T DUMP AN EMPTY PROGRAM
57.	STRINGS CANNOT BE REDIMENSIONED
58.	ERROR IN FORMAT STRING
59.	"RUNMAT" PROCESSOR NOT IN SYSTEM
60.	TOO MANY NUMBERS ENTERED FOR INPUT
61.	MATRICES HAVE DIFFERENT NUMBER TYPES
62.	SIGNAL BUFFER IS FULL OR NO SUCH PORT
63.	COMMANDS ARE ILLEGAL IN "LOAD" MODE
64.	LINE NUMBER MISSING IN "LOAD" MODE
65.	FILENAME IN USE FOR DIFFERENT TYPE FILE
66.	FILENAME IN USE, OLD FILE BEING BUILT OR REPLACED
67.	FILENAME IN USE AND NO "!" SUPPLIED
68.	FILENAME IN USE BY A DIFFERENT ACCOUNT
69.	FILE IS A PROCESSOR OR DRIVER
70.	DATA READ ERROR
71.	IPL NEEDED TO INITIALIZE DRIVER
72.	DEVICE NOT ACCESSIBLE
73.	DEVICE NOT ON LINE
74.	DEVICE REQUIRES MANUAL INTERVENTION
75.	LINE EXCEEDS BUFFER SIZE IN "LOAD" MODE
76.	FILE OR DEVICE IS OPEN ELSEWHERE
77.	NO ACCOUNT ON SELECTED LOGICAL UNIT
78.	FILE IS BEING BUILT, REPLACED, OR DELETED
79.	ILLEGAL DEVICE OPERATION
80.	LOGICAL UNIT DOES NOT HAVE ENOUGH CONTIGUOUS BLOCKS
97.	NO $DEC OR $DAU ON SYSTEM
98.	ILLEGAL VALUE ENTERED FOR INPUT
99.	ESC OR CTRL C TRAPPED BY ERROR BRANCH

(continued on next page)

No.	Status	Message
100.	0	ILLEGAL CHANNEL NUMBER
101.	1	CHANNEL IN USE
102.	2	CHANNEL NOT OPEN
103.	3	FILENAME IN USE FOR DIFFERENT TYPE FILE
104.	4	FILENAME IN USE, OLD FILE BEING BUILT OR REPLACED
105.	5	FILENAME IN USE AND NO "!" SUPPLIED
106.	6	FILENAME IN USE BY A DIFFERENT ACCOUNT
107.	7	FILENAME IN USE FOR A PERMANENT FILE
108.	10	LOGICAL UNIT DOES NOT HAVE ENOUGH FREE BLOCKS
109.	11	ACCOUNT'S DISC BLOCK ALLOTMENT IS INSUFFICIENT
110.	12	SYNTAX ERROR IN COST, PROTECTION, OR SIZE
111.	13	ILLEGAL FILENAME
112.	14	LOGICAL UNIT NOT ACTIVE
113.	15	FILE NOT FOUND
114.	16	FILE NOT FOUND, AND NO INDEX ENTRY FOR NEW FILE
115.	17	FILE IS BEING BUILT, REPLACED, OR DELETED
116.	20	INCORRECT FILE TYPE
117.	21	FILE IS READ PROTECTED
118.	22	FILE IS WRITE PROTECTED
119.	23	FILE IS COPY PROTECTED
120.	24	FILE IS A PROCESSOR OR DRIVER
121.	25	FILE OR DEVICE IS OPEN ELSEWHERE
122.	26	ILLEGAL RECORD NUMBER
123.	27	RECORD IS LOCKED
124.	30	ILLEGAL ITEM NUMBER
125.	31	ITEM TYPES DON'T MATCH
126.	32	RECORD NOT WRITTEN
127.	33	DATA READ ERROR
128.	34	IPL NEEDED TO INITIALIZE DRIVER
129.	35	DEVICE NOT ACCESSIBLE
130.	36	DEVICE NOT ON LINE
131.	37	DEVICE REQUIRES MANUAL INTERVENTION
132.	40	NO ACCOUNT ON SELECTED LOGICAL UNIT
133.	41	ILLEGAL DEVICE OPERATION
134.	42	LOGICAL UNIT DOES NOT HAVE ENOUGH CONTIGUOUS BLOCKS
135.	43	DEVICE ATTRIBUTES NOT SET UP PROPERLY

Appendix 3: GLOSSARY

IRIS - Interactive Real-time Information System. This is the name of the system as a whole, including REX, the accounting and security system, the INDEX, DISCSUBS, DMAP, CONFIG, and ACCOUNTS files, and the system command processors.

REX - Real-time EXecutive. REX consists primarily of software which resides in approximately the first 4000 core locations. Page zero contains system flags and counters, many widely used constants, and pointers to subroutines in REX that may be used by processors and drivers.

ROUTINE - Any sequence of instructions or program statements is called a routine.

PROCESSOR - A machine language routine which operates under control of the processor task and which may be called for execution directly from any interactive terminal. Examples of processors are BASIC, SAVE, KILL, LIBR, CHANGE, and QUERY.

STAND-ALONE PROGRAM - A machine language routine which takes over full control of the system and does not operate under REX. Stand-alone programs are loaded into core by use of the SHUTDOWN command. Examples of stand-alone programs are GPM and the computer diagnostic routines.

PROGRAM - A higher-level language routine which is executed by being either interpreted or compiled by a processor. For example, any routine written in BASIC is called a program. Since a program is not in machine code it is treated by REX as a data file to be operated on by a processor.

SUBROUTINE - A subroutine is any routine that is written as a separate module (even though it may be included with the main routine or other subroutines) and is used by a GOSUB statement, a JSR (Jump to Subroutine) instruction, or a system CALL. A subroutine may be part of REX, part of a processor, part of a stand-alone program, or part of a higher level language program.

DISCSUB - A discsub is a machine language subroutine that resides in the DISCSUBS file as part of IRIS or an extension to REX. Selected disc-resident subroutines may be made core-resident at IPL time (see Section 2.9).

RE-ENTRANT - A re-entrant routine (usually a subroutine) is one that may be interrupted at any time and re-entered to perform a higher priority task, then will complete the first task correctly when control is returned from the interrupt. A stack is usually used for the routine's temporary storage, including the return address, to implement this capability.

RECURSIVE - A recursive routine is a re-entrant routine that calls itself. See recursive.

BIT - One binary digit. May contain a one or a zero.

BYTE - Eight bits. One byte may contain an ASCII code or a special internal code.

WORD - Sixteen bits. The computer transfers information one word at a time. One word may contain one machine instruction, one integer number, or two bytes. Two or more words are required to store a floating point number.

STRING - A sequence of ASCII codes stored one code per byte with the top bit of each byte set to one and the first byte of each pair in the left half (most significant byte) of the word. A string is usually terminated by a zero byte which is octal 000 (not an ASCII zero which is 260 octal nor an ASCII null which is 200 octal), but in some cases a string may be terminated by any byte less than 200 octal.

ASCII - American Standard Code for Information Interchange. A seven-bit code used for data transfers in and out of the computer, usually with even parity; ie, the eighth bit is set to a "one" or "zero" such that the number of "one" bits in the byte is even. When ASCII is used internally to the IRIS system, the eighth bit is unconditionally set to a "one". An Appendix in the IRIS User Reference Manual lists all 128 ASCII codes.

ACCOUNT ID - A string consisting of a word or group of words and symbols up to a maximum of 12 characters which must be typed in by a user to activate his terminal and log him on to the system. There are over 4 x 10↑21 possible account ID's.

PORT - An interactive input/output channel on the IRIS system. For in-house operation, each port is usually hard-wired to one terminal. For remote operation, each port is connected to a data set, thereby allowing several terminals to use the same port (one at a time). There is an input/output buffer, a data file table, and an active file associated with each port.

IPL - the first phase of the IRIS startup procedure is the Initial Program Load which brings a fresh copy of REX, SIR, and DBUG into core from the system disc.

SYSGEN - A System Generation (SysGen) is the process of writing the operating system and all other files on the system disc (Logical Unit number zero). All files currently on the system disc will be lost if a SysGen is performed, but user files may be saved by first copying them to another Logical Unit or to paper tape or magnetic tape.

Copyright 1978
Educational Data Systems

TASK - A core-resident routine written in a specific form and invoked by use of a QUEUE system command to put an entry on the task queue. The task is executed when its task node reaches the top of the task queue. See Section 8.13.

JOB - A set of tasks required to perform a desired complete function such as running a user's program.

FILE - A set of blocks on a Logical Unit which may be accessed from a processor or, via a processor, by a program. A file consists of a file header block, from zero to 128 header extender blocks, and from zero to 65000 data blocks. A file may contain a system processor or other system component, a binary core image of a stand-alone program, a high level language program (such as a Business BASIC program), text (including assembly language source code), or any other type of data. See Section 1.4 for more information.

FILE HEADER - The first block of a file. The header does not contain any user's data. What the header does contain is the Filename, related information such as file type and protection, and the disc address of each data block in the file (data block disc addresses are not listed in the header of a contiguous file). A data file header may also contain a data record format map (formatted file) or directory information (indexed file). See Section 1.4 for more information.

ACTIVE FILE - There is an active file associated with each interactive port. All programs and data entered through the port are generally placed in its active file by a processor. The active file does not have a Filename unless it has been saved. The active file is active in the sense that it is used for swapping; ie, it is used to store a user's program and local data between time slices.

PASSIVE FILE - All user files stored on the disc under a Filename are passive files. A passive program file may be copied into a port's active file for processing without disturbing the passive file. The SAVE command replaces a passive file or creates a new passive file by copying the active file into the passive file.

PRIVATE FILE - A passive file which has a password included in the Filename (offset by a CTRL E) so as to prevent access to the file by other users on the same account.

PUBLIC FILE - A passive file with no password (no CTRL E) included in the Filename.

FILENAME - The word Filename is used to represent the name (and optional password) assigned to a file by a user at the time he creates the file. If a password is used, it is separated from the name by a CTRL E character (written E). The name and password may be any combination of letters, digits, and periods, except that the first character of the name must be a letter. Acceptable Filenames include:

NAMEEPASSWORD
XY14Q99
TE543382XB.5

The password is not echoed (not printed) during type-in. This is to prevent other users on your account from learning your passwords and gaining access to files you wish to keep private. The E must be typed again after the password to resume normal echo operation. Up to fourteen characters total are allowed including the E code, making a total of over 6 x 10↑21 possible Filenames.

The same Filename may be used on different Logical Units; therefore, any Filename given by a user is assumed to be on that user's assigned Logical Unit unless otherwise specified by entering the Filename in the form

lu/Filename

where lu is the number of the Logical Unit where the file is to be found or built. The same Filename may be used to identify one file on each Logical Unit. Any time a new file is being built, the Filename may be given in the form

$ddd.cc <pp> lu/Filename!

where ddd.cc is the amount to be charged to the account of any other user who accesses the file, and pp is the desired protection (see Section 8.7). The cost and protection may be given in either order, and both are optional, but the Filename must be given last. If not specified, the cost will be zero and the protection will be 77. An exclamation mark following the Filename allows this file to replace another file of the same type and on the user's own account, in which case the old file's cost and protection will be used if not specified here.

For a peripheral driver file there is a special form of Filename consisting of a dollar sign and the device's mnemonic; for example, $PTR is the paper tape reader, and $LPT is the line printer. Dollar sign Filenames are also used for system subroutine replacements. The first character after the dollar sign in such a Filename must be a letter.

PHYSICAL UNIT - A secondary data storage unit such as a disc cartridge. Data are transferred to and from the Physical Unit in blocks of 256 words.

LOGICAL UNIT - A Physical Unit (or one partition of a Physical Unit) which has been formatted by a SysGen or by INSTALL for use by IRIS. Usually an entire Physical Unit is treated as one Logical Unit, but provision is made for partitioning a Physical Unit into two or more units; this may be necessary in the case of a very large disc pack where the Real Address would otherwise require more than one word. A Logical Unit may not include more than one Physical Unit; eg, the fixed disc and the removable cartridge on a dual drive must be treated as two (or more) Logical Units. Each Logical Unit has a copy of BZUP in Real Address zero, an INDEX whose header is in Real Address one, an ACCOUNTS file whose header is at Real Address three, and a disc map (DMAP) whose header is at Logical Cylinder zero, Logical Track one, Logical Sector zero (Real Address LRT).

CYLINDER - A set of tracks on one disc drive that can be accessed without moving the heads. The term is derived from the high stack disc packs where the set of tracks is a set of equal sized circles around the same axis, one on each disc surface. Such circles are all on the surface of the same imaginary geometric cylinder. Note: a head-per-track disc is considered as having one cylinder (number zero), whereas the cylinder number specifies the head position on a moving arm disc.

SURFACE - One side of one platter of a disc or disc cartridge. A 2315 or 5440 type cartridge has only one disc platter and hence two surfaces. Other discs or cartridges may have from one to twenty usable surfaces.

TRACK - The path traced on a disc surface by one head in one position. A track is the intersection of one cylinder with one surface.

SECTOR - A fraction of one revolution of a disc. May be thought of as a pie shaped section of the disc. Within this amount of rotation, one block of data may be recorded in each track.

BLOCK - The intersection of one track with one sector. Each block will store 256 16-bit words of data. Formatted files on magnetic tape are also written in blocks of 256 words.

DISC ADDRESS - A disc address is a means of identifying a particular disc block on a given Logical Unit. A disc address may be in any of three forms as follows:

LOGICAL ADDRESS - In this form the Logical Cylinder, Logical Track, and Logical Sector are carried as binary integers in separate words (see definitions of these terms below). This form is generally used only in the system's allocate and deallocate routines and in some disc drivers.

REAL ADDRESS - A 16-bit binary number ranging from zero to one less than the number of blocks on a Logical Unit. This is the form carried within the system in file headers, etc. In this form, the address is packed into one binary word according to the formula

Real Disc Address = LC x LRC + LT x LRT + LS

where: LC = Logical Cylinder
LT = Logical Track
LS = Logical Sector
LRC = Logical-to-Real Cylinder Conversion Factor
LRT = Logical-to-Real Track Conversion Factor

LRT must be equal to the number of sectors (NSCT), and LRC must be equal to the number of blocks in a cylinder (NSCT times NTRK, where NTRK is the number of tracks). It is not necessary to define a logical-to-real sector conversion factor since it would always equal one.

PHYSICAL ADDRESS - The form of disc address used by the disc controllor. The disc driver subroutine converts the Logical Unit number and the Real Address into a Physical Address and outputs it to the disc controller. For some controllers, the Physical Address is identical to the Real Address; otherwise, the Physical Address is never stored within the system.

LOGICAL CYLINDER - An integer ranging from zero to one less than the number of cylinders in a Logical Unit. The Logical Cylinder is equal to the real cylinder unless the Physical Unit is partitioned into two or more Logical Units, in which case the physical cylinder number is derived by adding the Logical Cylinder to the number of the first real cylinder.

LOGICAL TRACK - An integer ranging from zero to one less than the number of heads on a disc drive. The Logical Track number selects one head (track) for reading or writing.

LOGICAL SECTOR - An integer ranging from zero to one less than the number of sectors on a disc. The Logical Sector selects the sector time during which a transfer takes place, thereby selecting one block within the track specified by the Logical Cylinder and Logical Track numbers.

Copyright 1978
Educational Data Systems

Appendix 4: OCTAL NUMBERS

The octal number system (radix eight) is used when operating the computer via its control panel, although the computer operates internally on binary (radix two) "words", each word consisting of sixteen bits (binary digits). Each indicator light or data switch on the control panel represents a single bit. The lights and switches are arranged in groups of three, each group representing one octal digit. The following table gives the light or switch configurations for each octal digit:

Octal Digit	Switches or Lights
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

A "0" represents a dark lamp or a switch in the down position (binary zero), and a "1" represents a lighted lamp or a switch in the up position (binary one).

Note that there are five octal groups on the panel, and one bit by itself at the left end, making up the sixteen bit word. For example:

106743 = 1 000 110 111 100 011

Copyright 1978
Educational Data Systems

Appendix 5: SOFTWARE DEFINITIONS

This Appendix lists the IRIS Software Definitions and the REX Page Zero Definitions of the latest issue in effect at the time this manual was printed. Source tapes of the Software Definitions (DEFS) and the Page Zero Definitions (PZ) are supplied with IRIS systems delivered by EDSI, and new tapes will be supplied on request for a nominal production charge.

This Appendix should be used for reference only, and the latest tapes should be used for all assemblies of new IRIS modules.

```
; SOFTWARE DEFINITIONS FOR "IRIS" R7.3
; 2-13-78
;                 All Rights Reserved
;    Copyright © 1974, Educational Data Systems
;    Copyright © 1978, Educational Data Systems
;   This document may not be reproduced without the
; prior written permission of Educational Data Systems

; MISCELLANEOUS.DEFINITIONS
.DUSR K     =400    ;BYTE SWAP CONSTANT
.DUSR NOP   =100010 ;NO OPERATION
.DUSR INFO  =600    ;SYSTEM INFORMATION TABLE
.DUSR BPS   =10200  ;BEGINNING OF PROCESSOR STORAGE
.DUSR MBUS  =20200  ;MINIMUM BEGINNING USER STORAGE
.DUSR LSIR  =14000  ;LOCATION FOR SIR (MIN IS BPS+3600)
.DUSR LSYSL =34000  ;LOCATION FOR SYSL

; DEFINITIONS FOR FLAGCHANGE SUBROUTINE
.DUSR SET   =140000 ;SET FLAG BIT
.DUSR RESET =100000 ;RESET FLAG BIT
.DUSR TOGGL =040000 ;TOGGLE FLAG BIT
.DUSR SKIPO =020000 ;SKIP IF RESULT IS ONE
.DUSR SKIPZ =010000 ;SKIP IF RESULT IS ZERO

        .RDX 10

; DISPLACEMENTS IN TASK QUEUE NODE
 ;    A2   =0 ;REGISTER A2
 ;    A1   =1 ;REGISTER A1
 ;    A0   =2 ;REGISTER A0
 ;    A3   =3 ;REGISTER A3
 ;    PC   =4 ;PROGRAM ADDRESS * 2 + CARRY
.DUSR FLAP =5 ;FLAGS & CURRENT PRIORITY
.DUSR PRIL =6 ;PRIORITY INCREMENT & LIMIT
.DUSR TASK =7 ;TASK ENTRY POINTER
.DUSR TCBP =8 ;TASK CONTROL BLOCK POINTER
.DUSR LINK =9 ;LINK POINTER TO NEXT NODE

        .RDX 8
```

The next page lists displacements into the system information table (INFO).

```
.DUSR SDAT.= 0 ;SYSTEM CREATION DATE (HOURS AFTER 1-1-76)
.DUSR SPED.= 1 ;AVG CPU SPEED (INSTR/MSEC), NOVA 3 FLAG
.DUSR MILU.= 2 ;MAXIMUM # INSTALLED LOGICAL UNITS
.DUSR NDCH.= 3 ;NUMBER OF DATA CHANNELS PER PORT
.DUSR LPCA.= 4 ;LOCATION OF PORT CONTROL AREA
.DUSR TNAP.= 5 ;TOTAL NUMBER OF ACTIVE PORTS

.DUSR MBUS.= 7 ;MINIMUM BEGINNING OF USER STORAGE
.DUSR TOPW.=10 ;TOP WORD OF CORE TO BE USED
.DUSR ABUF.=11 ;AUXILIARY BUFFER SIZE (NUMBER OF WORDS)
.DUSR TBUF.=12 ;MAG TAPE BUFFER SIZE (NUMBER OF WORDS)
.DUSR NCQN.=13 ;NUMBER OF EXTRA CHARACTER QUEUE NODES
.DUSR NNOD.=14 ;MINIMUM NUMBER OF FREE NODES
.DUSR NSIG.=15 ;NUMBER OF SIGNAL BUFFER NODES
.DUSR NSUB.=16 ;MAXIMUM NUMBER OF DISCSUBS
.DUSR KTSL.=17 ;COEFFICIENTS FOR TIME SLICE CALCULATION
.DUSR KSCH.=20 ;COEFFICIENTS FOR TIMESHARING SCHEDULER
.DUSR KPV1.=21 ;COEFFICIENTS FOR PARTITION VALUATION
.DUSR KPV2.=22 ;COEFFICIENTS FOR PARTITION VALUATION
.DUSR .TSA.=23 ;TEMPORARY STORAGE "A" POINTER (6 WORDS)
.DUSR .TSB.=24 ;TEMPORARY STORAGE "B" POINTER (6 WORDS)
.DUSR .TSQ.=25 ;TEMPORARY STORAGE "Q" POINTER (6 WORDS)
.DUSR .TSZ.=26 ;TEMPORARY STORAGE "Z" POINTER (6 WORDS)
.DUSR .TSC.=27 ;TEMPORARY STORAGE "C" POINTER (16 WORDS)
.DUSR HRS. =30 ;CPU TIME - HOURS SINCE 1-1-76
.DUSR TSC. =31 ;   PART OF HOUR IN TENTH-SECONDS
.DUSR CPLU.=32 ;CURRENT PROCESSOR LOGICAL UNIT
.DUSR CPDA.=33 ;CURRENT PROCESSOR DISC ADDRESS
.DUSR CPTN.=34 ;CURRENT PROCESSOR TYPE NUMBER
.DUSR SDFT.=35 ;SIZE OF EACH PORT'S DATA FILE TABLE
.DUSR DSCO.=36 ;DISC ADDRESS OF "SCOPE"
.DUSR DBYE.=37 ;DISC ADDRESS OF "BYE"
.DUSR DDSP.=40 ;DISC ADDRESS OF "DSP"
.DUSR DSUB.=41 ;DISC ADDRESS OF "DISCSUBS"
.DUSR DMSG.=42 ;DISC ADDRESS OF "MESSAGES"
.DUSR DCON.=43 ;DISC ADDRESS OF "CONFIG"
.DUSR .TTT.=44 ;POINTER TO TERMINAL TYPE TABLE
.DUSR MASK.=45 ;INITIAL INTERRUPT MASK
.DUSR ..STK=46 ;POINTER TO "CALL" STACK POINTER
.DUSR .RGS.=47 ;POINTER TO REGISTER BUFFER FOR "CALL"
.DUSR .TASQ=50 ;POINTER TO TASK QUEUE POINTER
.DUSR .DBS.=51 ;POINTER TO D(BLOCK IN BSA) CELL
.DUSR .RCV.=52 ;POINTER TO RECOVER ROUTINE
.DUSR .LUT.=53 ;POINTER TO LOGICAL UNIT TABLE
.DUSR .TBA.=54 ;POINTER TO MAGNETIC TAPE BUFFER AREA
.DUSR .ELB.=55 ;POINTER TO END OF LAST DISC BUFFER
.DUSR .IHT.=56 ;POINTER TO INTERRUPT HANDLER TABLE
.DUSR .SGB.=57 ;POINTER TO SIGNAL BUFFER POINTERS
.DUSR .BPT.=60 ;POINTER TO BUFFER POOL TABLE
.DUSR .UPT.=61 ;POINTER TO USER PARTITION TABLE
.DUSR THTC.=62 ;TEN HERTZ TASK COUNTER
.DUSR OVLA.=63 ;OVERLOAD CONDITION ACCUMULATOR
.DUSR NBBP.=64 ;NUMBER OF BUFFERS IN BUFFER POOL
.DUSR PFRF.=65 ;POWER FAIL RECOVER FLAG
.DUSR BPSP.=66 ;BEGIN PATCH SPACE (AFTER LAST PATCH)
.DUSR ENDP.=67 ;END OF PATCH SPACE (SET BY SIR)
```

```
; PORT CONTROL BLOCK (PCB) DISPLACEMENTS

.DUSR ICW. = 0 ;INPUT CONTROL WORD
.DUSR OCW. = 1 ;OUTPUT CONTROL WORD
.DUSR FBA. = 2 ;FIRST BYTE ADDRESS OF I/O BUFFER -1
.DUSR LBA. = 3 ;LAST BYTE ADDRESS OF I/O BUFFER
.DUSR IBP. = 4 ;INPUT BYTE POINTER
.DUSR OBP. = 5 ;OUTPUT BYTE POINTER
.DUSR LIB. = 6 ;LAST INPUT BYTE POINTER
.DUSR LOB. = 7 ;LAST OUTPUT BYTE POINTER
.DUSR TIB. =10 ;TEMPORARY INPUT CHARACTER BUFFER
.DUSR TOB. =11 ;TEMPORARY OUTPUT CHARACTER BUFFER
.DUSR FLW. =12 ;FLAG WORD (SEE BELOW)
.DUSR ULU. =13 ;USER'S ASSIGNED LOGICAL UNIT NUMBER
.DUSR URA. =14 ;USER'S RETURN ADDRESS
.DUSR ORA. =15 ;OLD RETURN ADDRESS
.DUSR CTU. =16 ;CPU TIME USED (UPPER HALF = HOURS)
.DUSR CTL. =17 ;CPU TIME USED (LOWER HALF = TENTH-SECONDS)
.DUSR ACT. =20 ;ACCOUNT NUMBER, PRIVILEGE LEVEL
.DUSR TON. =21 ;CPU TIME AT LOG-ON (MINUTES)
.DUSR PRI. =22 ;USER'S ASSIGNED PRIORITY
.DUSR EFP. =23 ;EFFECTIVE CURRENT PRIORITY, PAC
.DUSR DQT. =24 ;DEQUEUE TIME (TENTH-SECONDS)
.DUSR DFT. =25 ;POINTER TO DATA FILE TABLE (SEE PAGE 5)
.DUSR PDC. =26 ;PAUSE DELAY COUNTER (TENTH-SECONDS)
.DUSR AHA. =27 ;ACTIVE FILE HEADER DISC ADDRESS
 ;           =30 ;(SPARE)
.DUSR NLP. =31 ;NODE LINK POINTER FOR $TERMS STORAGE
.DUSR SND. =32 ;POINTER TO DRIVER'S "SEND" SUBROUTINE
.DUSR OCC. =33 ;OUTPUT COLUMN COUNTER
.DUSR ODC. =34 ;OUTPUT DELAY COUNTER
.DUSR RDE. =35 ;RETURN DELAY, EOM OR TERMINAL TYPE CODE
.DUSR PCW. =36 ;PORT CONTROL WORD FOR $MMUX (SEE PAGE 4)
.DUSR TTN. =37 ;TERMINAL TYPE NUMBER & FLAGS (SEE PAGE 4)


; EACH BIT IN FLW IS A FLAG AS FOLLOWS:
;  BIT*   MEANING
;   15_ BINARY INPUT/OUTPUT MODE (PASS BYTE AS IS)
;   14
;   13  DSP BREAKPOINT IS SET
;   12_ DSP IS ACTIVE ON THIS PORT
;   11  SIGNAL WILL ACTIVATE FROM PAUSE
;   10  A BREAK HAS BEEN DETECTED
;    9_
;    8  PROCESSOR TASK IS ON QUEUE
;    7  OUTPUT IS ACTIVE
;    6_ INPUT IS ACTIVE
;    5  LOG OFF AFTER PAUSE DELAY
;    4  IGNORE CTRL E (LOG-ON MODE)
;    3_ IGNORE CTRL O
;    2  ENABLE XOFF AND XON
;    1
;    0  ECHO INPUT CHARACTERS

; * NOTE:  BIT 15 IS THE MOST SIGNIFICANT BIT
; EACH BIT IN PCW IS A FLAG AS FOLLOWS:

;  BIT*   MEANING
;   15_ 0
;   14  THIS PORT IS ON EDSI MIGHTY-MUX
;   13  0
;   12_ DEVICE CONTROL OUTPUT (1 = HIGH, 0 = LOW)
;   11  NORMAL DEVICE STATUS INPUT (1 = HIGH, 0 = LOW)
;   10  THIS IS A PHANTOM PORT
;    9_ AUTO LOG-OFF ENABLED
;    8  AUTO FREQUENCY SCAN ENABLED
;    7  INHIBIT PARITY CHECK AND GENERATION
;    6_ TWO STOP BITS (NORMAL = ONE)
;    5  \  CHARACTER LENGTH: 11 = 8 BITS, 10 = 7 BITS
;    4  /                    01 = 6 BITS, 00 = 5 BITS
;    3_ EVEN PARITY (IF ENABLED)
;    2  \   CURRENT BAUD RATE:
;    1   )    7 = 9600, 6 = 4800, 5 = 2400, 4 = 1200,
;    0  /     3 = 600,  2 = 300,  1 = 150,  0 = 110.


; FORMAT OF TTN WORD, USED IN $TERMS:

;  BIT*   MEANING
;   15_ SPECIAL DELAY CHARACTERS EXIST - SEE $TERM.xxx
;   14
;   13
;   12_
;   11
;   10
;    9_
;    8
;    7  ESCAPE SEEN IN INPUT, USE TRANSLATION TABLE #1
;    6_ OUTPUT TRANSLATION IN PROGRESS
;    5  INPUT TRANSLATION IN PROGRESS
;    4  EXPECTING CURSOR POSITION ('RD' HAS BEEN SENT)
;    3_ \
;    2   ) TERMINAL TYPE NUMBER (0 - 17 OCTAL)
;    1   )
;    0  /

; * NOTE:  BIT 15 IS THE MOST SIGNIFICANT BIT
; DFT HAS EIGHT WORDS PER CHANNEL AS FOLLOWS:

.DUSR FLU = 0 ;FILE'S LOGICAL UNIT NUMBER
              ;-1 IN FLU ==> DEVICE (ON LU #0)
.DUSR FDA = 1 ;FILE HEADER DISC ADDRESS
.DUSR CBN = 2 ;CURRENT BLOCK NUMBER
              ;CBN = INIT ENTRY ADDRESS IF DEVICE
              ;CBN = PARTITION ENTRY POINTER IN CH # -1
.DUSR STS = 3 ;CHANNEL STATUS (SEE BELOW)
.DUSR FSZ = 4 ;FILE'S SIZE (# DATA BLOCKS IF CONTIGUOUS)
              ;FSZ = RECORD NUMBER IF NOT CONTIGUOUS FILE
.DUSR WPR = 5 ;NUMBER OF WORDS PER RECORD
.DUSR FRR = 6 ;FIRST REAL RECORD NUMBER
.DUSR CNP = 7 ;CHANNEL NODE POINTER (FOR MODULAR FILES)



; EACH BIT IN CHANNEL STS IS A FLAG AS FOLLOWS:

;  BIT*   MEANING
;   15_ RECORD IS LOCKED (IN CHM1 ==> PROGRAM IS LOCKED)
;   14  FILE IS WRITE PROTECTED
;   13  FILE IS CONTIGUOUS
;   12_ FILE IS NOT FORMATTED
;   11  PERIPHERAL DEVICE
;   10  FILE IS INDEXED
;    9_ (RESERVED FOR BYTE NUMBER OVERFLOW)
;    8 \
;    7  \
;    6_  \
;    5    )  DISPLACEMENT OF
;    4    ) RECORD INTO BLOCK
;    3_   ) (NUMBER OF BYTES)
;    2   /
;    1  /
;    0 /

; * NOTE:  BIT 15 IS THE MOST SIGNIFICANT BIT
; TASK NUMBERS FOR "QUEUE"

.DUSR   PROCE = 0 ;PROCESSOR IN CHANNEL #-1
.DUSR   TENHZ = 1 ;TEN HERTZ TASKS
.DUSR   SIGNA = 2 ;SEND SIGNAL
.DUSR   ESCAP = 3 ;ESCAPE FROM PROCESSOR


; DEFINE SPECIAL FLAGS FOR DISCSUBS

.DUSR X =40000 ;EXTENDED SUBROUTINE (TWO BLOCKS)
.DUSR N =20000 ;INCLUDED WITH ANOTHER IF CORE-RESIDENT
.DUSR D =10000 ;VERSION IS DISC-RESIDENT ONLY
.DUSR A =04000 ;ALTERNATE VERSION FOR CORE RESIDENCY


; DISCSUB NUMBERS FOR "CALL"

.DUSR FAULT = 0+D+X ;PRINT TRAP MESSAGE, ABORT TASK
.DUSR ALLOC = 1     ;ALLOCATE DISC BLOCKS
.DUSR DALLC = 2     ;DEALLOCATE DISC BLOCKS
.DUSR FFILE = 3     ;FIND FILE IN INDEX
.DUSR EXTEN = 4     ;CHANGE TO EXTENDED FILE
.DUSR ALCON = 5     ;ALLOCATE A CONTIGUOUS FILE
.DUSR CDTA  = 6     ;CONVERT DRATSAB TO ASCII
.DUSR CIA   = 7     ;CONVERT INTEGER TO ASCII (ANY RADIX)
.DUSR CSTR  =10     ;COMPARE STRINGS
.DUSR PASSC =11     ;PASSWORD COMPARE
.DUSR ERROR =12     ;ERROR ROUTINE FOR BASIC
.DUSR MESSA =13     ;CANNED MESSAGE TO I/O BUFFER
.DUSR BREAK =14     ;BREAKPOINT SETUP FOR DSP
.DUSR ACNTL =15     ;ACCOUNT LOOKUP
.DUSR DELET =16     ;DELETE A FILE
.DUSR PDELE =17+N   ;DELETE A PROCESSOR OR DRIVER
.DUSR BUILD =20+X   ;BUILD A NEW FILE
.DUSR BILDD =21+N+X ;BUILD A "$" FILE
.DUSR OPEN  =22     ;OPEN A FILE OR A DEVICE
.DUSR OPENU =23+N   ;OPEN A FILE OR A DEVICE FOR UPDATE
.DUSR OPENL =24+N   ;OPEN AND LOCK A FILE OR A DEVICE
.DUSR OPENR =25+N   ;OPEN A FILE OR A DEVICE FOR REFERENCE
.DUSR CLOSE =26     ;CLOSE A CHANNEL
.DUSR CLEAR =27     ;CLEAR A CHANNEL
.DUSR GETRR =30+X   ;GET RECORD FOR READ
.DUSR GETRW =31+N+X ;GET RECORD FOR WRITE
.DUSR FINDI =32     ;FIND AN ITEM (NOT IMPLEMENTED)
.DUSR READI =33     ;READ AN ITEM
.DUSR WRITI =34+N   ;WRITE AN ITEM
.DUSR WRITN =35+D   ;WRITE A NEW ITEM
.DUSR READC =36     ;READ FROM CONTIGUOUS FILE
.DUSR WRITC =37+N   ;WRITE INTO CONTIGUOUS FILE

                    ; (CONTINUED ON NEXT PAGE)
; DISCSUB NUMBERS (CONTINUED)

.DUSR CHARG =40       ;CHARGE FOR FILE ACCESS
.DUSR SYSCO =41       ;TRANSMIT A SYSTEM COMMAND (CALL 98)
.DUSR CNVDA =42       ;CONVERT DATE TO ASCII
.DUSR CNVAD =43       ;CONVERT ASCII TO DATE
.DUSR CNVDT =44       ;CONVERT DATE AND TIME (CALL 99)
.DUSR RDFHI =45       ;READ FILE HEADER INFO (CALL 97)
.DUSR SPECI =46       ;SPECIAL FUNCTIONS
.DUSR RECOV =47+D     ;RECOVER FROM A STALL OR CRASH
.DUSR PATNF =50       ;PSEUDO DIVIDE ARC TANGENT FUNCTION
.DUSR PLOGF =51       ;PSEUDO DIVIDE NATURAL LOG FUNCTION
.DUSR PSQRF =52       ;PSEUDO DIVIDE SQUARE ROOT FUNCTION
.DUSR PEXPF =53+X     ;PSEUDO DIVIDE EXPONENTIAL FUNCTION
.DUSR PSINF =54+X     ;PSEUDO DIVIDE SINE FUNCTION
.DUSR PCOSF =55+N+X   ;PSEUDO DIVIDE COSINE FUNCTION
.DUSR PTANF =56       ;PSEUDO DIVIDE TANGENT FUNCTION
.DUSR LINKP =57       ;LINK PROGRAMS (BASIC'S "CHAIN")
.DUSR DIREC =60       ;SET UP DIRECTORIES FOR INDEXED FILE
.DUSR SEARC =61+X     ;SEARCH INDEXED FILE DIRECTORY
.DUSR SHUFF =62       ;SHUFFLE DIRECTORY BLOCKS
.DUSR DEKEY =63       ;DELETE KEY FROM DIRECTORY
.DUSR RELEA =64+N     ;RELEASE A DIRECTORY BLOCK
.DUSR FIXDI =65+X     ;FIX DIRECTORIES OF MOVED INDEXED FILE
.DUSR REQUE =66       ;RE-ORDER TASK QUEUE
.DUSR AFSET =67       ;SET UP ACTIVE FILE FOR SWAP-OUT
.DUSR SIGPA =70       ;SIGNAL OR PAUSE
.DUSR MRDS  =71+N     ;MAG TAPE READ STATUS
.DUSR MTASK =72       ;MAG TAPE SUPPLEMENTARY TASKS
.DUSR MRFHD =73       ;MAG TAPE READ FILE HEADER
.DUSR MRFIL =74       ;MAG TAPE READ INPUT FILE
.DUSR MTFPE =75+X     ;MAG TAPE READ/WRITE TRANSFERS
.DUSR MNEXT =76+N     ;MAG TAPE GO TO NEXT DRIVE
.DUSR MTAPA =77+X     ;MAG TAPE ALL OTHER FUNCTIONS
.DUSR RWCTU =100      ;READ/WRITE CASSETTE TAPE UNIT

.DUSR PACK  =102      ;PACK NUMERIC TO STRING
.DUSR UNPAC =103+N    ;UNPACK STRING TO NUMERIC
.DUSR EDITN =104      ;NUMERIC EDIT
.DUSR EDITA =105+N    ;ARITHMETIC EDIT
.DUSR EDITD =106+N    ;DATE EDIT (DD/MM/YY)
.DUSR REOPT =107+X    ;RE-OPTIMIZE INDEXED FILE DIRECTORY
.DUSR FINDF =110+N    ;FIND A FILE (CALL 96)
.DUSR RDISC =111      ;READ OR WRITE WORD TO DISC (CALL 95)
.DUSR CHFLT =112      ;CHANGE FILE TYPE (CALL 94)
.DUSR WRWRD =113+N    ;WRITE WORD TO CORE (CALL 93)
.DUSR XCOM1 =114      ;FOR BI-SYNC DRIVER
.DUSR XCOM2 =115      ;FOR BI-SYNC DRIVER

.DUSR XCODE =117      ;AUXILIARY INPUT CODES FOR BASIC


 ; DISCSUB NUMBERS 130 - 137 ARE TO BE
 ; USED FOR CUSTOMER'S OWN SUBROUTINES.
 ; NUMBERS UP TO 777 OCTAL MAY BE USED
 ; IF NSUB IN INFO TABLE IS INCREASED.
; CORE-RESIDENT SUBROUTINE NUMBERS FOR "CALL"

.DUSR SCOPE =@ 0 ;EXIT TO "SCOPE" PROCESSOR
.DUSR RPZER =@ 1 ;RELEASE PARTITION ZERO
.DUSR CHKCH =@ 2 ;CHECK CHANNEL
.DUSR ALLCL =@ 3 ;CLEAR ALL CHANNELS
.DUSR FOFC  =@ 4 ;FIND OPEN FILE (CONTINUE)
.DUSR FOFI  =@ 5 ;FIND OPEN FILE (INITIALIZE)
.DUSR LOADU =@ 6 ;LOAD USER'S ACTIVE FILE
.DUSR DISCB =@ 7 ;READ OR WRITE A GROUP OF BLOCKS
.DUSR UNLOC =@10 ;UNLOCK RECORD
.DUSR WONA  =@11 ;WAIT FOR OUTPUT NOT ACTIVE
.DUSR CHKRP =@12 ;CHECK READ PROTECTION
.DUSR CHKWP =@13 ;CHECK WRITE PROTECTION
.DUSR CHKCP =@14 ;CHECK COPY PROTECTION
.DUSR MOVEW =@15 ;MOVE WORDS
.DUSR MOVBY =@16 ;MOVE BYTES
.DUSR CBSA  =@17 ;CHECK "BSA CHANGED" FLAG
.DUSR CRLA  =@20 ;CONVERT REAL TO LOGICAL DISC ADDRESS
.DUSR CLRA  =@21 ;CONVERT LOGICAL TO REAL DISC ADDRESS
.DUSR CPPPN =@22 ;CONVERT PCB POINTER TO PORT NUMBER
.DUSR CPNPP =@23 ;CONVERT PORT NUMBER TO PCB POINTER


; MAGNETIC TAPE DRIVE TABLE (IN $MTA0, $MTA1, ETC.)

.DUSR CFN. = 0 ;CURRENT TAPE FILE NUMBER
.DUSR CRN. = 1 ;CURRENT TAPE RECORD NUMBER
.DUSR MDE. = 2 ;CURRENT OPERATING MODE
.DUSR STT. = 3 ;CURRENT STATUS OF DRIVE
.DUSR CHA. = 4 ;CURRENT CHANNEL ADDRESS
.DUSR PHD. = 5 ;POINTER INTO HEADER (0 IF CONTIGUOUS)
.DUSR PXH. = 6 ;POINTER INTO EXTENDER (0 IF NOT EXTENDED)
.DUSR NWC. = 7 ;WORD COUNT (NEGATIVE)
.DUSR MCN. =10 ;COUNTER
.DUSR USR. =11 ;CURRENT USER
.DUSR ITH. =12 ;ADDRESS OF INTH IN $MTAS
.DUSR UNT. =13 ;DRIVE NUMBER


; COMMAND DISPLACEMENTS FROM INTH IN $MTAS

.DUSR LADR =-21 ;LAST CONTROLLER ADDRESS
.DUSR TSK. =-20 ;SET TASK
.DUSR SGD. =-17 ;SIGNAL DONE ROUTINE
.DUSR BSY. =-16 ;BUSY SUBROUTINE
.DUSR EFG. =-15 ;ERROR FLAG
.DUSR RDP. =-14 ;REGNANT DRIVE POINTER
.DUSR RDT. =-10 ;REGNANT DRIVE TABLE (4 WORDS)
.DUSR MWT. = -7 ;WRITE RECORD
.DUSR MRD. = -6 ;READ RECORD
.DUSR WEF. = -5 ;WRITE EOF
.DUSR RWD. = -4 ;REWIND
.DUSR SPC. = -3 ;SPACE FORWARD
```

```
; HEADER BLOCK DISPLACEMENTS (SEE MANAGER MANUAL)

.DUSR NAME = 0   ;FILENAME STRING (7 WORDS)
.DUSR ACNT = 7   ;PRIV LEVEL, ACCOUNT (GROUP, USER)
.DUSR TYPE = 10  ;FILE TYPE AND PROTECTION
.DUSR NBLK = 11  ;NUMBER OF BLOCKS IN FILE (INCL. HEADER)
.DUSR STAT = 12  ;FILE STATUS (SEE MANAGER MANUAL)
.DUSR NITM = 13  ;NUMBER OF ITEMS PER RECORD        \  ALSO
.DUSR LRCD = 14  ;LENGTH OF EACH RECORD (# WORDS)    ) USED
.DUSR NRPB = 15  ;NUMBER OF RECORDS PER BLOCK        )  BY
.DUSR NRCD = 16  ;NUMBER OF RECORDS IN FILE         /  DSP.
.DUSR COST = 17  ;DIMES CHARGED FOR ACCESS TO FILE
.DUSR CHGS = 20  ;TOTAL CHARGES FOR FILE USAGE (2 WORDS)
.DUSR LDAT = 22  ;LAST ACCESS DATE (HOURS, TENTH-SECONDS)
.DUSR CDAT = 24  ;FILE CREATION DATE (HOURS, TENTH-SECONDS)
.DUSR NTAC = 26  ;NUMBER OF TIMES ACCESSED
.DUSR CATR = 27  ;"CATALOG" RECORD NUMBER
.DUSR CLAS = 30  ;CATALOG CLASSIFICATION (2 WORDS)
 ;        32-34   (SPARE)
.DUSR PPRI = 35  ;PROGRAM'S ASSIGNED PRIORITY
.DUSR SNUM = 36  ;SCO NUMBER OF LAST SCO APPLIED
.DUSR ADAT = 37  ;DATE LAST SCO APPLIED (HOURS AFTER 1-1-76)
.DUSR DASA = 40  ;DECIMAL ACCUMULATOR SAVE AREA (10 WORDS)
.DUSR DSPS = 50  ;STORAGE FOR DSP (20 WORDS)
.DUSR FMAP = 70  ;DATA FILE FORMAT MAP (101 WORDS)
.DUSR HTEM =171  ;TEMP CELL USED BY ALLOC, DALLC & ACNTL
.DUSR STAD =172  ;START ADDRESS (DRIVER OR STAND-ALONE)
.DUSR ABLK =173  ;NUMBER OF ACTIVE DATA BLOCKS
.DUSR DSAF =174  ;DEFAULT SIZE OF ACTIVE FILE (# BLOCKS)
.DUSR CORA =175  ;CORE ADDRESS OF FIRST DATA BLOCK
.DUSR UNIT =176  ;LOGICAL UNIT NUMBER WHERE FILE RESIDES
.DUSR DHDR =177  ;REAL DISC ADDRESS OF HEADER BLOCK

 ; 200-377 HOLD REAL DISC ADDRESSES OF DATA BLOCKS

 ; CORE ADDRESSES ARE AT 400 WORD STEPS FROM CORA




; DISPLACEMENTS FOR EACH ENTRY IN "ACCOUNTS" FILE

 ;          0-5  ;ACCOUNT ID STRING
.DUSR APR. = 6   ;ASSIGNED PRIORITY
.DUSR ALU. = 7   ;ASSIGNED LOGICAL UNIT
.DUSR ACN. =10   ;ACCOUNT NUMBER (PRIV, GROUP, USER)
.DUSR CMR. =11   ;CONNECT MINUTES REMAINING
.DUSR CSR. =12   ;CPU SECONDS REMAINING
.DUSR MDB. =13   ;MAX. DISC BLOCKS ALLOTTED
.DUSR DBU. =14   ;DISC BLOCKS NOW IN USE
.DUSR PDB. =15   ;PEAK DISC BLOCK USAGE
.DUSR CHG. =16   ;FILE USE CHARGES (FLOATING 2-WORD BCD)
```

```
; LOGICAL UNIT FIXED INFORMATION TABLE (LUFIX)

.DUSR DINT =-24 ;POINTER TO INTERRUPT HANDLER
.DUSR DMSK =-23 ;DISC CONTROLLER'S MASK BIT
.DUSR DSIZ =-22 ;SIZE OF DRIVER (# WORDS)
.DUSR PFRD =-21 ;POWER FAIL RESTART DELAY
.DUSR EMSK =-20 ;"ANY ERROR" STATUS MASK
 ;          -17 ;"WRITE PROTECTED" MASK
 ;          -16 ;"NO SUCH DISC" MASK
 ;          -15 ;"DATA CHANNEL LATE" MASK
 ;          -14 ;"ADDRESS CHECK ERROR" MASK
 ;          -13 ;"ILLEGAL DISC ADDRESS" MASK
.DUSR IDRV =-12 ;"INITIALIZE DRIVER" SUBROUTINE ENTRY
.DUSR SLUR =-11 ;"SKIP IF LU READY" SUBROUTINE ENTRY
.DUSR SKNB =-10 ;"SKIP IF NOT BUSY" SUBROUTINE ENTRY
.DUSR REDS = -7 ;"READ STATUS" SUBROUTINE ENTRY
.DUSR SEEK = -6 ;"SEEK OR RECALIBRATE" SUBROUTINE ENTRY
.DUSR NSCT = -5 ;NUMBER OF SECTORS (BLOCKS PER TRACK)
.DUSR NTRK = -4 ;NUMBER OF TRACKS PER CYLINDER
.DUSR LRTC = -3 ;LOGICAL-TO-REAL TRACK CONVERSION FACTOR
.DUSR LRCC = -2 ;LOGICAL-TO-REAL CYLINDER CONVERSION FACTOR
.DUSR DFLG = -1 ;DISC FLAG WORD (SEE BELOW)
 ;             0 ;READ/WRITE ENTRY TO DRIVER ROUTINE


 ; NOTE:  BITS ARE USED IN DFLG WORD AS FOLLOWS:
 ;  BIT    15,  CHANGEABLE CARTRIDGE FLAG
 ;  BIT    14   FIXED HEAD DISC (USE ALLOC FOR ACTIVE FILE)
 ;  BIT    13   (UNUSED)
 ;  BIT    12,  (UNUSED)
 ;  BIT    11   SKIP SECTOR BETWEEN TRACKS WITHIN CYLINDER
 ;  BIT    10   SAME SECTOR NEXT TRACK \  NEXT BEST BLOCK
 ;  BIT     9,  NEXT SECTOR NEXT TRACK  ) IF DESIRED IS
 ;  BIT     8   NEXT SECTOR SAME TRACK /  NOT AVAILABLE.
 ;  BIT     7   CANNOT TRANSFER SEQUENTIAL SECTORS
 ;  BIT     6,  SECTORS ARE PHYSICALLY SEQUENTIAL
 ;  BITS  5-0   DEVICE ADDRESS



; LOGICAL UNIT VARIABLE INFORMATION TABLE (LUVAR)

.DUSR NCYL = 0 ;NUMBER OF CYLINDERS
.DUSR PART = 1 ;PARTITIONING INFORMATION
 ;           2 ;PARTITIONING INFORMATION
 ;           3 ;(RESERVED -- DO NOT USE!!)
.DUSR AVBC = 4 ;AVAILABLE BLOCK COUNT (SET BY SIR)
.DUSR MINB = 5 ;MIN. # OF BLOCKS FOR CREATING NEW FILE
 ;         = 6 ;(SPARE)
.DUSR FUDA = 7 ;FIRST UNUSED REAL DISC ADDRESS (SET BY SIR)
.DUSR ERRC =10 ;DATA CHECK ERROR COUNT
 ;         11 ;ADDRESS CHECK ERROR COUNT
 ;         12 ;DATA CHANNEL LATE COUNT


.EOT    ; "IRIS" R7.3 SOFTWARE DEFINITIONS
```

```
; "REX" PAGE ZERO DEFINITIONS FOR "IRIS" R7.3
; 1-23-78

;                     All Rights Reserved
;      Copyright © 1974, Educational Data Systems
;      Copyright © 1978, Educational Data Systems
;     This document may not be reproduced without the
; prior written permission of Educational Data Systems

C2=2
C3=3

RUP=5
RUS=RUP+1
RTP=RUS+1

.BSA=RTP+1
.HBA=.BSA+1
.HXA=.HBA+1
.SSA=.HXA+1
.ABA=.SSA+1

BBA=.ABA+1
BPI=BBA+1

CM400=21
C4=CM400+1
C5=C4+1
C6=C5+1
C7=C6+1

AI1=C7+1
AI2=AI1+1
AD1=AI2+1
AD2=AD1+1

C10=AD2+1
C11=C10+1
C12=C11+1
C13=C12+1
C14=C13+1
C15=C14+1
C16=C15+1
C17=C16+1
C20=C17+1
C37=C20+1
C40=C37+1
C77=C40+1
```

```
C100=C77+3
C177=C100+1
C200=C177+1
C205=C200+1
C215=C205+1
C240=C215+1
C244=C240+1
C260=C244+1
C271=C260+1
C300=C271+1
C334=C300+1
C377=C334+1
C400=C377+1
C777=C400+1
C1000=C777+1
C1777=C1000+1
C2000=C1777+1
C4000=C2000+1
C774C=C4000+1
C170K=C774C+1

ESCF=C170K+1
RTL=ESCF+1
BSACF=RTL+1
ERRF=BSACF+1

.CALL=ERRF+1

CALL=JSR @.CALL
FLAGC=JSR @.CALL+1
QCHAR=JSR @.CALL+2
QUEUE=JSR @.CALL+3
DQUEUE=JSR @.CALL+4
CHANNEL=JSR @.CALL+5
FREENODE=JSR @.CALL+6
```

```
.PCA=.CALL+13
.BPS=.PCA+1
.INFO=.BPS+1

C600=.INFO

.ACBY=.INFO+1
.ACIB=.ACBY+1
.ACSB=.ACIB+1
.BDIV=.ACSB+1
.BMUL=.BDIV+1
.BUMP=.BMUL+1
.DEC=.BUMP+1

.FALT=.DEC+2
.FIX=.FALT+1
.FLOT=.FIX+1
.FLUT=.FLOT+1
.IA2D=.FLUT+1
.IA2L=.IA2D+1
.INTR=.IA2L+1
.LODA=.INTR+1
.MSG=.LODA+1
.NRET=.MSG+1
.RBLK=.NRET+1
.SRET=.RBLK+1
.STBY=.SRET+1
.STDA=.STBY+1
.STI=.STDA+1
.STO=.STI+1
.STOB=.STO+1
.STPL=.STOB+1
.WBLK=.STPL+1
.XACB=.WBLK+1
.XSTB=.XACB+1
```

Copyright 1978
Educational Data Systems

```
DA  =160
DAC =164
DAS =165
DB  =166
DBC =172
DBS =173

.DA =174
.DA3=175
.DB =176
.DB3=177

C160=.DA
C163=.DA3
C166=.DB
C171=.DB3


BINDIVIDE   =JSR @.BDIV
BINMULTIPLY=JSR @.BMUL
BUMPUSER    =JSR @.BUMP
DECIMAL     =JSR @.DEC
FIX         =JSR @.FIX
FLOAT       =JSR @.FLOT
FINDLUT     =JSR @.FLUT
GETBYTE     =JSR @.ACBY
INBYTE      =JSR @.ACIB
INSTBYTE    =JSR @.ACSB
ISA2DIGIT   =JSR @.IA2D
ISA2LETTER =JSR @.IA2L
LOADDA      =JSR @.LODA
OUTBYTE     =JSR @.STOB
OUTTEXT     =JSR @.MSG
PUTBYTE     =JSR @.STBY
READBLOCK   =JSR @.RBLK
STORDA      =JSR @.STDA
STINPUT     =JSR @.STI
STOUTPUT    =JSR @.STO
TRAPFAULT   =JSR @.FALT
WRITBLOCK   =JSR @.WBLK
XGETBYTE    =JSR @.XACB
XPUTBYTE    =JSR @.XSTB

.EOT     ; "REX" R7.3 PAGE ZERO DEFINITIONS
```

Appendix 6: DRIVER ATTRIBUTES

After loading or replacing any system device driver or peripheral driver file, it may be necessary to modify the attributes table at the end of the driver before doing an IPL. Also, these attributes may be modified as required at any time to change buffer size, active file size, or certain default characteristics such as the EOM code, return delay, or special character delays.

This Appendix gives detailed information about setting up the attributes tables in various drivers. Section 11 of this manual describes IRIS drivers in general, and detailed descriptions of a driver's attributes table and port definition table are given on page 11-3. If the reader has not already done so, it is advisable to read Section 11.1 and to understand the examples shown in Figures 11.1 and 11.2 before attempting to modify the attributes or port definitions for any driver.

In some drivers there are additional parameters just ahead of the attributes table which may be set for special conditions. Since each driver has different characteristics, a separate section of this Appendix is devoted to each standard IRIS driver as follows:

page	driver	for
A6-2	-	General Information
A6-2	-	Master Terminal (port 0)
A6-3	$MTI	Multi-Terminal Interface
A6-4	$MMUX	EDS 300 or 310 Multiplexer
A6-6	$ALU	DCC ALU Multiplexer
A6-6	$DGMX	Data General 4060 Multiplexer
A6-6	$TTY50	Device 50/51 I/O Board
A6-6	$PHA	Phantom Ports

The following drivers do not have any parameters that can be changed by the system manager:

$RTC	Real Time Clock
$PTR	High Speed Paper Tape Reader
$PTP	High Speed Paper Tape Punch
$PTM	Paper Tape on Master Terminal
$DEC	Decimal Arithmetic Software
$DAU	Decimal Arithmetic Driver for Micro-N
$MTAS	Magnetic Tape System Driver
$TERMS	System Driver for Terminal Control

Copyright 1978
Educational Data Systems

Location BPS+1 (currently 10201) of any driver file contains a pointer to its attributes table, which is represented by "atrib" in the following example. To examine a driver's attributes, give DSP the commands:

```
        F $driver
        D10201
        10201: atrib . . . [press ESC]
        D atrib
```

where $driver is the Filename of the driver file, and "atrib" represents the address contained in location 10201.

When dumping the attributes table, the ESC key should be pressed after the second 177777 appears. The attributes table has the form:

```
        atrib: PCB location (or 0 if none)
               Interrupt mask bit word
               Device address code
                / Linkage pointer table  \
               {  (two words per pointer) }
                \     DO NOT CHANGE!!     /
               177777
                / Port definition table  \
               {  (eight words per entry) }
                \    Change as desired    /
               177777
```

The attributes table and port definition table are described in detail in Section 11.1. Some drivers also have special control information just ahead of the "atrib" cell. Usually, only such special information and the port definitions would be changed by the manager to specify a new arrangement of ports and/or default characteristics for each port. After logging on, an interactive user can change some of the characteristics of his port by use of the PORT system command. See "How to Change Other PORT Characteristics" in the IRIS User Reference Manual. The use of PORT, however, changes the port's attributes in core only; an IPL will again set up each port to the default values specified in the driver's port definition table.

MASTER TERMINAL (Port 0)

Port zero does not have an attributes table. The eight-word port definition table for the master terminal is in the REX file, and there is a pointer to it in location 200 of REX. If the system includes an EDSI 310 multiplexer with the device 10/11 option enabled, then port zero becomes a phantom port after a normal IPL with $MMUX active.

$MTI (Multi-Terminal Interface)

Use of $MTI requires $MMUX and set up of attributes information for the ports to be used. The system manager must determine the number of ports to be driven by $MTI and enter this number (in octal) at location ATRIB-2 in the $MTI file; this number of physically sequential ports must be available on the multiplexer. Then determine the physical port number of the first port to be used, and enter this number (in octal) at location ATRIB-1 in the $MTI file; this multiplexer port will become logical port number zero ("record" number zero) when READing or WRITing to $MTI from the application program.

The attributes table in $MTI must be empty. The ports selected above must be specified as non-interactive ports in the attributes table of the multiplexer driver (active file size = 0), and all other characteristics of the ports (speed, default EOM code, buffer size, etc.) must be specified there.

Set the number of signal buffer nodes (NSIG in CONFIG file) to be a minimum of twice the number of ports to be controlled by $MTI plus any other signalling requirements. It is recommended that NSIG be at least four times the number of $MTI ports. Do an IPL to activate all attributes specified above.

The PORT processor may be used later by the system manager to change the baud rate, return delay, and special character delays for specific ports. Note, however, that the physical port number rather than the $MTI port number must be given to PORT. PORT cannot be used to set up or change line lengths for $MTI ports since $MTI does not force a return at the end of a line. PORT can also be used to change the input terminating character, but this is not generally done since the application program can easily change the input terminator at any time. WARNING! A return delay change made by use of PORT may not be effective if $MTI is in use at the time the change is made.

Copyright 1978
Educational Data Systems

$MMUX (EDSI 300 or 310 Multiplexer)

In addition to setting up the Port Definition Table as described in Section 11.1, the word just before ATRIB must be set equal to the total number of ports (in octal) being defined. This number must also be exactly equal to the number of ports physically present in the Mighty-Mux system, even if not all ports are actually used. For example, if an EDS-301 expansion board with 16 ports is connected to the basic board (which has 8 ports), then the total number of ports defined in the $MMUX Port Definition Table must be exactly 24, and the word at ATRIB-1 must contain 30 octal.

The Port Control Word (the second word in each set of eight) must be set up in accordance with Section 11.1. In particular, be sure to set the "EDSI MUX" bit (ie, the octal 40000 bit).

Example of a Port Definition Table (PDT) for $MMUX -- Assume a Mighty-Mux system being used for the following configuration:

EDSI 310 mux with a 301-A8 expansion board (16 ports total).

Four interactive ports with CRT's on ports #1 - #4 running at 1200 baud, 7-bit character plus even parity bit, 135-byte I/O buffer, and 24-block active file.

One interactive port with 300-baud modem on port #5, 7-bit character plus even parity bit, 79-byte I/O buffer, 24-block active file, and automatic baud rate scan enabled.

Two unused ports (ports #6 and #7).

Eight ports used to interface CRT's to $MTI at 9600 baud, 7-bit character plus even parity bit, and 95-byte I/O buffer.

One non-interactive port running a line printer at 9600 baud, 8-bit character without parity, "ready" status = "high", and 512-byte I/O buffer.

The PDT for the configuration described on the preceding page would look like this:

```
           20    ;TOTAL # PORTS ON MMUX = 16
ATRIB:  66000    ;FIRST MMUX PORT'S PCB LOCATION
         2000    ;MASK BIT
           25    ;DEVICE ADDRESS
           -1    ;NO LINKAGE POINTERS
            4    ;FOUR CRT PORTS AS FOLLOWS:
        40054    ; 7-BIT CHAR, EVEN PARITY, 1200 BAUD
          207    ; 135-BYTE I/O BUFFER
            0    ; NO RETURN DELAY OR TERMINAL TYPE CODE
            0    ; NO SPECIAL CHARACTER DELAYS
            0
           30    ; 24-BLOCK ACTIVE FILE
            0    ; (THIS CELL NOT USED)
            1    ;ONE INTERACTIVE MODEM PORT
        55452    ; PCW (SEE PAGE 11-4)
          117    ; 79-BYTE I/O BUFFER
            0
            0
            0
           30
            0
            2    ;TWO UNUSED PORTS
            0
            0
            0
            0
            0
            0
            0
           10    ;EIGHT PORTS FOR $MTI
        40057    ; 7-BIT CHAR, EVEN PARITY, 9600 BAUD
          137    ; 95-BYTE I/O BUFFER
            0
            0
            0
            0    ; NO ACTIVE FILE (NOT INTERACTIVE)
            0
            1    ;ONE PORT FOR LINE PRINTER
        44077    ; PCW (SEE PAGE 11-4)
         1000    ; 512-BYTE I/O BUFFER
            0
            0
            0
            0    ; NO ACTIVE FILE
            0
           -1    ;END OF PORT DEFINITION TABLE
```

$ALU (DCC ALU Multiplexer)
and
$DGMX (Data General 4060 Multiplexer)

The instructions for $MMUX also apply to $ALU and $DGMX except for the following:

> Set the PCW word to zero. These multiplexers do not provide for software control of parity, data bits, or baud rate as does the EDS multiplexer. The user must make sure that his terminals match his multiplexer in these respects.
>
> Auto log off and $MTI are also not available for these multiplexers.

$TTY or TTY50 (Dual Device Code I/O Port)

The standard IRIS system provides for one extra I/O port in addition to the Master Terminal (always device code 10/11). Please note that the Master Terminal's driver is already built into IRIS. Activating $TTY enables a second I/O port with device code 50/51. Other device code pairs can be special ordered through EDS, and all can be enabled on the same system.

To set up ATRIB, there must be only one port definition table, of which the initial word is one and the next word (PCW) is zero. The parity, data bits, and baud rate are not under software control so the user must make sure that his terminal is set up properly in this respect.

$PHA (Phantom Ports)

The standard IRIS system comes with one phantom Port. More ports can be added by defining more ports in ATRIB in $PHA. If PRINT # channel is to be used in a BASIC program running on a phantom port, that port must have a standard size I/O buffer.

Appendix 7: TRAP NUMBERS

A trap number may appear in a trap message (see Section 6.1) in the form "TRAP #n AT [location] IN [processor]", where n is a number in the list on the following pages, [location] is the location within the processor (the address on its listing) where the fault was detected (or where a discsub was called that detected a fault), and [processor] is the Filename of the processor that was in core at the time the fault was detected.

For example, "TRAP #3 AT 7636 IN COPY" would indicate a disc error (data check error, seek error, data channel late, or disc time-out) which was not recoverable in 16 retries had occurred while running the COPY processor, and the actual trap occurred at location 7636. In the case of a TRAP #3, the actual location in the calling processor (COPY in this example) is given in the first status word (see page A7-3).

See page A7-2 for the meaning of the trap number and page A7-3 for the meaning of the contents of registers A0 through A2 in the trap status. Register A3 is usually the actual core address where the fault was detected.

The following shorthand notations are used:

a( )	"core address of"
d( )	"disc address of"
==>	"implies"
x	register contains no useful information

Trap numbers 100 and greater will occur only during an IPL, an INSTALL, or a SysGen, and the computer will halt after any such trap unless it determines that a retry is possible. Pages A7-4 and A7-5 give the meaning of the trap number and register contents, respectively, for traps #100 and up.

Copyright 1978
Educational Data Systems

Trap #	Meaning
0	Undetermined (refer to listings)
1	Disc is write protected
2	No such disc
3	Can't recover from disc error
4	Disc timed out
5	Illegal disc address
6	Disc busy before transfer started
7	Illegal or inactive Logical Unit
10	Illegal core address for disc read or write
11	Writing HBA, disc address or LU <> header in HBA
12	Fault in interrupt service (eg, lost character)
13	Active file too small and no blocks available
14	DISCSUB calls nested too deep
15	Called DISCSUB does not exist
16	Bad directory in indexed file
17	DSP has wrong file type (should be 77400)
20	BASIC not on Logical Unit zero
21	BASIC has wrong file type (should be 33702)
22	RUN not found on Logical Unit zero
23	RUN has wrong file type (should be 33602)
24	RUNMAT has wrong file type (should be 33402)
25	BASIC user area is too small (must increase LBSA)
26	NBLK < 1 for read/write file
27	NBLK > # disc addresses for read/write file
30	Impossible status returned by a subroutine
31	No data blocks in active file
32	Store output byte while output is active
33	File is larger than user partition
34	Processor aborted due to 25.6 seconds overtime
35	Impossible statement code in BASIC program
36	Illegal priority given to queue a task

Trap #	A0	A1	A2
1	Return address	Disc address	Core address
2	Return address	Disc address	Core address
3	Return address	Disc address	Error type (see * below)
4	Disc status	Disc address	Return address
5	Return address	Disc address	Core address
6	Return address	Disc address	Core address
7	Logical Unit #	Disc address	Return address
10	Return address	Disc address	Core address
11	Logical Unit #	Disc address	Return address
12	Trap address	x	x
13	# blocks short	x	x
14	Subroutine ID	Nesting limit	Return address
15	Subroutine #	x	Return address
16	Directory flags	d(header)	a(HBA)
17	x	x	x
20	BASIC's Unit #	x	x
21	x	BASIC's type	x
22	x	x	x
23	x	RUN's type	x
24	x	RUNMAT's type	x
25	a(NVS)	a(EUS)	a(BUS)
26	NBLK from header	d(header)	a(HBA)
27	x	a(end of header)	a(block not found)
30	x	x	x
31	x	# blocks	x
32	Return address	OUTTEXT address	PCB pointer
33	NBLK	Maximum NBLK	a(partition)
34	Address	(A1)	(A2)
35	Partition addr	Statement address	Statement code
36	Return address	Priority given	Node addr (see ** below)

* On a trap #3, the value in A2 indicates the type of disc error as follows:

0 ==> Data Error (CRC error detected by controller)
1 ==> Seek Error (Address check error)
2 ==> Data Channel Late (Missed DMA cycle)
3 ==> Time Out (Disc controller didn't go DONE)

** On a trap #36, the node is not on any queue or chain, so it may be examined by use of DSP if desired.

Copyright 1978
Educational Data Systems

The following traps can occur only during a SysGen, IPL, or INSTALL:

Trap #	Meaning
100	More than 16 sectors defined for disc
101	Not enough disc space for system
102	Disc block already marked
103	INDEX has too few entries (header may be clobbered)
104	Disc driver table in CONFIG is too large
105	Illegal DISCSUB number in core-resident list
106	SCOPE not on disc or not a processor
107	ACCOUNTS not on disc or not at real address 3
110	BYE not on disc or not a processor
111	Core overflow on IPL as configured
112	PCA overflows allotted core (too many ports)
113	DISCSUBS file doesn't exist or has been clobbered
114	Two DISCSUBS with the same number
115	DISCSUB with number greater than NSUB-1 (maximum)
116	Bad DISCSUBS object tape (has a gap or too large an address)
117	Not enough core for SysGen
120	Less than 16K core
121	Can't initialize for minimum configuration
122	Not as much core as specified by TOPW
123	LBSA is less than minimum allowable
124	Auxiliary buffer area overlaps PCA
125	Negative patch space (BPSP > ENDP in INFO table)
126	Too many lockable partitions
127	Partition size is too small or too large
130	Logical Unit is too large (FUDA > 177777)
131	Logical Unit zero is too large (FUDA > 100000)
132	Available block count exceeds FUDA
133	MBUS (at 607 in CONFIG) is too low for RUN or not 200 mod 400
134	Mighty-Mux missing (or has defective clock)
135	No $TERMS ($TERMS is required if any $TERMn is on system)
136	Too many $TERMn drivers on system (limit is 15)
137	Too many upper core partitions

Trap #	A0	A1	A2
100	# sectors	x	x
101	x	x	x
102	x	Disc address	x
103	NBLK	NRCD	a(HBA)
104	x	x	x
105	Subroutine #	May be NSUB	x
106	x	d(SCOPE)	400 ==> not a processor
107	x	d(ACCOUNTS)	3 ==> at wrong address
110	x	d(BYE)	400 ==> not a processor
111	x	x	x
112	TOPW + 1	x	a(end of PCA) + 1
113	0 ==> file empty	d(DISCSUBS)	a(HBA) = DISCSUBS header
114	x	Table pointer	DISCSUB number
115	x	x	DISCSUB number
116	x	0 ==> gap in tape	x
117	x	x	x
120	TOPW	x	a(INFO)
121	x	x	x
122	TOPW	x	a(INFO)
123	x	Minimum LBSA	x
124	a(end of ABUF)	- # words overlap	x
125	ENDP+1-BPSP	ENDP	BPSP
126	# partitions	# lockable	x
127	Partition size	Maximum size	Minimum size
130	FUDA	FUDA overflow	x
131	FUDA	x	x
132	FUDA	AVBC	x
133	MBUS	Minimum value	MBUS mod 400 (should=200)
134	x	x	x
135	x	x	x
136	x	x	x
137	# requested	# possible	x

Appendix 8: PAPER TAPE LOADER

A special paper tape binary loader is supplied with IRIS. This loader will accept tapes in standard Data General absolute format, but each record read from the tape is read into a buffer and checksummed before any data are stored in core. This prevents data from being stored in the wrong place due to a read error on the address word.

A special bootstrap program is required to load the EDSI paper tape loader into core. The sequence is:

Set data switches	Then press
xx7600	RESET, EXAMINE
060510*	DEPOSIT
063610*	DEPOSIT NEXT
000777	DEPOSIT NEXT
001400	DEPOSIT NEXT
004774	DEPOSIT NEXT
004773	DEPOSIT NEXT
105305	DEPOSIT NEXT
000776	DEPOSIT NEXT
171000	DEPOSIT NEXT
004767	DEPOSIT NEXT
107300	DEPOSIT NEXT
045013	DEPOSIT NEXT
151400	DEPOSIT NEXT
004763	DEPOSIT NEXT
105300	DEPOSIT NEXT
010411	DEPOSIT NEXT
000771	DEPOSIT NEXT
xx7604	START

* Change last digit to 2 for high speed reader

xx selects the highest available core module

The binary loader occupies the area between the above bootstrap program and the top of core. To use the loader, start at the top word of core (location xx7777) with switch zero down to select the master terminal's reader or up to select the high speed reader.

INDEX

IRIS MANAGER MANUAL INDEX

POINT 4 DATA CORPORATION

2569 McCabe Way / Irvine, California 92714 / (714) 754-4114