Educational Data Systems 17981 Sky Park Circle, Irvine, California 92707

INTERACTIVE REAL-TIME
INFORMATION SYSTEM

(IRIS)

SYSTEM

REFERENCE MANUAL

This manual is intended for use by persons intending to extend IRIS by the addition of a new module. It contains all information necessary to write a new processor, subroutine, or peripheral driver for the IRIS system.

The reader should refer to the IRIS Manager Reference Manual for information regarding operating the system, adding new modules to the system, and using utility packages. Refer also to the Glossary in the IRIS User Reference Manual.

August 1974
EDS 1019-4

IRIS System Reference Manual

TABLE OF CONTENTS

1. INTRODUCTION TO IRIS

IRIS is an Interactive Real-time Information System designed to support real time data acquisition, communications, interactive timesharing, and background 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 for the programmer who must write such an extension and add it to IRIS.

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.
TEX	The TEX file contains the remainder of the IPL routine (in the file's header), the Time-sharing Executive (TEX) 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 P.S. which may be used for troubleshooting SIR and TEX.
INDEX	The INDEX contains the Filename and disc address of each file on the disc. Each Logical Unit has its own INDEX.
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 are used too infrequently to be kept in core are stored in the Disc-Resident Subroutines (DISCSUBS) file. 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.

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

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 change the system configuration by modifying this file and then doing an IPL. All other components of IRIS are configuration independent.

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 get 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 while the system is in normal use.

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 or when it is desired to destroy all data on a given Logical Unit.

PLOAD — The Program Loader is used to load new files from paper tape to update or extend the system.

Other processors, such as BASIC, RUN, SAVE, KILL, COPY, and LIBR, are optional components of the complete IRIS environment, but they are not required for operation of the system.

1.2 Initial Program Load Sequence

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

The first step of an IPL is to get BZUP from the system disc block zero into core page zero. Refer to the section on Start Up and Shut Down Procedures in the IRIS Manager Reference Manual for the various methods of starting IPL. Also refer to the sections on BZUP and P.S. if the IPL sequence is to be interrupted for use of a debugging package.

Location 377 is 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 20000 (octal). If switch zero is up, control is then transferred to BZUP. If switch zero is down, then the TEX header is loaded into core, and control is passed to the IPL routine in the TEX header.

IPL reads the remainder of the TEX file into core by use of the disc driver in BZUP. If the switches are set to the starting location of P.S. (currently 21000 octal) control is then transferred to P.S.; otherwise, control goes to SIR. Control may be passed from P.S. to SIR by executing a jump to location LSIR (defined in the Software Definitions). LSIR is currently 10000 (octal). SIR examines the CONFIG file to bring the necessary disc drivers into core, sets up LUFIX and LUVAR tables for each disc, sets up the DISCSUB location table for CALL, locates the SCOPE, DSP, DISCSUBS, MESSAGES and BYE files and puts their disc addresses in the information table, requests a type-in of the date and time, sets up each port's Resident Table Area (RTA), Data File Table (DFT), and Input/Output buffer, scans the INDEX to create a fresh copy of the Disc Map (DMAP), and creates an active file for each port. Control is then transferred to the START routine in TEX where the interrupt system is initialized, and finally to the idle task.

SIR allocates space for Data File Tables, I/O Buffers, etc. in the shaded areas of the core map (see Section 1.3). The space above RTA is filled first, then the area between ENDPS and BPS, and finally below RTA. This sequence is used because some multiplexers require the RTAs to be in a particular location in core.

Figure 1.1: Map of Core

The diagram below shows how core is used by IRIS.

Location or label	Contents	Remarks
0	TEX Page Zero	Pointers, constants, decimal accumulator, etc.
177		
200	Processor Page Zero	This block is part of the processor file.
577		
600	INFO	System information table.
	TEX	Core-resident portion of the Time-sharing Executive
	Disc Driver	Driver for system disc.
PATSP	Patch space	As specified by manager.
ENDPS		See note below.
BPS	Processor / User storage	This area and locations 200-577 are occupied by one processor such as BASIC, RUN, SAVE, LIBR, etc. Space not occupied by the processor may be used by it for the user's active file.
BSA	BSA	Block Swap Area
HBA	HBA	Header Block Area
HXA	HXA	Header Extender Area
SSA	SSA	Subroutine Swap Area
ABA	ABA	Auxiliary Buffer Area (optional)
		See note below.
RTA	RTA	Resident Table Area
TOPW		

Note; shaded areas are allocated by SIR for Data File Tables, Drivers, I/O Buffers, etc. (see Section 1.3).

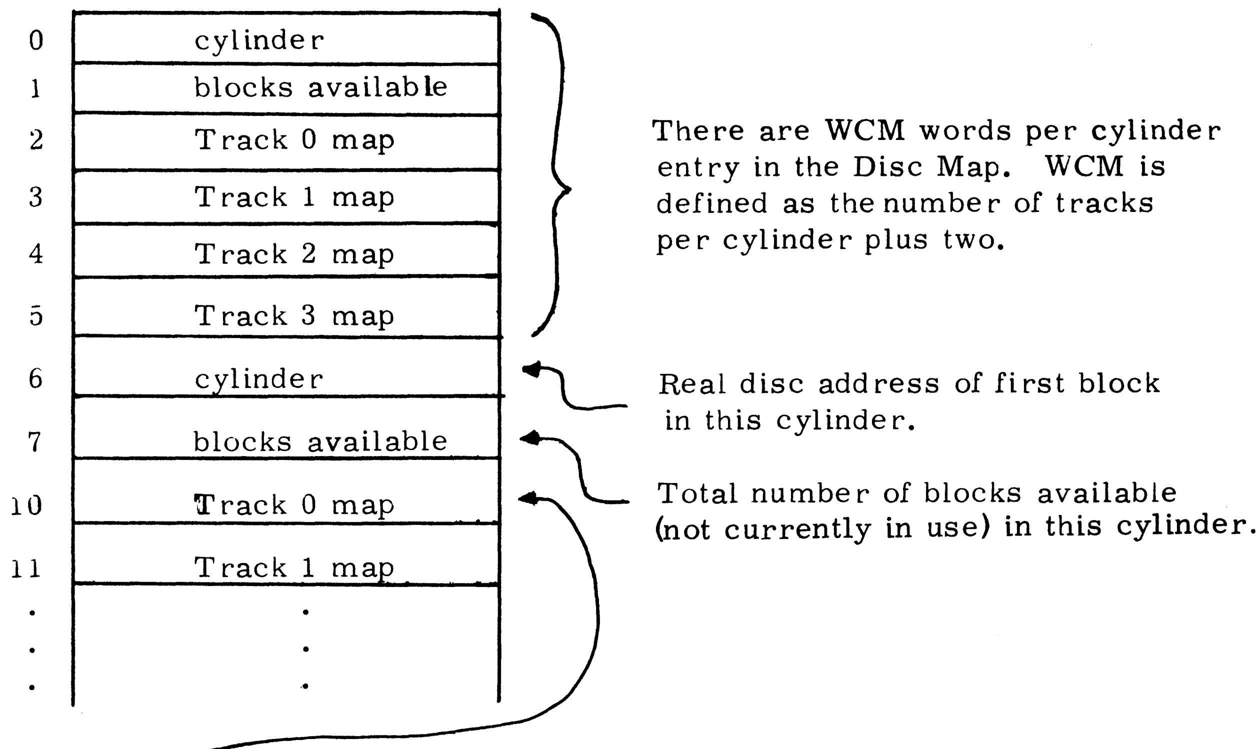
1.3 Disc and Core Usage

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. Figure 1.2 shows the structure of the disc map. The system disc (Logical Unit zero) also has the TEX (Time-sharing Executive) file whose header is at Real Disc Address two, and a DISCSUBS file which immediately follows 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.

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 TEX. 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 which contains various configuration and status information. The Real-time Executive occupies approximately the next 4K words, following which is patch space up to the defined Beginning of Processor Storage. The various disc block buffer areas begin near the top of core, and the processor may use all of the space from BPS to BSA.

The Resident Table Area (RTA) 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 shaded areas of the core map for Data File Tables, I/O buffers, stacks and tables used by TEX, peripheral drivers, etc.

Figure 1.2: Structure of Disc Map (DMAP)



Each track mapword maps the available blocks (sectors) in that track. A "one" bit indicates that the sector is unavailable or in use, and a "zero" bit indicates an available sector. The least significant bit represents sector zero. Bits at left end of word representing non-existant sectors are set to all ones.

The above example assumes four disc surfaces (four tracks per cylinder).

The DSPS cells and FMAP cells of the DMAP file 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.

Figure 1.3: File Header Displacements

Displacement symbol	Displacement value	Attribute (see text for details)
NAME	0	Filename (seven words)
ACNT	7	privilege level, account number
TYPE	10	file type and protection
NBLK	11	number of blocks in the file
STAT	12	file status
NITM	13	number of items per record
LRCD	14	length of each record (# words)
NRPB	15	number of records per block
NRCD	16	number of records in file
COST	17	charge for access to file (in dimes)
CHGS	20	charges for file access (income)
LDAT	22	last access date (hours, tenth-seconds)
CDAT	24	file creation date (hours, tenth-seconds)
CATR	26	CATALOG record number
	27	(27-47 not currently assigned)
DSPS	50	storage reserved for DSP (20 words)
FMAP	70	data file format map (101 words)
HTEM	171	temp cell used by system subroutines
STAD	172	starting address (driver or batch program)
DREP	173	disc address of replacing file
LUND	174	Logical Unit of data blocks
CORA	175	core address of first data block
UNIT	176	disc drive Logical Unit number
DHDR	177	Real Disc Address of header
	200	200-377 may hold Real Disc Addresses

Note: all values are in octal.

1.4 Disc File Structures

IRIS provides facilities for two structurally different forms of disc files: random and contiguous. Both consist 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 while the contiguous file's header contains only a value indicating the total number of blocks in the file. The differences are discussed in detail in Sections 1.6 and 1.7. 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. Displacements are defined in the Software Definitions for all of the attributes. The displacement symbol and its currently assigned value (in octal) are given in Figure 1.2 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.

ACNT - This 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, bit 13 write protected, bit 12 copy protected (against users at any lower privilege level)
- bit 11 read protected, bit 10 write protected, bit 9 copy protected (against users at the same privilege level)
- bit 8 runnable processor
- bit 7 load active file when selected
- bit 6 initiate input before first swap-in
- bit 5 (not used)
- bits 4-0 contain the file's type (see Section 1.5).

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

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

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

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 and delete the file.

NITM - In a formatted data file (type 31) this word specifies the number of items in each record, including the record written flag if used.

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

NRPB - The 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 digits.

LDAT - The 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 1973, and the second word represents the remaining part of an hour in tenths of a second.

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.

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.

DSPS - These sixteen words are reserved in each file's header for use only by the Disc Service Processor and other system routines.

FMAP - These 65 words are used only in a formatted data file (see Section 1.6). The format map cells in an active file header may be used for temporary storage by a processor since the active file can not be formatted.

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

STAD - For a machine code (batch or executable) file, this word indicates the program's starting address, or bit 15 is set to indicate 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. The STAD word is not used for other types of files.

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.

LUND - The Logical Unit number for the data blocks will be different from UNIT only if this is a copy of the file's header that has been placed on the system disc for faster access to the data blocks.

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.

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 LUND) 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 to 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 DHDR.

1.5 File Types

The lower five bits of the TYPE word in a file's header contain the actual file type discussed under "How to CHANGE File Characteristics" in the IRIS User Reference Manual. The type is used by SCOPE to match a program file to its related processor and by LUSR to load the active file only if its type matches the processor. The use of the file type is discussed further in Section 2 of this manual.

1.6 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:

0	end of map
1	
2	
3	
4	floating point binary
5	decimal (BCD)
6	
7	
10	
11	ASCII string
12	unsigned binary
13	
14	
.	
.	
77	file mark

Types left blank 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 (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 need be allocated to a formatted file.

1.7 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 can not be expanded at a later time except by building a new larger file and copying the data. Although it is up to the caller (a processor or an application program) to determine item locations and types within each record, the contiguous file does offer the sophisticated user several advantages over the formatted file:

- The maximum record length is 65535 words, compared to 256 words in a formatted file.
- The maximum size of a file is 65534 data blocks, compared to 32768 data blocks in a formatted file.
- Records are packed tightly, spanning block boundaries if necessary, rather than storing an integral number of records per block.
- A single data transfer may span record boundaries since the address (record number and byte position) specify only a starting location.

- The record's location is calculated from information in core rather than reading the file's header for a format map, thus saving up to two disc accesses per data transfer.

- Although all records are the same length, their formats may be different as determined by the application program.

Note: it is strongly recommended that the record length be made a power of two so that records are packed without spanning block boundries since transferring such a record requires two or more disc accesses.

1.8 Control and Information Tables

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

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. Its 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.

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

LUFIX pointer
LUVAR pointer
Logical Unit number

See Section 4.7 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.

RTA	Each active port has a Resident Table Area which contains various information about the state of the port and the user on the port. The Regnant User Pointer (RUP) in page zero points to the RTA of the user whose processor 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 RTA.
DFT	Each active port also has a Data File Table which can be found via a pointer in its RTA. Each DFT has six words per channel as described in the Software Definitions.

1.9 Flag and Status Words

One of the items in each RTA 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	Signal will activate from pause
10	A break has been detected
9	Suppress RETURN in RUN
8	System is to swap out active file
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	Suppress XOFF and XON
1	Suppress parity check
0	Echo input characters

One of the items in each 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	
9	(reserved for byte number overflow)
8	Displacement of record into block (number of bytes)
7	
6	
5	
4	
3	
2	
1	
0	

A blank entry in either of the above tables means that the bit is not currently in use.

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 are variations of binary-coded-decimal formats. All nine forms are shown in detail in Figure 2.4 along with some examples of a decimal number's appearance in octal notation. The floating binary and BCD formats are manipulated by software subroutines or by use of the EDS-10 Decimal Arithmetic Unit.

Figure 1.4: Number Formats

Unsigned binary integer (type 12 in a format map)

15 ... 0
16-bit binary integer

Signed binary integer (not used in a data file)

15	14 ... 0
S	15-bit binary integer (two's complement if < 0)

Floating point binary (type 4 in a format map)

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

Unsigned BCD (not used in a data file)

15 ... 12	11 ... 8	7 ... 4	3 ... 0
BCD digit	BCD digit	BCD digit	BCD digit

Signed BCD integer (see note 4)

15	14 ... 12	11 ... 8	7 ... 4	3 ... 0
S	3-bit digit	BCD digit	BCD digit	BCD digit

Floating point BCD (see notes 4 and 5)

word 0				word 1			
15 ... 12	11 ... 8	7 ... 4	3 ... 0	15 ... 12	11 ... 8	7 ... 1	0
BCD digit	BCD digit	BCD digit	BCD digit	BCD digit	BCD digit	characteristic (see note 6)	S

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

word 4	word 5	
15 0	15 1	0
characteristic (see note 6)	not used (all zeroes)	S

D represents a BCD digit

Notes:

1) A heavy dot represents the binary point or decimal point of the mantissa.

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

3) The numbers above each figure are bit position numbers.

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 1.6). The number is carried as absolute value and sign.

5) The three word and four word BCD formats are the same as the two word format shown, except that the mantissa holds ten or fourteen BCD digits, respectively.

6) The characteristic 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.

2. HOW TO WRITE A PROCESSOR

A processor is a machine language program written in a specific configuration for proper interaction with the Time-sharing Executive **(TEX). 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 accessable as a system command. This chapter provides all information necessary to write a new processor, add it to the system, debug it, and make it accessable for general use.

2.1 Core Locations and Entry Points

Core locations 200 through 577 (octal) and locations BPS through BSA-1 are available for use by a processor (see Figure 2.1). BPS (Beginning of Processor Storage) is defined in the Software Definitions tape, and .BSA (a pointer to BSA) is in page zero of TEX along with many other pointers (refer to a listing of the TEX Page Zero Definitions). The available space may be used as desired by the processor, except that cells 370 through 373 are reserved for pointers to the "swap-in", "swap-out", "escape" and "control C" entry points, and the "initial entry" is at location BPS. Usually the processor occupies the page zero area 200-577 and additional space starting at BPS, using the space between the end of the processor and BSA for data and for user program storage. The disc block buffer areas (BSA, HBA and HXA) may also be used if certain restrictions are observed.

The user's I/O buffer, data file table, and other information are found in his RTA (Resident Table Area) via RUP (the Regnant User Pointer) in TEX page zero. RUP is set by the system before swap-in to point to the current user's RTA.

2.2 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 Section 2.9). In any case, SCOPE finds and selects the desired processor and loads the address of BPS into the URA (User's Return Address) of the user's RTA (Resident Table Area).

FIGURE 2.1: PRØCESSØR CØRE LØCATIØNS

```
; THIS IS A TYPICAL PRØCESSØR FØR "IRIS"
; 9-10-73

; ALL RIGHTS RESERVED
; CØPYRIGHT (C) 1973 BY EDUCATIØNAL DATA SYSTEMS
; 2415 WINDWARD LANE, NEWPØRT BEACH, CALIF. 92660


.TXTM   1         ;REQUIRED FØR CØRRECT PACKING ØF TEXT
.LØC    INFØ-400  ;ALL PRØCESSØRS MUST START AT 200

; CELL 200 MUST CØNTAIN AN ASSEMBLED VALUE.
; (DØ NØT START WITH A .BLK ØR ANØTHER .LØC)

; USE THIS AREA FØR CØNSTANTS, PØINTERS, ETC.


.LØC     SWAPI    ;ENTRY PØINTERS

         SWPI     ;PØINTER TØ SWAP-IN SUBRØUTINE
         SWPØ     ;PØINTER TØ SWAP-ØUT SUBRØUTINE
         ESCR     ;PØINTER TØ ESCAPE RØUTINE
         CTLC     ;PØINTER TØ CØNTRØL C RØUTINE

; LØCATIØNS 374 THRØUGH 577 MAY BE USED FØR SWAP-IN AND
; SWAP-ØUT SUBRØUTINES ØR FØR ANY ØTHER DESIRED PURPØSE.

.LØC     INFØ-.  ;PAGE ZERØ ØVERFLØW TEST


.LØC     BPS      ;BEGINNING ØF PRØCESSØR STØRAGE

; INITIAL ENTRY IS AT THIS PØINT.
         .
         .
         .

.END     ;END ØF THE PRØCESSØR

; ALL REMAINING SPACE UP TØ BSA-1 MAY BE USED BY THE
; PRØCESSØR AS DESIRED.  TYPICALLY, THE ACTIVE FILE
; IS BRØUGHT INTØ THIS AREA BY THE SWAP-IN RØUTINE.

; SØME SPACE SHØULD BE RESERVED IN THE PRØCESSØR
; AREA FØR PATCHES DURING DEBUGGING PRØCEDURES.
```

2) At the next time slice the system loads the selected processor into core if it is not already in core, inserts a breakpoint jump if a DSP breakpoint has been set in this proccessor on this port, and does a JSR to the swap-in routine via the pointer in location 370. The swap-in routine performs any initializing required (see Section 2.4) and returns 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 as set up by SCOPE. Subsequent entries will resume operation (after the JSR to "swap-in") at the point where execution was terminated in the preceeding 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 (JSR @.STI).
   b) Processor wants to do output and it already has an output in progress (CALL WONA).
   c) End of time slice (JSR @.BUMP due to RTL $\leq$ 0).
   d) User presses ESC or CTRL C.
   e) Processor completes or aborts its task (CALL EXIT).
   f) Processor detects a hardware or software error (JSR @.FALT).
   g) A DSP breakpoint is encountered.

   Any of the first three 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. Condition e or f will select SCOPE as the user's processor; condition g will select DSP and cause the registers, carry bit, and a 65-word area of core to be saved. The action of the ESC and CTRL C keys depends 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, the processor should clear it and take whatever action is appropriate.

   b) If the processor is not in core for this user at the time ESC or CTRL C is pressed then the system's action consists of entering the processor via the pointer in location 372 or 373, respectively, for the next time slice (after the JSR to "swap-in"). CTRL C will act as an escape unless input is enabled.

4) After the time slice is terminated for any of the above reasons (except a JSR @.FALT or a DSP breakpoint) the system does a JSR to the processors's "swap-out" routine via the pointer in location 371. The swap-out routine must perform any wrap up required to save information for the next time slice (see Section 2.5).

2.3 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 size of the area between the end of the BASIC interpreter (RUN) and BSA, plus a block for its header. TEX 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 it merely has a pointer to a JMP 0,3 in cells 370 and 371 (see Figure 2.2). If the processor has 101 (octal) or less cells of interim storage required, it may use cells FMAP through FMAP+100 (octal) in the active file header for interim storage. The processor must read the active file header into HBA, copy its interim storage, and write the header out if it chooses this method. See Figure 2.3 for a programming example. The active file header's real disc address is contained in AHA of the regnant user's RTA.

If a processor has more than 101 (octal) words to save between swaps then it must use the active file. See Figure 2.4 for a programming example.

2.4 Swap In Procedure

Each time a user's time slice begins, the selected processor is brought into core and the system does a JSR to its swap-in routine via the pointer in location 370. If the active file and/or its header is used for storage between time slices then the swap-in routine must read it into core and perform any other initializing required. A "load user" subroutine is provided in TEX and may be reached by a CALL LUSR instruction sequence. LUSR reads the active file header into HBA and, if its type (lower five bits of TYPE word) matches the processor, the active file is also read into core, and LUSR does a skip return. LUSR does a non-skip return if the types don't match. Alternately, the swap-in routine may read the active file header itself, may read any other file or header into core as required, or may simply JMP 0,3 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 BPS with the address in URA; equality indicates that this is the initial entry.

2.5 Swap Out Procedure

After each time slice is terminated for any of the reasons given in Section 2.2 (except a fault or a breakpoint) the system will do a JSR to the processor's swap-out routine via the pointer in location 371. If no wrap up is required then location 371 may point to a JMP 0,3 instruction; otherwise, the swap-out routine must save all information necessary for the next time slice. Typically, the swap-out routine will either:

a) copy a temporary storage area in page zero into the FMAP through FMAP+100 cells of the active file header, and/or

b) set up the core address (CORA) and the disc addresses in the active file header for use by the system in writing out the active file.

If LUSR was called by the swap-in routine then the system will automatically write out the active file after the swap-out routine has been called, but the processor's swap-out routine must read the active file header into HBA. If only the active file header is used for temporary storage then the swap-out routine must write it out itself.

The active file contains SAF blocks, including the header, where SAF (Size of Active File) is defined in the System Configuration tape and may be modified later in the CONFIG file. The active file header contains the real disc addresses of these blocks, but another processor may have left them distributed anywhere in the last 200 words (octal) of the header. Each cell in the last half of a header is "wired" to a particular core address relative to CORA (see Section 1.4, File Structure). Also, any disc address in an active file header may be complemented to indicate that it is inactive (the block is not to be transferred in or out of core). The processor's swap-out routine must set CORA to the first core address of the active file area in core and position the disc addresses in the header so that there are disc addresses in true form for blocks that are to be transferred. All other disc addresses must be retained in the active file header in complemented form. Additional blocks may be allocated to the active file if SAF blocks are insufficient to hold the active area.

2.6 Use of System Subroutines

All system subroutines listed in APPENDIX 1 may be used by a processor. The most commonly used subroutines are described elsewhere in this section, and APPENDIX 1 lists all available system subroutines. 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.

2.7 Input/Output

All I/O is via a one-line buffer for each port. Pointers in each port's RTA 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 JSR @.STI instruction. The user is bumped and input is enabled. The processor will be swapped in and control returned to the next instruction after the user presses RETURN to terminate input.

Access Input Byte is then called by a JSR @.ACIB 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. A RETURN code (octal 215) indicates end of input.

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

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. 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.

Store Output Byte is called by a JSR @.STOB instruction to store the byte in the lower half of register A2 into the user's I/O buffer. The byte is returned in A0 with the top half of the word cleared. Buffer overflow is prevented by overlaying the last byte in the buffer rather than incrementing the pointer beyond the end of the I/O buffer.

Text Message Output is called by a JSR @.MSG instruction followed by a .TXTF "text" pseudo-op. Copies any given "text" string to the user's I/O buffer and returns to the next instruction following the text.

Canned Message Output is called by a CALL MESSAGE instruction sequence with the number of any available "canned" message in register A1. APPENDIX II of this manual lists the currently available messages.

Convert Integer to ASCII is called by a CALL CIA instruction sequence with an integer to be outputted 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; i.e. A for eleven, B for twelve, etc.

Start Output is called by a JSR @.STO instruction after using the above routines in any combination to store ASCII codes in the user's I/O buffer. The string of ASCII codes must be terminated by a zero byte by clearing A2 and executing a JSR @.STOB before starting output (this is not necessary if the last output was generated by a JSR @.MSG).

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

2.8 File Access

File I/O is 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 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 exist 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 HSLA (hours since last accessed) cell will be zeroed. CHANNEL OPENR will open the file, but the regnant user will not be charged for its use, the HSLA cell will not be changed, and the file will be marked as write protected. CHANNEL OPENU will do the same as OPEN except it will error on write protection. 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 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.

If the processor CALLs EXIT before closing the channel, the processor channels will be CLEARed and the file being built will be destroyed. If the new file was replacing an old file, the old file will be restored.

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 deleted when the last user has CLOSEd or CLEARed it.

Any new file being created must be closed by the processor by issuing a CHANNEL CLOSE instruction before exiting to the system. 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. Since the system clears all channels after a processor's exits to it, a CALL ALLCLEAR is usually not used by a processor.

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 or WRITITEM. 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 transferred to the device so that the processor does 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 the system's RBLK or WBLK to transfer 256 (decimal) words from 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 FOFC.

2.9 Processor Type

Each file's header has a TYPE word which is described under Disc File Structures (Section 1.4 of this manual). The file type of a processor, however, has special significance not discussed in that section.

Protection - All processors should be write protected to prevent inadvertant replacement or deletion. Read protect a processor only if it is for private use. Copy protection has no significance on a processor except to 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 1 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 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. A one in bit 7 indicates that SCOPE should continue to scan the command line for a program Filename and load the selected program file (if any) into the port's active file. Bit 6 indicates that SCOPE should not cause the new processor to be swapped in immediately but should start input and cause initial entry when input is terminated.

2.10 Debugging Procedures

If a new processor was created by use of ASSEMBLE it will be necessary to change it into a runnable processor. See "How to CHANGE File Characteristics" in the IRIS User Reference Manual.

The Disc Service Processor is a powerful tool for use in debugging a processor. The breakpoint is especially useful for this purpose. Refer to "How to Use DSP" in the IRIS Manager Reference Manual.

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. 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 EXIT in the main body of the code.

In some cases it is desirable to temporarily enter a JSR @.FALT instruction in the code in case the processor takes an unexpected branch. Such a case may occur following a call to a DISCSUB 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 users.

```
;        FIGURE 2.2:  PRØCESSØR WITH NØ SWAPPING


; THIS PRØCESSØR PRINTS "I AM A PRØCESSØR"
; 9-10-73

; ALL RIGHTS RESERVED
; CØPYRIGHT (C) 1973 BY EDUCATIØNAL DATA SYSTEMS
; 2415 WINDWARD LANE, NEWPØRT BEACH, CALIF. 92660


.TXTM    1
.LØC     INFØ-400

CR:      215


.LØC     SWAPI    ;ENTRY PØINTERS

         .+4      ;SWAP-IN
         .+3      ;SWAP-ØUT
         CEXIT    ;ESCAPE
         CEXIT    ;CØNTRØL C
         JMP 0,3

CEXIT:   CALL
         EXIT


.LØC     BPS

XYZ:     LDA      2,CR      ;STØRE A RETURN CØDE
         JSR      @.STØB
         JSR      @.MSG     ;STØRE THE MESSAGE
         .TXTF    "I AM A PRØCESSØR"
         JSR      @.STØ     ;START ØUTPUT
         CALL
         WØNA
         JMP      XYZ


.END     ;"I AM A PRØCESSØR" SØURCE
```

FIGURE 2.3: SWAPPING STØRAGE IN ACTIVE FILE HEADER

```
.LØC    SWAPI   ;ENTRY PØINTERS

        SWPI    ;SWAP-IN SUBRØUTINE
        SWPØ    ;SWAP-ØUT SUBRØUTINE
        QEXIT   ;ESCAPE ENTRY
        QEXIT   ;CØNTRØL C ENTRY

QEXIT:  CALL    ;EXIT FRØM PRØCESSØR
        EXIT

        0
SWPI:   STA     3,.-1     ;SWAP IN SUBRØUTINE
        LDA     3,RUP     ;REGNANT USER PØINTER
        LDA     0,.BPS
        LDA     1,URA.,3
        SNE     0,1       ;INITIAL ENTRY ?
        JMP     @SWPI-1 ;   YES, START-UP NØT REQUIRED
        LDA     2,.HBA    ;PØINTER TØ HEADER BLØCK AREA
        LDA     1,AHA.,3;ACTIVE FILE HEADER ADDRESS
        SUB     0,0       ;ACTIVE FILES ARE ØN LØGICAL UNIT #0
        JSR     @.RBLK    ;READ ACTIVE FILE HEADER
        LDA     1,SAV     ;-[NUMBER ØF CELLS TØ SWAP-IN]
        LDA     3,SAV1    ;PØINTER TØ 1ST PAGE ZERØ CELL TØ SWAP IN
SWPI1:  LDA     0,FMAP,2;LØAD TEMPØRARY CELLS
        STA     0,0,3
        INC     3,3
        INC     2,2
        INC     1,1,SZR ;DØNE CØPYING ?
        JMP     SWPI1   ;   NØ
        JMP     @SWPI-1 ;   YES, RETURN TØ SYSTEM

SWPØ:   STA     3,SWPI-1;SWAP ØUT SUBRØUTINE
        LDA     2,.HBA    ;READ ACTIVE HEADER
        LDA     3,RUP
        LDA     1,AHA.,3
        SUB     0,0
        JSR     @.RBLK
        LDA     1,SAV     ;-[NUMBER ØF CELLS TØ SWAP-ØUT]
        LDA     3,SAV1    ;PØINTER TØ 1ST PAGE ZERØ CELL TØ SWAP-ØUT
        LDA     0,0,3
        STA     0,FMAP,2;STØRE TEMP CELLS IN FMAP ØF ACTIVE HEADER
        INC     2,2
        INC     3,3
        INC     1,1,SZR ;DØNE CØPYING ?
        JMP     .-5       ;   NØ
        LDA     2,.HBA    ;   YES
        LDA     3,RUP
        LDA     1,AHA.,3
        SUB     0,0
        JSR     @.WBLK    ;WRITE ØUT HEADER
        JMP     @SWPI-1 ; AND RETURN TØ SYSTEM
```

FIGURE 2.4: SWAPPING WITH ACTIVE FILE

```
.LØC    INFØ-400

.TS:    TS
TS:     .BLK    20


.LØC    SWAPI   ;ENTRY PØINTERS

        SWPI    ;SWAP-IN
        SWPØ    ;SWAP-ØUT
        ESCR    ;ESCAPE
        CTLC    ;CØNTRØL C


.LØC    400

SWPI:   STA     3,SWPØ-1;SET UP AFTER SWAP-IN
        CALL            ;LØAD USER'S ACTIVE FILE
        LUSR
        JMP     SWPI1   ;FILE TYPES DØN'T MATCH
        LDA     0,.TPZ  ;PAGE ZERØ SAVE AREA IN ACTIVE AREA
        LDA     1,C20   ;NUMBER ØF PAGE ZERØ CELLS SAVED
        ADD     0,1     ;LAST SØURCE ADDRESS FØR MØVE
        LDA     2,.TS   ;PØINTER TØ PAGE ZERØ TEMP STØRAGE
        JSR     @.MØVE  ;MØVE INTERIM STØRAGE INTØ PAGE ZERØ
        JMP     @SWPØ-1 ;RETURN TØ SYSTEM

SWPI1:  SUB     0,0     ;INITIALIZE ACTIVE FILE
        STA     0,TS
        STA     0,TS+1
        STA     0,TS+4
        STA     0,TS+6
        STA     0,TS+12
        LDA     2,.HBA
        LDA     0,PTYPE ;TYPE ØF THIS PRØCESSØR
        STA     0,TYPE,2
        STA     0,CØST,2;CLEAR CØST ØF ACTIVE FILE
        STA     0,NAME,2;ALSØ CLEAR THE NAME
        LDA     3,RUP
        LDA     1,AHA.,3;WRITE ØUT NEW HEADER
        JSR     @.WBLK
        JMP     @SWPØ-1 ;RETURN TØ SYSTEM
```

(FIGURE 2.4 CONTINUED ØN NEXT PAGE)

(FIGURE 2.4 CONTINUED)

```
        0
SWPØ:   STA     3,.-1     ;WRAP UP FØR SWAP-ØUT
        LDA     O,.TS
        LDA     1,C20
        ADD     O,1
        LDA     2,.TPZ    ;MØVE INTERIM PAGE ZERØ STØRAGE CELLS
        JSR     @.MØVE    ; TØ USER ACTIVE AREA
        LDA     2,RUP
        LDA     1,FLW.,2;SET BIT 8 ØF (FLW) SØ SYSTEM WILL
        LDA     O,C400    ; WRITE ØUT USER AREA
        ADD     1,0
        STA     O,FLW.,2
        LDA     1,AHA.,2
        LDA     2,.HBA
        SUB     O,0
        JSR     @.RBLK    ;READ ACTIVE HEADER INTØ HBA
        LDA     O,.TPZ    ;SET UP ACTIVE FILE
        STA     O,CØRA,2;BEGINNING ØF ACTIVE AREA IN CØRE
        LDA     O,NBLK,2
        LDA     1,C2
        SUB     1,0
        STA     O,TEMP    ;NUMBER ØF DISC ADDRESSES TØ SHUFFLE
        INC     2,2       ;PØINTERS FØR SHUFFLING DISC ADDRESSES
        MØV     2,3
        LDA     O,NBS     ;NUMBER ØF BLØCKS TØ SWAP ØUT
        NEG     O,0
SWPØ1:  INC     2,2       ;SET UP HEADER WITH CØNTINUØUS DISC ADDRESSES
        INC     3,3       ; FRØM 200 THRU 200+NBLK-1 (NØ EMPTY CELLS)
        LDA     1,DHDR,3
        SNZ     1,1       ;EMPTY SLØT ?
        JMP     .-3       ;  YES, IGNØRE
        SKZ     O,0       ;FINISHED WITH ACTIVE AREA ?
        JMP     .+4       ;  NØ
        SSP     1,1       ;  YES, ALREADY NEGATIVE ?
        JMP     .+4       ;  YES
        JMP     .+2       ;  NØ, NEGATE IT
        SSP     1,1       ;PØSITIVE ?
        NEG     1,1       ;  NØ, SET PØSITIVE
        STA     1,DHDR,2
        DSZ     TEMP      ;DØNE ?
        JMP     SWPØ1     ;  NØ
        JMP     @SWPØ-1   ;  YES, RETURN AND WRITE ØUT USER ACTIVE AREA


        .LØC    INFØ-.    ;PAGE ZERØ ØVERFLØW CHECK
```

3. DISC-RESIDENT SUBROUTINES

All disc-resident subroutines are assembled together from a single set of source tapes to produce a single object tape which is 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 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 of DISCSUBS into core and transferring control to the desired subroutine. Two lines of assembly code are required to call a 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 checks the selected channel before transferring control to the subroutine. One DISCSUB may call another, and subroutines may be nested up to NSTL levels deep in this manner. The nesting limit, NSTL, is defined in the INFO table. When one DISCSUB calls another, SSA is written on the disc to save any temporary storage cells in the first DISCSUB.

In the case of an extended DISCSUB, the first block is brought into HXA and the second block is brought into SSA. Caution must be exercized 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 or 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.

One of the tasks of the System Initializing Routine (SIR) is to scan the DISCSUBS file and set up the Disc Address Table (DAT) and the Starting Address Table (SAT) with the disc address and the actual in-core starting address, respectively, of each disc-resident subroutine. SIR also reserves NSTL-1 disc blocks for use in saving SSA when nesting subroutines.

Disc-resident subroutines are 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. In IRIS it is possible to eliminate some or all of these disc accesses and thereby enhance the system throughout by specifying that certain DISCSUBS routines are to become core-resident. See section 5.4 of the IRIS Manager Reference Manual for detailed information.

3.1 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; i.e., all storage reference instructions must use either relative addressing or page zero system pointers.

3. It must be self-initializing; i.e., 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 3.2), 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, STO, MSG, RBLK, WBLK, etc.), but routines such as BUMP, WONA, and STI, which might bump the user, may not be called.

Arguments may be passed both to and from the DISCSUB in registers A0, A1, A2, and the carry bit. A3 may also 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 @.CRET instruction for a non-skip return, or by a JMP 1,3 or a JMP @.SRET instruction for a skip return. Many DISCSUBS use a non-skip return under error conditions and a skip return when the task is successfully completed. Provision is also made for multiple skip returns by the two-word instruction:

```
JSR @.NRET
n*K!NRET
```

where n indicates the return point (e.g., 3 to skip three words after the call). This return is equivalent to a JSR @.CRET if n=0 or a JSR @.SRET if n=1. Obviously, A3 cannot pass information back to the caller in this case, but the other registers and carry may still be used. NRET has been defined such that the expression n*K!NRET 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.

The only legal exit from a DISCSUB other than a return to the caller is a JSR @.FALT 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 and the regnant user's active file to be aborted.

3.2 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. The first of these two 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 FAULT subroutine is labeled FALTX. The entry point must be the first word of the subroutine.

The new subroutine must be assigned a number, and its name is equated to this number on the Software Definitions tape. Be sure that the number of DISCSUBS routines does not exceed the definition for NSUB. If necessary, increase NSUB to be greater than the last DISCSUB number. The new routine is then edited into the DISCSUBS source tapes, and Discsubs is re-assembled. See "How to Replace DISCSUBS" in the IRIS Manager Reference Manual if this new version is to be put on the system without doing a complete system generation.

A single block may be added to DISCSUBS by using DSP to append a block and then to copy the new block from a newly assembled object file. DSP's R command may also be used to read an object tape into the new block. An IPL must be performed to make the new subroutine accessable via CALL or CHANNEL.

The 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 core-resident (part of REX)
bit 14	subroutine is extended (occupies 2 blocks)
bit 13	include with preceding routine if core-resident
bit 12	can't be made core-resident (use alternate)
bit 11	alternate version for core-residency only
bit 10	(not used)
bit 9	(not used)
bits 0-8	subroutine identification number

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.

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.

A completed DISCSUBS block would look something like this:

```
          .LOC  11400   ;"DISCSUBS" BLOCK #21

DSB21:    SINH
          SINHX-DSB21
          COSH                     Linkage Table
          COSHX-DSB21
          DSB21-D21E

SINHX:    JSR  SINHI
          102663                   This technique is recommended
          015252                   to get a table pointer, yet main-
          135661                   tain relocatability.
          002447
            :
SINHI:    STA   3,SADDR
            :
SADDR:    0
COSHX:    JSR   COSHI
            :
D21E      =.     ;END OF "DISCSUBS" BLOCK #21
.LOC      DSB21+400-. ;BLOCK OVERFLOW TEST
```

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 under a different name such as DSUB. Use the R and W commands in BZUP to copy this new version into the DISCSUBS file, then do an IPL. If new blocks are being added, use the A command in DSP to first append the required blocks to DISCSUBS. When finished copying blocks, kill the temporary file, or leave it on the disc for backup. If increasing NSUB without a complete system generation, NSUB in the CONFIG file must be increased.

3.3 How to Debug a DISCSUB

DSP may be used to examine and/or 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 subroutine call, but breakpoints cannot be set in the disc-resident subroutine itself. Two alternatives are possible, however: If there are no other users on the system, halts may be inserted in the routine. If the system is in use, insert a JSR @. FALT 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 JSR @. FALT will affect anyone who uses the subroutine, whereas a DSP breakpoint affects only the user who sets it. Other users should not be allowed to call a new routine, however, before it has been debugged.

3.4 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. These 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

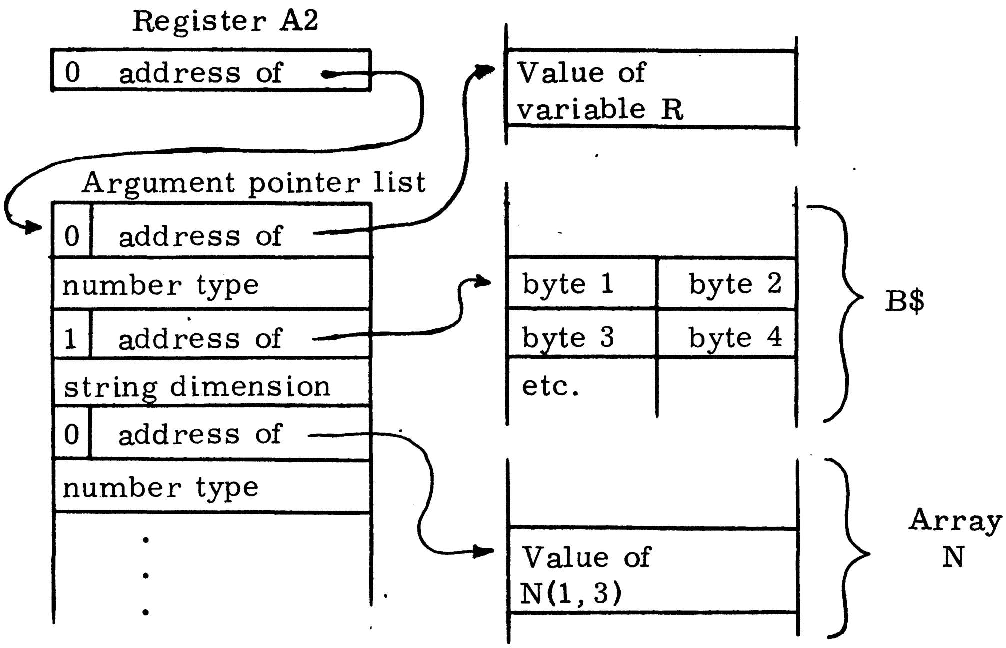
Registers A0 and A1 contain the first and last addresses of the currently unused cells in the BASIC user's storage area. This space is available for use as temporary storage by the subroutine.

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 parameters 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 sign bit of the pointer will be zero, 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). In the case of a string, the sign bit of the pointer will be one, the pointer will point to the word containing the first two bytes of the string. The string dimension will also be found in the parameter table following the pointer to the string.

For example, suppose a Business BASIC program contains the statement:

```
120 CALL 5, R, B$, N(1, 3)
```

These parameters will be passed to the subroutine number 5 as follows:



The remainder of the argument pointer list will be filled with pointers to a dummy variable which is ignored by 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 BASIC it must be included in the DISCSUBS file. Refer to Sections 3.1 and 3.2 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 203 (octal) in RUN. Look through CALLT for the first minus one (177777 octal), and replace it with a word containing the desired BASIC subroutine call number (in octal) in the lower (right hand) byte and the actual DISCSUB number in the higher (left hand) byte. Be sure the next cell in CALLT is 177777. CALLT may be extended through location 577 octal.

4. 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 application programs, or

2) A peripheral device which may be OPENed by any caller granted access; the caller then does input or output by READing or WRITEing to the device as if it were a data file, or

3) A Logical Unit having its own INDEX, thus allowing any caller granted access to READ, WRITE, and BUILD files on the device

User oriented devices such as typewriters, teleprinters, and CRT terminals are desirable as interactive ports, although a line printer - card reader combination 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 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 may be written such that some of its channels are interactive ports and others are used to interface peripheral devices.

4.1 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; e.g. $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 dependant on its type. The fifth pointer 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. 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 catagories 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 the pointer if a subroutine is not included. This catagory 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 and READ subroutines and pointers are replaced by SEND CHARACTER and SKIP IF NOT BUSY subroutines and pointers, respectively; SIR places absolute pointers to these subroutines in the SND and SNB cells of each RTA. Also, the FINIS pointer is replaced by a pointer to the first word of the driver which is to be core-resident. This catagory includes:
   a) Interactive device drivers (e.g. $TTY),
   b) Multiplexer drivers (e.g. $EDS8),
   c) System subroutines (e.g. $DEC), and
   d) System device drivers (e.g. $DAU).

The attributes table, which is at the end of the driver, consists of three words as follows:

ATRIB: This cell usually contains a zero. When brought into core, SIR puts a pointer to the first RTA (if any are assigned) into this cell. However, if the hardware requires a specific first RTA location, that location should be put at ATRIB rather than a zero, and SIR will attempt to allocate core to accomodate this requirement. (There may be no RTAs assigned; see below.)

ATRIB+1: This word should have a single "one" bit if desired to enable interrupts from the device (this bit of the system's mask word will be zeroed). This word may be zero if no interrupts are to be enabled, but it may not have more than a single one bit.

ATRIB+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 will be generated.

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 it must also have a power fail restart subroutine whose entry point is at INTH-1; the purpose of this routine is to re-initiate operation following a power failure. If possible, the restart should be done without loss of data.

ATRIB+2 is followed by a linkage pointer table and a port definition table. Each entry in the linkage pointer table consists of two words as follows:

a) absolute core location for pointer
b) assembled location to point to in file

The pointer table is terminated by a -1 which may be at ATRIB+3 if no pointers are to be generated. This -1 is immediately followed by the port definition table which consists of four words per entry as follows:

a) number of ports (add @ if interactive)
b) default speed (characters per second)
c) buffer size (number of bytes)
d) line length (number of characters)

An RTA is assigned for each port in this list; alternately, the driver may supply its own I/O buffer rather than supplying a list of ports here. The list of ports must also be terminated by a -1. The table may be empty, but the -1 terminator is required. No active files or data file tables should be assigned for peripheral devices. The attributes and these two tables must be entirely within the last block of the file.

Caution: The driver must be intrinsically relocatable 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.

4.2 How to Write a Peripheral Driver

Figure 4.1 shows the general form of a peripheral driver file. Everything from the pointer to ATRIB (location BPS+1) through the cell labeled ATRIB is brought into core by SIR, and the four pointers (ATRIB, FINIS, WRITE and READ) are modified to point to the actual

FIGURE 4.1: PERIPHERAL DRIVER FILE

```
.TXTM   1       ;FØR CØRRECT TEXT PACKING
.LØC    BPS     ;ALL DRIVERS MUST START AT BPS

        INTH    ;PØINTER TØ INTERRUPT HANDLER
        ATRIB   ;PØINTER TØ ATTRIBUTES TABLE
        FINIS   ;PØINTER TØ WRAP-UP SUBRØUTINE (ØR -1)
        WRITE   ;PØINTER TØ ØUTPUT SUBRØUTINE (ØR -1)
        READ    ;PØINTER TØ INPUT SUBRØUTINE (ØR -1)

INIT:   ---     ---     ;INITIALIZING RØUTINE
        ---
        JMP     0,3

        JMP     PFRST   ;PØWER FAIL RESTART ENTRY
INTH:   ---     ---     ;INTERRUPT HANDLER
        ---
        JMP     @.INTR

PFRST:  ---     ---     ;PØWER FAIL RESTART SUBRØUTINE
        ---
        JMP     0,3

FINIS:  ---     ---     ;WRAP-UP RØUTINE
        ---
        JMP     0,3

WRITE:  ---     ---     ;ØUTPUT SUBRØUTINE
        ---
        JMP     0,3

READ:   ---     ---     ;INPUT SUBRØUTINE
        ---
        JMP     0,3

ATRIB:  0       ;FIRST PØRT'S RTA LØCATIØN (SET BY SIR)
        400     ;MASK BIT
        XXX     ;DEVICE ADDRESS
                ;LINKAGE TABLE HERE IF REQUIRED (SEE TEXT)
        -1      ;LINGAGE PØINTER TABLE TERMINATØR
        .RDX 10
        5       ;NUMBER ØF RESIDENT TABLE AREAS TØ BE ASSIGNED...
        10      ; WITH THIS DEFAULT SPEED (CHARACTERS/SECØND)...
        200     ; THIS I/Ø BUFFER SIZE (NUMBER ØF BYTES)...
        75      ; AND THIS LINE LENGTH (NUMBER ØF CHARACTERS).
        3       ;ETC. (REPEAT THE FØUR PARAMETERS AS REQUIRED)
        30
        80
        80
        -1      ;PØRT DEFINITIØN TABLE TERMINATØR

.END    ;END ØF DRIVER
```

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 initializing subroutine when a caller OPENs the device and will JSR to the FINIS subroutine when the caller CLOSEs or CLEARs the channel. 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 READ and WRITE routines must look the same to the system as the READ ITEM and WRITE ITEM system subroutines. If a device does not have input capabilities there must be a -1 in place of the READ entry pointer, if there is no output capabilities there must be a -1 in place of the WRITE entry pointer, and if there is no wrap-up routine there must be a -1 in place of the FINIS pointer.

4.3 How to Write an Interactive or System Device Driver

A driver for a system device or an interactive device has the same form as one for a peripheral device with the following exceptions:

1) 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,

2) 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.

3) The READ routine is not used; the READ entry pointer is replaced by a minus one.

4) The WRITE routine is not used; the WRITE entry pointer is replaced by a pointer to a send (SND) subroutine which accepts a character in register A0 and outputs it to the port whose RTA pointer is given in register A2 (-1 in A0 means "start output", and -2 in A0 means "start input"),

5) Each port is assigned an active file and data file table if and only if bit 15 of the "number of ports" word is one (set by an @ symbol), and

6) The file type must be 77001.

FIGURE 4.2: SYSTEM DEVICE DRIVER FILE

```
.TXTM   1        ;FØR CØRRECT TEXT PACKING
.LØC    BPS      ;ALL DRIVERS MUST START AT BPS

        INTH     ;PØINTER TØ INTERRUPT HANDLER
        ATRIB    ;PØINTER TØ ATTRIBUTES TABLE
        INTH-1   ;PØINTER TØ FIRST CØRE-RESIDENT CELL
        SEND     ;PØINTER TØ "SEND CHARACTER" SUBRØUTINE
        -1       ;NOT USED

INIT:   ---      ---      ;INITIALIZING SUBRØUTINE
        ---
        JMP      0,3

        JMP      PFRST    ;PØWER FAIL RESTART ENTRY
INTH:   ---      ---      ;INTERRUPT HANDLER
        ---
        JMP      @.INTR

PFRST:  ---      ---      ;PØWER FAIL RESTART SUBRØUTINE
        ---
        JMP      0,3

SEND:   ---      ---      ;CHARACTER ØUTPUT SUBRØUTINE
        ---
        JMP      0,3

ATRIB:  0        ;FIRST PØRT'S RTA LØCATIØN (SET BY SIR)
        400      ;MASK BIT
        XXX      ;DEVICE ADDRESS
                 ;LINKAGE TABLE HERE IF REQUIRED (SEE TEXT)
        -1       ;LINGAGE PØINTER TABLE TERMINATØR
        .RDX 10
        7@       ;NUMBER ØF INTERACTIVE PØRTS TØ BE ASSIGNED...
        10       ; WITH THIS DEFAULT SPEED (CHARACTERS/SECØND)...
        200      ; THIS I/Ø BUFFER SIZE (NUMBER ØF BYTES)...
        75       ; AND THIS LINE LENGTH (NUMBER ØF CHARACTERS).
        1        ;NØ "@" ==> NØT AN INTERACTIVE PØRT
        165      ;          (NØ ACTIVE FILE, NØ DATA FILE TABLE)
        135
        132
        -1       ;PØRT DEFINITIØN TABLE TERMINATØR

.END    ;END ØF DRIVER
```

Note that in the case of a multiplexer each port may have a different I/O buffer size, line length, and speed. The line length and speed may be changed by the user after logging on to the port.

The SND pointer is converted to an absolute pointer by SIR and stored in the corresponding cell of each RTA for later use by the system.

Caution: The initializing routine must be entirely within the first block of the file, and the attributes table must be entirely within the last block (for a small driver these may be one and the same block).

4.4 How to Write a Multiplexer Driver

A multiplexer driver is the same as any other system device driver (see Section 4.3) 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 the presence or absence, respectively, of an @ symbol on each "number of ports" word in the port definition table. The @ symbol (i.e., a one in the top bit of the word) causes SIR to allocate an active file and a data file table for each port so designated. Absence of an @ symbol means no active file or data file table, hence the port cannot be used interactively, but it can be used to interface a peripheral device if a suitable peripheral driver if provided (see Section 4.5) and the multiplexer driver provides facilities for peripheral drivers as described below.

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    .+5           ; YES
LDA    3,TON.,2      ; NO
SKZ    3,3           ;PERIPHERAL SERVICE PROVIDED ?
JSR    1,3           ; YES
JMP    (service next port)
 .
 .     (service this port as an
 .     interactive terminal)
```

FIGURE 4.3: SYSTEM SUBRØUTINE FILE

```
.TXTM   1       ;FØR CØRRECT TEXT PACKING
.LØC    BPS     ;ALL DRIVERS MUST START AT BPS

        INTH    ;PØINTER TØ INTERRUPT HANDLER
        ATRIB   ;PØINTER TØ ATTRIBUTES TABLE
        DEC     ;PØINTER TØ FIRST CØRE-RESIDENT CELL
        -1      ;NØ "SEND CHARACTER" SUBRØUTINE
        -1      ;NØ "SKIP NØT BUSY" SUBRØUTINE

INIT:   ---     ---      ;INITIALIZING SUBRØUTINE
        ---
        JMP     0,3

DEC:    ---     ---      ;SUBRØUTINE ENTRY
        ---
        ---
        JMP     0,3


ATRIB:  0       ;NØ RTA
        0       ;NØ MASK BIT
        0       ;NØ DEVICE ADDRESS
        .DEC    ;THIS PAGE ZERØ PØINTER...
        DEC     ;IS TØ PØINT TØ THIS LØCATIØN
        .XXX    ;REPEAT PØINTER PAIRS AS REQUIRED
        XXX
        -1      ;LINGAGE PØINTER TABLE TERMINATØR
                ;NØ PØRT DEFINITIØNS
        -1      ;PØRT DEFINITIØN TABLE TERMINATØR

.END    ;END ØF FILE
```

FIGURE 4.4: TYPICAL DISC DRIVER FØR FIXED HEAD DISC

```
FD1       =20        ;DEVICE ADDRESS

          0          ;(RESERVED FØR FUTURE USE)
          0          ;(RESERVED FØR FUTURE USE)
      FD1RS-.+2      ;SIZE ØF DRIVER
          5000       ;PØWER FAIL RESTART DELAY
          1          ;"ANY ERRØR" STATUS MASK
          20         ;"WRITE PRØTECTED" MASK
          4          ;"NØ SUCH DISC" MASK
          10         ;"DATA CHANNEL LATE" MASK
          0          ;"ADDRESS CHECK ERRØR" MASK
          0          ;"ILLEGAL DISC ADDRESS" MASK
JMP       0,3        ;"INITIALIZE DRIVER" SUBRØUTINE
JMP       0,3        ;"SKIP IF LU READY" SUBRØUTINE
JMP       FD1SN      ;"SKIP IF NØT BUSY" SUBRØUTINE
JMP       FD1RS      ;"READ STATUS" SUBRØUTINE
JMP       0,3        ;"SEEK ØR RECALIBRATE" SUBRØUTINE
          10         ;NUMBER ØF SECTØRS
          100        ;NUMBER ØF TRACKS
          10         ;LØGØCAL-TØ-REAL TRACK
          1000       ;LØGICAL-TØ-REAL CYLINDER
      2000+FD1       ;ALLØC INFØ, DEVICE ADDRESS

FD1E:     DØBC       2,FD1      ;DRIVER ENTRY PØINT
          MØV        0,2
          LDA        0,PART,2;FIRST REAL CYLINDER
          ADD        0,1,SZC ;READ ØR WRITE ?
          JMP        .+3
          DØAS       1,FD1      ;  READ
          JMP        .+2
          DØAP       1,FD1      ;  WRITE
          SUBZL      1,1        ;ØNE BLØCK TRANSFERRING
          JMP        1,3        ;RETURN

FD1SN:    SKPBZ      FD1        ;SKIP IF NØT BUSY
          JMP        0,3
          JMP        1,3

FD1RS:    DIAC       0,FD1      ;READ STATUS
          JMP        0,3
```


where register A2 contains the RTA 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.

4.5 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 4.2) except that it has no interrupt handler of its own. All interrupts are handled by the multiplexer driver; therefore, the peripheral driver's INTH pointer must be -1, the attributes table must be all zeroes (three 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 RTA's TON cell (see Section 4.4). The peripheral driver's INIT routine must 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.

The character processing subroutines must not change register A2 which contains the RTA pointer. Each subroutine returns with a JMP 0, 3 as soon as possible since interrupts are disabled during this processing.

4.6 How to Write a System Subroutine Replacement

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.

4.7 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 the IRIS User Reference Manual 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
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 issues the instructions to transfer one or more blocks and returns to the location following the block count (equivalent to a JMP 1,3) with the number of blocks being transferred in register A1. The values returned in the other registers are immaterial. Note that the driver does not wait for the transfer to be completed.

The driver uses the information in its LUFIX table (Logical Unit Fixed Information) at the beginning of the driver (see Figure 4.4) and in the LUVAR table (Logical Unit Variable Information) at the location given in register A0.

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	CCYL	current cylinder
7	FUDA	first unused Real Disc Address
10	ERRC	data check error count
11		address confirmation error count
12		data channel late count

The partitioning information is used only if a Physical Unit is partitioned into two or more Logical Units; the form of the PART word is as required by the driver. AVBC is the number of disc blocks currently available (not allocated) on the Logical Unit. MINB is the minimum value of AVBC to allow building a new file. CCYL is to be used by the driver to inhibit seeking if the head is already at the desired position; if not needed for this purpose it may be used as desired by the driver, but it will be set to -1 by the system if a head position error occurs. FUDA is used by the system to determine whether the Real Disc Address supplied by the caller is too large. The three error count words are incremented by the system whenever such errors are detected.

The LUFIX table is assembled with the driver, just preceeding the driver's entry point. Its contents are at negative displacements from the entry point as follows:

Disp.	Label	Contents
-24	DINT	pointer to interrupt handler
-23	DMSK	disc controller's mask bit
-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/track)
-4	NTRK	number of tracks per cylinder
-3	LRTC	logical-to-real track conversion factor
-2	LRCC	logical-to-real cylinder conversion factor
-1	DFLG	flag word (see below)
0		driver 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. 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 at (A3)+2 when the driver's read/write routine is entered.

PFRD should be a binary integer representing the number of times around a loop consisting of a JSR to the driver's SLUR subroutine (and other instructions totaling 13.65 microseconds) that the drive may require after a power failure before it will again be ready for disc transfers. The timing for the delay loop is always calculated 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. PFRD must be zero if operator intervention is required to restart the disc after a power failure.

EMSK must contain one or more "one" bits to produce non-zero 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, and none of these 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 initializing subroutine if any initialization other than an IORST is required on initial start up or after a power failure. The CCYL cell in each LUVAR will be set to -1 by the system after the JSR to IDRV.

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. SLUR must not change register A2.

SKNB must contain a jump to a subroutine that does a skip return if the disc is ready and is not busy, or a non-skip if it is busy or not ready. SKNB must not change register A1 or A2 or use any page zero cells or constants.

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. REDS must not change register A1 or use any page zero cells or constants.

SEEK must contain a jump to a "seek or recallibrate" subroutine which will initiate a seek to the cylinder identified by the Real Disc Address given in register A1 or do a recallibrate and wait for it to be completed if (A1)= -1. SEEK must not change register A2 which contains the LUVAR pointer. Only certain moving arm 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, NTRK indicates the number of tracks per cylinder (number of heads), LRCT indicates the Logical to Real Track conversion factor, and LRCC indicates the Logical to Real Cylinder conversion factor.

FIGURE 4.5: TYPICAL DISC DRIVER FØR BZUP

```
; CAUTIØN !!   BPCF1 MUST BE FIRST WØRD ØF DRIVER
;              BRBF1 MUST BE AT BPCF1+2
;              BWBF1 MUST BE AT BPCF1+5
;              ALL CØDE MUST BE INTRINSICALLY RELØCATABLE


BPCF1:  0                 ;PARTITIØNING CØNSTANT

        DØAS    0,FD1
BRBF1:  LDA     0,.-1     ;READ A DISC BLØCK
        JMP     BWBF1+1

        DØAP    0,FD1
BWBF1:  LDA     0,.-1     ;WRITE A DISC BLØCK
        STA     0,.+4
        DØBC    2,FD1     ;ØUTPUT CØRE ADDRESS
        LDA     0,BPCF1
        ADD     1,0       ;ADD PARTITIØNING CØNSTANT
        DØA     0,FD1     ;ØUTPUT DISC ADDRESS AND START
        SUB     0,0
        INC     0,0,SNR
        JMP     0,3       ;TIME ØUT
        SKPBZ   FD1
        JMP     .-3
        DIAC    0,FD1     ;READ STATUS
        MØVR#   0,0,SZC   ;ANY ERRØR ?
        JMP     0,3       ;   YES
        JMP     1,3       ;   NØ

.DMR BZF1Ø=JMP BPCF1+BSIZE-. ; BZUP ØVERFLØW TEST
```

DFLG is a flag word made up as follows:

Bit(s)	Meaning	
15	changeable cartridge	
14-11	(unused)	
10	same sector, next track	next best block if desired block not abailable
9	next sector, next track	
8	next sector, same track	
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. Bits 6 through 10 define allocation parameters so that the file can be transferred in or out of core in the minimum time. 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 after reading one block to use information from that block for transferring the next consecutive block.

4.8 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 transfer 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.

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 a partitioning constant for converting the Real Disc Address to the corresponding Physical Disc Address. See Figure 4.5 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 currently 41 words (octal) available for the driver.

Note: the F1 used in labels in Figure 4.5 should be replaced by characters such as F3 or M5, indicating the third fixed head disc type or the fifth moving arm disc type, respectively, and FD3 or MD5 would be used as the corresponding device addresses.

5. 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.

5.1 Software Definitions Tape

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 re-assembled. The Software Definitions tape is required only on pass one of an assembly.

5.2 Page Zero Definitions Tape

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.

5.3 How to Assemble System Components

To assemble the Disc-Resident Subroutines (DISCSUBS), a peripheral driver, or any processor, feed the source tapes to the assembler in the following sequence:

Pass 1: Software Definitions
Page Zero Definitions
component source tapes

Pass 2: component source tapes

The Software Definitions and Page Zero Definitions are not necessary on pass 2 but may be included if desired for listings. BASIC, RUN and RUNMAT also require a source tape number zero (the same tape is used for all three processors) which should follow the Page Zero Definitions on pass 1 and may be included on pass 2 if a listing of it is desired.

APPENDIX 1: SYSTEM SUBROUTINES

The following subroutines are included in IRIS. Some are core-resident (in REX) while others are in the DISCSUBS file and may be made core-resident by the system manager (see Section 5.4 of the IRIS Manager Reference Manual). All are available for use by any machine code routines added to IRIS, including processors, DISCSUBS, task handlers, interrupt handlers, peripheral drivers, etc.

STORE BYTE   stores one byte at a given byte address in core.

STORE OUTPUT BYTE   stores a byte in the regnant user's I/O buffer.

MESSAGE   outputs a canned message from the MESSAGES file.

CONVERT INTEGER TO ASCII   outputs a binary number to the regnant user's I/O buffer after converting it to any radix.

TEXT MESSAGE OUTPUT   outputs the message "ERROR #_____" to the regnant user's I/O buffer.

START OUTPUT   initiates output from regnant user's I/O buffer to the user's terminal.

WAIT FOR OUTPUT NOT ACTIVE   assures that a previous output has been completed before beginning another output.

START INPUT   enables input from the regnant user's terminal into the user's I/O buffer.

ACCESS BYTE   accesses one byte from a given location in core.

ACCESS INPUT BYTE   accesses the next byte from the regnant user's I/O buffer, ignoring spaces and CTRL E codes.

ACCESS STRING BYTE   accesses the next byte from the regnant user's I/O buffer.

CONVERT DRATSAB TO ASCII   converts a string of bytes in DRATSAB code (compressed Hollerith) into the corresponding ASCII codes.

COMPARE STRINGS   tests whether two strings are equivalent.

PASSWORD COMPARE   tests whether the user supplied the correct password.

IS (A2) A DIGIT? determines whether register A2 contains an ASCII code for a decimal digit.

IS (A2) A LETTER? determines whether register A2 contains an ASCII code for a letter.

LOAD USER loads the regnant user's active file into core.

BUMP USER bumps the regnant user from core.

FAULT aborts a process due to an illegal condition or a hardware failure and prints a fault message

START IPL aborts all system operations and perform an Initial Program Load.

EXIT exits from a processor.

CHECK "BSA CHANGED" FLAG allows new information to be stored in BSA.

SEND SIGNAL sends a signal to a user on another port or to a later program segment on the same port.

RECEIVE SIGNAL receives a signal if any have been sent to the regnant user's port.

PAUSE bumps the regnant user for a specified time duration or (optionally) until a signal is sent to the user in the pause state.

SPECIAL FUNCTIONS will access certain parameters such as system time, port number, amount of time a user has used, etc.

ALLOCATE BLOCKS ON THE DISC allocates disc blocks to a file.

DEALLOCATE DISC BLOCKS deallocates blocks from a file on the disc.

CHECK CHANNEL determines whether a channel is in use.

CHECK PROTECT BITS determines whether a file or a Logical Unit is protected.

BUILD FILE creates a new file, which may replace an old file by the same Filename.

EXTEND FILE increases a file's size to greater than 128 data blocks.

DELETE FILE deletes a file.

DELETE PROCESSOR deletes a processor file.

FIND FILE finds a file or a device in an INDEX.

FIND OPEN FILE scans all channels on all ports to determine whether another user has a designated file or Logical Unit open.

OPEN opens a file or a device on a channel.

CHARGE charges a user for the use of another user's file.

MOVE moves the contents of a group of words in core to another area in core.

MOVE BYTES moves a group of bytes in core to another area in core.

WRITE DISC BLOCK writes one block (256 words) from core onto a disc.

READ DISC BLOCK reads one block (256 words) from a disc into core.

GET RECORD locates a designated record in a file and brings the data block into core.

WRITE ITEM writes an item into a file or to a peripheral device.

READ ITEM reads an item from a file or from a peripheral device.

UNLOCK RECORD unlocks a record that has been locked by a file access.

CLOSE CHANNEL closes a data channel.

CLEAR CHANNEL clears a data channel.

CLEAR ALL CHANNELS clears all channels of the regnant user's port.

ACCOUNT LOOKUP finds a user's account entry in an ACCOUNTS file via the Account I. D., account number, or entry position.

SET DECIMAL ACCUMULATOR sets the decimal accumulator (DA) to contain the floating value zero, one, or "plus infinity".

FLOAT BINARY TO DECIMAL converts a signed binary integer to floating-point decimal form.

FIX DECIMAL TO BINARY converts a floating-point decimal number to binary form.

BREAK DECIMAL NUMBER separates a floating-point decimal number into its integer and fractional parts.

DECIMAL ARITHMETIC & INPUT/OUTPUT loads or stores the decimal accumulator (DA), performs an arithmetic operation, or inputs or outputs a value in DA as an ASCII string.

ADD DECIMAL INTEGERS adds two unsigned 4-digit binary coded decimal integers.

SUBTRACT DECIMAL INTEGERS subtracts two unsigned 4-digit binary coded decimal integers.

OPEN FOR UPDATE opens a file or 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.

CONVERT LOGICAL TO REAL ADDRESS converts a logical disc address to a real disc address.

CONVERT REAL TO LOGICAL ADDRESS converts a real disc address to a logical disc address.

INCREMENT REAL DISC ADDRESS determines the n^{th} legal real disc address after a given real disc address for a given Logical Unit.

FIND LOGICAL UNIT TABLES locates the entry in the Logical Unit table and the fixed and variable information tables for a given Logical Unit number.

BINARY MULTIPLY forms the 32-bit product of two 16-bit binary integers.

BINARY DIVIDE forms the 16-bit quotient of two 16-bit binary integers and also returns the 16-bit remainder.

CONVERT RTA POINTER TO PORT NUMBER determines the number of a port from the location of its Resident Table Area.

CONVERT PORT NUMBER TO RTA POINTER determines the location of the Resident Table Area for a given port number.

APPENDIX 2: CANNED MESSAGES

The MESSAGES file contains a variety of "canned" messages that can be transferred to the regnant user's I/O buffer by a CALL MESSAGE instruction sequence with the message number (see list below) in register A1. There are no leading spaces, but a RETURN code is appended to the message. There are three possible returns from the MESSAGE subroutine as follows:

Non-skip if MESSAGES file has not been loaded on the system disc.

1-skip if there is no message assigned to the number given.

2-skip if the message has been transferred to the regnant user's I/O buffer.

The message may be appended to previous output and may be followed by more output. The currently available messages are:

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 INITAIAL 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. ARRAY TOO LARGE FOR SYSTEM
23. ARRAY SIZE EXCEEDS INITIAL DIMENSIONS
24. ONLY ONE DIMENSION ALLOWED FOR A STRING

25.
26. STRING NOT DIMENSIONED
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 ALREADY OPEN
41. BAD FILE NAME
42. NO SUCH FILE
43. FILE BEING DELETED, REPLACED, OR BUILT
44. NOT A DATA FILE (CAN'T OPEN OR REPLACE)
45. FILE IS READ PROTECTED
46. FILE IS WRITE PROTECTED
47. DISC FULL, CAN'T BUILD FILE OR ADD RECORDS
48. ACCOUNT'S DISC ALLOCATION USED UP, CAN'T BUILD FILE
49. CHANNEL NOT OPEN
50. FILE NOT FORMATTED
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 SIZES
62. SIGNAL BUFFER IS FULL
63. COMMANDS ARE ILLEGAL IN "LOAD" MODE
64. LINE NUMBER MISSING IN "LOAD" MODE

100. SOURCE FILE IS READ PROTECTED
101. SOURCE FILE IS NOT A TEXT FILE
102. SOURCE FILE IS BEING MODIFIED
103. SOURCE FILE DOESN'T EXIST
104. SOURCE FILE NAME IS ILLEGAL
105. DESTINATION FILE EXISTS IN ANOTHER ACCOUNT
106. DESTINATION FILE EXISTS AND IS BEING MODIFIED
107. DESTINATION FILE EXISTS BUT IS NOT A TEXT FILE
108. DESTINATION FILE NAME IS ILLEGAL
109. ACCOUNT IS OUT OF DISC SPACE. THE FILES ARE SAVED!
110. SYSTEM IS OUT OF DISC SPACE. THE FILES ARE SAVED!
111. READ PROTECTED FILE
112. COPY PROTECTED FILE
113. WRITE PROTECTED FILE
114. FILE BEING MODIFIED
115. ILLEGAL NAME
116. NO SUCH FILE
117. SYSTEM FILE
118. FILE BEING BUILT, REPLACED, OR DELETED