Educational Data Systems 2415 Windward Lane, Newport Beach, California 92660

INTERACTIVE REAL-TIME INFORMATION SYSTEM

(IRIS)

SYSTEM SUBROUTINE REFERENCE MANUAL

This manual contains a detailed description and calling sequence for each system subroutine listed in APPENDIX 1 of the IRIS System Reference Manual. The reader should refer to Section 2 of that manual for additional information.

EDS 1020-4
July, 1974

(IRIS) SYSTEM SUBROUTINE REFERENCE MANUAL

TABLE OF CONTENTS

4. MISCELLANEOUS

INTRODUCTION

The IRIS environment includes many subroutines which may be called by processors, tasks, or other subroutines. These subroutines have been divided into four general catagories as seen in the Table of Contents. In each catagory, some subroutines are always core-resident while others are disc-resident (in the DISCSUBS file). All subroutines called with a JSR calling sequence are always core-resident. Most disc-resident subroutines may be made core-resident if sufficient core space is available (see "How to Cause a DISCSUB to be Core Resident" in the IRIS Manager Reference Manual). The calling sequence and operation of a subroutine are not changed by making it core-resident.

Several subroutines are not described in this manual because they are either for use only by the system itself (BREAK and RECOVER), are called indirectly through another subroutine (READC, WRITC, WRITN, SHUFF, DEKEY, and RELEA), or are for a special purpose and are subject to change as required for a particular system (MTAPE, MTASK, etc).

Timing considerations are given for some subroutines. In all cases, the times given assume operation on a Nova 800 series computer. For approximate timing on a 1200 type computer, multiply the time given by 1.4. All timing given assumes that the subroutine is core-resident; if it is not, then the time required to read it from the disc must be added to the value given.

The contents of all registers and the carry flip-flop are shown for entry to and return from the subroutine. An x for an entry value means that the register is ignored and the value in the register doesn't matter. An x for a return value means that the contents of the register are undetermined. The term "unchanged" as a return value means that the register's contents are the same as at entry time. Core buffer areas (BSA, etc.) are listed along with the registers if used by the subroutine.

The subroutines are divided into three groups as follows:

- Group 1 - IRIS System Subroutines
- Group 2 - BASIC Subroutines
- Group 3 - Data File Extensions

These groups correspond to the respective item numbers on the EDS Software Price List. Group 2 and group 3 subroutines are available on a given system only if the respective software items have been licensed on the system.

Subroutine: WAIT FOR OUTPUT NOT ACTIVE (Group 1)

Calling Sequence: CALL
WONA

Use: Assures that a previous output has been completed before beginning another output.

Ac	Entry	Return
0	x	x
1	x	x
2	x	x
3	x	x
C	x	x

When an output is in progress it is illegal to disturb the user's I/O buffer or Output Byte Pointer (OBP). A call to WONA will return immediately if an output is not in progress or will bump the user if an output is in progress. In either case, the user's OBP is set equal to FBA before control is returned to the caller.

A call to WONA or STI must occur somewhere between a call to STO and any call to STOB, MSG, STO, or any other operations which might disturb the users I/O buffer or Output Byte Pointer. WONA may not be called from a disc-resident subroutine.

WONA is always core-resident.

Subroutine: STORE OUTPUT BYTE (Group 1)

Calling Sequence: JSR @.STOB

Use: Stores a byte in the regnant user's I/O buffer.

Ac	Entry	Return
0	x	byte
1	x	x
2	byte	x
3	x	return address
C	x	left/right byte flag

A2 is masked with 377 octal to clear the top half of the word and the result is copied to A0. The user's Output Byte Pointer (OBP) is incremented if and only if it is less than the Last Byte Address (LBA) of the user's I/O buffer and the byte bring stored is not a zero byte. The byte is then stored at the resulting byte address.

Refer to the writup on STORE BYTE for more information.

Subroutine: TEXT MESSAGE OUTPUT (Group 1)

Calling Sequence: JSR @.MSG
.TXTF "text"

Use: Copies a text string into the regnant user's I/O buffer.

Ac	Entry	Return (following "text")
0	x	0
1	x	x
2	x	(RUP)
3	x	address of next instruction
C	x	x

Copies the text string given into the regnant user's I/O buffer by use of the STOB subroutine, and returns to the next location following the first zero byte in the text string. Copies the zero byte at the end of the text string but leaves OBP pointing at the last non-zero byte. Therefore, a call to MSG may be followed by a call to STO or by additional calls to MSG, STOB, or CIA.

Subroutine: CANNED MESSAGE OUTPUT (Group 1)

Calling Sequence: CALL
MESSAGE

Use: Outputs canned message from the "MESSAGES" file.

Ac	Entry	Return (2-skip)
0	x	x
1	message number	x
2	x	x
3	x	x
C	x	x
BSA	x	desired message block
HBA	x	header of MESSAGES file

MESSAGE looks through the "MESSAGES" file pointer table, finds the requested message, and stuffs it into the regnant user's I/O buffer. A terminator code (zero byte) is appended to the end of the message.

There are three possible returns as follows:

Non-skip if the "MESSAGES" file does not exist on the system disc

1-skip if message number is illegal

2-skip if message found and outputted

Subroutine: ERROR NUMBER OUTPUT (Group 2)

Calling Sequence: CALL
ERROR

Use; Output the message ERROR #______ to the regnant user's I/O Buffer.

Ac	Entry	Return
0	$n*400_8+100010_8$	unchanged
1	x	x
2	x	x
3	x	x
C	x	x

Example:

```
              JSR    EROU
              2*K+NOP
              .
              .
              .
EROU:         LDA    0,0,3
              CALL
              ERROR
```

The Software Definitions tape assigns K=400 and NOP=100010 octal.

ERROR is disc-resident.

Subroutine: CONVERT INTEGER TO ASCII (Group 1)

Calling Sequence: CALL
CIA

Use: Outputs a binary number to the regnant user's I/O buffer after converting it to any radix 2 through 36.

Ac	Entry	Return
0	radix	x
1	binary integer	x
2	# digit positions	x
3	x	x
C	x	x

The value in A0 specifies the number radix into which the 16-bit binary integer in A1 is to be converted for output. Letters are used to represent digits greater than nine; i.e. A=10, B=11, C=12, . . . , Z=35.

The value in A2 specifies the minimum number of character positions for the result. For example, if (A2)=6 and the converted value of (A1) is 2 digits long, it will be preceded by 4 spaces; however, if (A2)=2 and the converted value of (A1) is 3 digits long, three digits will be printed.

CIA uses STOB to place ASCII digits and spaces in the regnant user's I/O buffer.

CIA is disc-resident.

Subroutine: START OUTPUT (Group 1)

Calling Sequence: JSR @.STO

Use: Initiates output from the regnant user's I/O buffer to the user's terminal.

Ac	Entry	Return
0	x	x
1	x	x
2	x	x
3	x	x
C	x	x

START OUTPUT copies the port's Output Byte Pointer (OBP) into its Last Output Byte pointer (LOB), then initiates output starting with the first byte in the user's I/O buffer and continuing until a zero byte is encountered. Control is immediately returned to the instruction following the JSR @.STO so that computation may continue while the output is in progress.

START INPUT may be issued while the output is in progress, but the I/O Buffer and the Output Byte Pointer must not be disturbed until the output is completed. For this reason, STORE OUTPUT BYTE (STOB), MESSAGE (MSG), etc. must not be called during an output. If another output is to follow, WAIT FOR OUTPUT NOT ACTIVE (WONA) must be called before calling STOB, MSG, MESSAGE, CIA, ERROR, or STO.

Subroutine: START INPUT (Group 1)

Calling Sequence: JSR @.STI

Use: Enables input from the regnant user's terminal into the user's I/O buffer.

Ac	Entry	Return
0	x	x
1	x	x
2	x	x
3	x	x
C	x	x

START INPUT enables input into the user's I/O buffer starting at the beginning of the buffer. Input is from the user's terminal unless some peripheral device has been selected for input.

The user is bumped from core until input is terminated normally by a RETURN code or aborted by an ESC or CTRL C code, whereupon the user is swapped back into core. In the case of a RETURN code, control is returned to the next instruction following the JSR @.STI. In the case of an ESC or CTRL C code, control is transferred to either the ESCAPE or the CTRL C processor entry, respectively.

In any case, both the user's Input Byte Pointer (IBP) and his Output Byte Pointer (OBP) will be reset to point to the beginning of his I/O buffer when he is swapped back in.

It is permissable to start input while an output is in progress, in which case input will be enabled when the output is terminated normally by accessing a zero byte.

If a non-active character is typed by the user at a time when input is not enabled, this character is placed in the user's Temporary Input Buffer (TIB) until such time as input is enabled, whereupon it will become the first input character and will then be echoed if echo is not disabled. Only the last character typed while input is not enabled will be retained in TIB.

Refer to BUMP for more information on swapping.

Subroutine: ACCESS INPUT BYTE (Group 1)

Calling Sequence: JSR @. ACIB

Use: Accesses the next byte from the regnant user's I/O buffer, ignoring spaces.

Ac	Entry	Return
0	x	x
1	x	x
2	x	byte (not a space)
3	x	return address
C	x	left/right byte flag

Accesses the next byte from the regnant user's I/O buffer. If the byte is a space (octal 240) it is ignored, and the next byte is accessed until any non-space byte is obtained. A RETURN code (octal 215) is the final input byte.

Subroutine: ACCESS STRING BYTE (Group 1)

Calling Sequence: JSR @.ACSB

Use: Accesses the next byte from the regnant user's I/O buffer.

Ac	Entry	Return
0	(see text)	x
1	x	x
2	x	byte
3	x	return address
C	x	left/right byte flag

Accesses the next byte from the regnant user's I/O buffer. A RETURN code is the final input byte.

No bytes are ignored. If register A0 is zero then the Input Byte Pointer is not incremented; i.e., the next subsequent ACSB or ACIB will access the same byte again. Otherwise, calls to ACSB and ACIB may be intermixed to access sequential input bytes until a RETURN code (octal 215) is encountered.

Subroutine: QUEUE CHARACTER FOR PROCESSING (Group 1)

Calling Sequence: QCHARACTER

Use: Queues input characters and output character requests for processing by the system.

Ac	Entry	Return
0	see text	see text
1	x	see text
2	RTA pointer	unchanged
3	x	return address
C	x	x

If $(A0) < 0$ this is a request for an output character. The character in TOB is returned in A0, and TOB is zeroed. If TOB was non-zero then a request for another output character is put on the queue. The top bit of (A0) will be a "one" to indicate the presence of a character in the lower byte; zero in A0 indicates end of output. In either case, A1 will be zero on return.

If $0 \leq (A0) < 400_8$ then it is an input character which is put on the queue to be processed. On return, register A0 is unchanged and (A1) = (A2)

If $(A0) > 400_8$ then it is an interrupt task which is being queued. Values in the range 400 to 437 octal select a task by number. Any value greater than 437 is taken as the absolute core address of the entry point to the task. On return from QCHARACTER, register A0 is unchanged and (A1) = (A2) in either case.

Interrupts must be disabled when QCHARACTER is used. Timing: 21.7 µS typical, 31.6 µS maximum on a Nova 800. QCHARACTER is always core-resident.

Subroutine: DECIMAL ARITHMETIC & INPUT/OUTPUT (Group 2)

Calling Sequence: JSR @.DEC

Use: Loads or stores the decimal accumulator (DA), performs an arithmetic operation, or inputs or outputs a value in DA as an ASCII string.

Ac	Entry	Return
0	function code	x
1	number type or buffer pointer	(see text)
2	argument pointer	(see text)
3	x	x
C	x	x

Register A0 must contain a value from zero to 11 (octal) to specify one of the following functions:

A0	Function	Remarks
0	store	(DA)→(A2)
1	load	((A2))→DA
2	subtract	(DA)-((A2))→DA *
3	add	(DA)+((A2))→DA
4	divide	(DA)÷((A2))→DA *
5	multiply	(DA)x((A2))→DA
6	input	uses byte address in A1 or ACIB if (A1)=0
7	output	uses byte address in A1 or STOB if (A1)=0
10	output	same as 7 except no leading space for +
11	output	same as 7 except formatted by string at (A2)

For functions 0 through 5 an argument pointer must be supplied in register A2, and a number type must be specified by a value in register A1 as follows:

A1	Number Type	Remarks
0	unsigned integer	range 0 to 9999
1	signed integer	range ±7999
2	2-word floating	six digit mantissa
3	3-word floating	ten digit mantissa
4	4-word floating	14 digit mantissa
5	6-word unpacked	15 digit mantissa

* Note: add 10 (octal) to the number type in A1 for inverse subtract or divide; i.e., ((A2)) -(DA)→DA or ((A2)÷(DA)→DA, respectively.

(continued)

See below for use of registers A1 and A2 by functions 6 through 11. All returns are non-skip except as indicated in the following detailed descriptions:

0	Store	Packs the value in DA into the form specified in A1 and stores at (A2). If the value is too large to store in the specified form then the error flag (ERRF) is set and the largest possible value is stored. The value in DA is not changed.
1	Load	Unpacks the argument at (A2), which is assumed to be of the form specified in A1, and normalizes the result in DA. The argument is not changed.
2 3 4 5	Subtract Add Divide Multiply	Unpacks the argument into DB and then performs the specified arithmetic operation. In case of overflow (such as division by zero) the error flag is set. Result is Normalized and returned in DA.
6	Input	Register A2 is ignored. If (A1) is zero then the Access Input Byte subroutine (ACIB) is used to scan an input string in the regnant user's I/O buffer. If (A1) is non-zero then it is used as the starting byte address of the input string. Clears DA to zero and does a non-skip return if no digits are found; otherwise does a skip return with the converted value in DA and the number of significant digits scanned in A0. The conversion routine will accept ASCII codes for an optional leading plus or minus sign and any number of decimal digits with one optional imbedded period (decimal point). A trailing exponent in the form $E\pm dd$ will also be accepted. In any case, the first character which is not converted will be returned in register A2. If a byte address was supplied in A1 then the byte address of the terminating character will be returned in A1.
7	Output	If (A1) is zero then STOB is used to Store Output Bytes as follows: either a space if (DA) is positive or a minus sign if (DA) is negative, a string of up to 14 digits, and an imbedded period if required. If (A1) is non-zero, then the output string is stored starting at the byte address in A1, and the next byte address is returned in A1. Defaults to floating form if the value in DA is outside the range $10^{-1} \leq (DA) < 10^{14}$.
10	Output	Same as 7 above except no leading space if (DA) is positive.
11	Formatted Output	Same as 10 above except register A2 must contain the byte address of a format string. The value in DA is outputted in the format specified by that string as described in the PRINT USING statement in the EDS Business BASIC Programming Manual. Returns byte address of first unused format character in A2. Skip return is normal;non-skip return if an error is detected in the format string.

Subroutine: ADD DECIMAL INTEGERS (Group 2)

Calling Sequence: LDA 3,.DEC
JSR @ -2,3

Use: Adds two unsigned 4-digit Binary Coded Decimal integers.

Ac	Entry	Return
0	augend	sum
1	addend	x
2	x	carry out (in MSB)
3	x	x
C	x	x

Registers A0 and A1 must each contain a 4-digit BCD integer, the sum of which will be returned in A0, also in BCD. If the sum exceeds 9999 then the most significant bit of A2 will be a one, and A0 will contain the excess over 10000; otherwise, the most significant bit of A2 will be zero. The remaining 15 bits of A2 are undetermined in either case. For multiple precision addition, the carry may be propagated by incrementing the addend in A1 (the least significant BCD digit of the addend may be ten).

Subroutine: SUBTRACT DECIMAL INTEGERS (Group 2)

Calling Sequence: LDA 3,.DEC
JSR @-1,3

Use: Subtracts two unsigned 4-digit Binary Coded Decimal integers.

Ac	Entry	Return
0	subtrahend	difference
1	minuend	x
2	x	borrow out (in MSB)
3	x	x
C	x	x

Registers A0 and A1 must each contain a 4-digit BCD integer, the difference of which will be returned in A0, also in BCD. If (A1) exceeds (A0) then the most significant bit of A2 will be a one and A0 will contain the value (A0)+10000-(A1); otherwise, the most significant bit of A2 will be zero. The remaining 15 bits of A2 are undtermined in any case. For multiple precision subtraction, the borrow may be propagated by incrementing the minuend in A1 (the least significant BCD digit of the minuend may be ten).

Subroutine: SET OR GET SIGN OF DA (Group 2)

Calling Sequence: JSR @.DEC

Use: Sets or retrieves the sign bit of the decimal accumulator.

Ac	Entry	Return
0	see text	x
1	see text	new sign of DA
2	x	x
3	x	x
C	x	x

The mantissa and characteristic of the value in DA are unchanged. Only the sign bit is affected as determined by the value in A0 as follows:

(A0) octal	Effect
12	set sign of mantissa = $(A1)_0$
13	get sign of mantissa

If $(A0) = 12_8$ then the least significant bit of A1 must contain either zero or one to set the sign of the mantissa either positive or negative, respectively. A1 is ignored if (A0) = 13.

Subroutine: WRAPUP DECIMAL OPERATION (Group 2)

Calling Sequence: JSR @.DEC

Use: Ensures completion of last decimal store.

Ac	Entry	Return
0	14_8	x
1	x	x
2	x	x
3	x	x
C	x	x

May be called after other decimal operations to ensure that the last store into core by the Decimal Arithmetic Unit has been completed before computation is resumed.

Subroutine: SET VALUE IN DA (Group 2)

Calling Sequence: JSR @.DEC

Use: Sets the decimal accumulator (DA) to contain the floating value zero, one, or "plus infinity".

Ac	Entry	Return
0	see text	0
1	x	(DAC)
2	x	unchanged
3	x	return address
C	x	x

The value set into DA is determined by (A0) as follows:

(A0) octal	Value set into DA
15	Zero
16	Plus one
17	"plus infinity" = 0.999999×10^{63}

Note: the error flag (ERRF) is also set when "plus infinity" is set into the accumulator.

Subroutine: BREAK DECIMAL NUMBER (Group 2)

Calling Sequence: JSR @.DEC

Use: Separates a floating-point decimal number into its integer and fractional parts.

Ac	Entry	Return
0	20_8	x
1	x	x
2	x	x
3	x	x
C	x	x

The integer portion of the decimal floating-point number in the decimal accumulator (DA) is copied into the decimal buffer register (DB). The fractional portion is left in DA and is normalized.

Subroutine: FIX DECIMAL TO BINARY (Group 2)

Calling Sequence: JSR @.FIX

Use: Converts a floating-point decimal number to binary form.

Ac	Entry	Return (skip)
0	x	sign (0 for +, 1 for -)
1	x	binary integer
2	x	x
3	x	x
C	x	x

FIX ignores any fractional portion of the floating-point decimal value in the decimal accumulator (DA) and converts the integer portion to a 16-bit binary integer with sign. There are two possible returns as follows:

Non-skip if (DA) is outside the range of a one-word binary integer; i.e., -177777 to +177777 octal or -65535 to +65535 decimal

Skip if (DA) is within the range of a one-word binary integer plus sign; the value returned in register A1 will be a 16-bit positive integer, and (A0) represents the sign (0 for positive or 1 for negative)

Subroutine: FLOAT BINARY TO DECIMAL (Group 2)

Calling Sequence: JSR @.FLOT

Use: Converts a signed binary integer to floating-point decimal form.

Ac	Entry	Return
0	sign	x
1	binary integer	x
2	x	x
3	x	x
C	x	x

The value given in A0 must be zero if the value in A1 is positive, or 1 if the value in A1 is negative. In either case, the value in A1 is taken as an unsigned 16-bit binary integer. FLOAT leaves the equivalent value in the decimal accumulator (DA) as a normalized floating point decimal number.

Subroutine: STORE DECIMAL ACCUMULATOR (Group 2)

Calling Sequence: JSR @.STDA

Use: Stores the contents of the Decimal Arithmetic Unit (DAU) accumulator into core in DA, DAS, and DAC.

Ac	Entry	Return
0	x	(DA first word)
1	x	(DAS)
2	x	unchanged
3	x	x
C	x	x

This call must be made prior to an actual manipulation of DA, DAS, or DAC by any processor, whether a DAU is actually installed or not. This call assures that the memory result and DAU result of an arithmetic operation will be identical. DA and DAS are loaded into A0 and A1 as a convenience to the user.

Subroutine: LOAD DECIMAL ACCUMULATOR (Group 2)

Calling Sequence: JSR @. LODA

Use: Loads the contents of DA, DAS, and DAC from core memory into the Decimal Arithmetic Unit (DAU) accumulator.

Ac	Entry	Return
0	x	unchanged
1	x	unchanged
2	x	unchanged
3	x	x
C	x	x

This call must be made to load the memory contents of DA, DAS, and DAC into the Decimal Arithmetic Unit (DAU) if the processor has modified the value in DA, DAS, or DAC. This call assures that any modified values of DA, DAS, or DAC in core can also be loaded into the DAU.

Subroutine: SPECIAL FUNCTIONS (Group 2)

Calling Sequence: CALL
SPECIAL

Use: Obtains special information about the system or the regnant user.

Ac	Entry	Return (skip)
0	0	x
1	desired function	function value
2	a(SLT)	x
3	x	x
C	x	x

The value in register A1 determines the information to be returned in the decimal accumulator (DA) and in register A1 as follows:

A1	Function	
0	CPU time used in tenth-seconds	
1	Connect time used in minutes	
2	Hours since 1-1-73	REAL TIME
3	Part of hour (in tenth-seconds)	
4	System Creation date (Hours after 1-1-73)	
5	Account number of regnant user	
6	Port number of regnant user	
7	Value set on console switches	
10	Last BASIC error type number	
11	Current BASIC line number	

If bit 15 of (A1) is one, then bits 0 through 14 are taken as an absolute core address, and the contents of that cell are returned as the function value.

There are two possible returns as follows:

Non-skip if an illegal function number in A1

Skip if function completed; the function value returns in DA as an unpacked floating-point BCD number and in register A1 as a 16-bit binary integer

Subroutines: TRANCENDENTAL FUNCTIONS (Group 2)

Calling Sequence:

```
JSR     @.STDA
CALL
Function
```

where Function is one of the following:

PSQRF	-	square root function
PLOGF	-	natural log function
PEXPF	-	exponential function
PSINF	-	sine function
PCOSF	-	cosine function
PTANF	-	tangent function
PATNF	-	arctangent function

Use: Calculates the specified function of a given argument value.

Ac	Entry	Return
0	x	x
1	x	x
2	x	x
3	x	x
C	x	x

Both DA in memory and the DAU will have the result of the selected function. If an error occurs then the result will be one of the following:

Function	Argument	Result	Error Flag Set
SQR	<0	$\sqrt{\lvert arg \rvert}$	yes
LOG	$\leq 10^{-16}$	-Infinity	if arguement ≤ 0
EXP	>148	+Infinity	yes
	<-148	Zero	no

The trigonometric functions (SIN, COS, and TAN) will never get an underflow or overflow error because the initial argument is reduced to the range 0 to 2π. All argument values are legal for the Arctangent function.

Subroutine: BUILD FILE or BUILD DEVICE FILE (Group 1)

Calling Sequence: CHANNEL BUILD or CHANNEL BILDD

Use: Creates a new file, which may replace an old file by the same Filename.

Ac	Entry	Return (7-skip)
0	channel number	d(file header)
1	B(Filename string)	B(terminator)
2	a(header information)	a(channel)
3	x	x
C	x	x
BSA	x	x
HBA	x	x

BUILD creates a new file or replaces an old one. The old file is replaced if and only if the Filename is followed by an exclamation mark "!". The old file is marked as being replaced and is deleted when the new file is closed or, if the old file is open an another channel, it will be deleted later when no one is using it. A file that is being built or replaced cannot be opened. The new file cannot be opened until CLOSE is called on the channel where it is being built. If a format map is supplied, a FAULT will occur if the map has zero or more than 64 item entries.

(A2) upon entry must point to a table of header information as follows:

Word #0	TYPE	file type
Word #1	NBLK	number of blocks ($1 \leq$ NBLK ≤ 129)
Word #2	STAD	starting address
Word #3	COST	cost in dimes (BCD)
Word #4	UNIT	Logical Unit number
Word #5	.FMAP	pointer to a format map

If TYPE=37 a formatted data file will be built unless the Filename string indicates a contiguous data file.

If the FMAP pointer given in word #5 is zero then there is no format map. The FMAP pointer may be indirect. The FMAP pointer word must be zero for all data files except type 31 (formatted). The selected Logical Unit must be on the system or a FAULT will occur; if the UNIT number given in word #4 is -1, BUILD uses the regnant user's Logical Unit (via RUP). If the user supplies a Logical Unit as a part of the Filename, e.g. 7/Fllename, it will supersede the Logical Unit number given in word #4.

(continued)

BUILD scans the Filename string for three parameters: protection, cost, and contiguous file size. These parameters may be given in any order providing they precede the Filename. The general form of the string is as follows:

<pp> $ddd.cc [n:b] lu/Filename !

This allows the caller to grant access to his file by other users at his privilege level and lower, to charge all others for its use, to create a contiguous file, to specify a Logical Unit other than the one assigned to his account, and to specify that a file on his account by the same name may be replaced. In this case pp represents a two digit number specifying the desired protection. The first digit gives protection against users at lower privilege levels, and the second digit gives protection against users on other accounts at your own privilege level. It is not possible to protect against higher privilege users. A file may be protected against other users of the same account only by use of a password. Each digit indicates protection as follows:

p	protection
0	None
1	Copy protect. Prohibits others from listing the file or saving it under a different Filename or on a different Logical Unit.
2	Write protect. Prohibits others from deleting the file or writing data into it.
4	Read protect. Prohibits others from using the file or reading data from it.

The types of protection may be combined by adding the values given for each desired type (see example below). The protection given in the TYPE (information word #0) will be used if and only if the protection is not specified in the Filename string.

The dollar sign indicates that the amount ddd.cc (dollars and cents) is to be charged to the account of any other user who gains access to the file. The cost given will be truncated to the nearest ten cents. The cost specified in information word #3 will be used if and only if the cost is not specified in the Filename string.

The "[" indicates that this file is to be a contiguous file containing n records of b bytes per record. The values of n and b must be in the range $1 \leq x \leq 65534$. A contiguous file will be built if and only if a type of 37 was selected in the header's information table. Caution: a space must precede the Filename or lu/Filename.

(continued)

There are eight possible returns as follows:

Non-skip if illegal channel number
A0=unchanged
BSA and HBA are unchanged

1-skip if channel in use
A0=unchanged
A1=FDA of channel
A2=a (channel)
BSA and HBA are unchanged

2-skip if illegal Filename, protection, cost, or syntax, or inactive Logical Unit
A0=same as (A3) for non-skip return from FFILE

3-skip if old file is type zero or types don't match
A0=TYPE of old file
A1=B(terminator)
BSA=an INDEX block
HBA=INDEX header

4-skip if old file is being built or replaced
A0=STAT of old file
A1=B(terminator)
BSA=unknown
HBA=file header

5-skip if an old file can't be replaced
BSA=unknown
HBA=file header

6-skip if disc or account is full
A0=# of blocks the disc needs if positive, or
A0= -# of blocks the account needs if negative
BSA=unknown
HBA=file header

7-skip if file is successfully built; registers and buffer areas as shown in table

Subroutine: ALLOCATE DISC BLOCKS (Group 1)

Calling Sequence: CALL
ALLOCATE

Use: Allocates disc blocks to a file.

Ac	Entry	Return
0	number of blocks	see text
1	x	x
2	pointer into header	x
3	x	x
C	x	x
BSA	x	x
HBA	file header	file header

(A0) must be the desired number of blocks to add to the file; e.g., if NBLK of the file equaled two and a user wanted to allocate a third block to that file, (A0) should contain one.

(A2) must be a core address in HBA or HXA. ALLOC will allocate (A0) blocks sequentially starting at (A2).

ALLOC will non-skip return if it cannot allocate the desired number of blocks. (A0) will contain the number needed (positive if the disc was full, or negative if the user's account was full) or (A0) will be zero if the Logical Unit is not active.

ALLOC will optimize its selection of disc addresses based on the allocation information bit of the DFLG cell of the appropriate LUFIX table.

Subroutine: EXTEND FILE (Group 3)

Calling Sequence: CALL
EXTEND

Use: Increases a file's size to greater than 128 data blocks.

Ac	Entry	Return (2-skip)
0	x	Logical Unit number of header
1	x	d(extension block)
2	x	a(HXA)
3	x	x
C	x	x
HBA	file header	extended file header
HXA	x	header extension block

EXTEND allocates a header extension block to a file and moves all disc addresses of the data blocks into the new block. The disc address of the extension block is put into the header as the only entry, and bit zero of the file's STAT word is set by EXTEND after the file has been extended. Each of the disc addresses of an extended header points to a disc block holding up to 256 words of data.

EXTEND will FAULT if the header in HBA is already extended.

There are two possible returns as follows:

Non-skip if disc or account is full
A0=number of blocks needed (positive if disc full, or negative if account was full)

Skip if the file extended; registers as shown

Subroutine: ALLOCATE A CONTIGUOUS FILE (Group 3)

Calling Sequence: CALL
ALCONTIG

Use: Allocates sequential disc blocks on the specified Logical Unit for a contiguous data file.

Ac	Entry	Return (skip)
0	# of blocks desired	Logical Unit number
1	x	x
2	x	a (BSA)
3	x	x
C	x	x
BSA	x	x
HBA	file header	file header
HXA	x	x

ALCONTIG searches through the disc map until it finds (A0) contiguous (physically sequential) disc blocks available on the specified Logical Unit. Only the disc address of the header is stored in the file header.

There are two returns as follows:

Non-skip if not enough contiguous space or account is full
$(A0) \geq 0 \Rightarrow$ not enough contiguous space
$(A0) < 0 \Rightarrow$ account full

Skip if successful; registers as shown in table

Subroutine: DEALLOCATE DISC BLOCKS (Group 1)

Calling Sequence: CALL
DALLC

Use: Deallocates disc blocks from a file.

Ac	Entry	Return
0	# of blocks to be in file	x
1	x	x
2	pointer into header	x
3	x	x
C	x	x
BSA	x	x
HBA	file header	file header

DALLC will deallocate disc blocks until (A0) blocks remain in the file. NBLK must be greater than (A0). If (A0) is non-zero, blocks are deallocated starting at (A2) and working toward the beginning of the file until (A0) blocks remain. (A2) is ignored if (A0) is zero. ALLOC and DALLC are the only map manipulating routines, and any alterations to the map should be made via them. The file owner's account is credited for the freed blocks.

Subroutine: DELETE FILE (Group 1)

Calling Sequence : CALL
DELETE

Use: Deletes a file.

Ac	Entry	Return(4 & 5-skip)
0	Logical Unit number	x
1	x	B(terminator)
2	B(Filename)	x
3	x	x
C	x	x
BSA	x	x
HBA	x	file header

If (A0) =-1, the regnant user's Logical Unit will be assumed. The Filename at (A2) may not be in HBA.

DELETE removes the Filename from the INDEX. If the file's income is non-zero then it is subtracted from the accrued charges in the file owner's account. If the file is not open on any port, the blocks of the file are immediately deallocated, and the original owner of the file is credited for the blocks. If the file is open, then the Filename is removed from the INDEX, and the file is marked to be deleted (bit 13 of the STAT word). CLEAR or CLOSE will deallocate the blocks and credit the original owner when the file is no longer in use.

There are six possible returns as follows:

Non-skip if illegal name
A1=byte address of terminator
BSA=unchanged
HBA=INDEX header

1-skip if not found
A0=disc address of an INDEX block which is in BSA
A1=byte address of terminator
A3=core address of empty INDEX entry in block in BSA
BSA=INDEX block
HBA=INDEX header

(continued)

2-skip if file was a processor, a driver, or type 0
A0=file type
Al=B(terminator)
BSA=INDEX block
HBA= file header

3- skip if file was write protected
Al=byte address of terminator
BSA=INDEX block
HBA= file header

4-skip if file was deleted but is being replaced; registers as shown in table

5-skip if file was deleted; registers as shown in table

Subroutine: DELETE PROCESSOR (Group 1)

Calling Sequence: CALL
PDELETE

Use: Deletes a processor file or a driver file.

Ac	Entry	Return (4 & 5-skip)
0	Logical Unit number	x
1	x	B(terminator)
2	B(Filename)	x
3	x	x
C	x	x
BSA	x	x
HBA	x	file header

If (A0) =-1 the regnant user's Logical Unit will be assumed.

The Filename at (A2) may not be in HBA.

DELETE PROCESSOR is the same in all respects as DELETE FILE except that the 2-skip return will occur only for a type 0 file; i.e., a processor or a driver may be deleted.

Subroutine: FIND FILE (Group 1)

Calling Sequence: CALL
FFILE

Use: Finds a file or a device in an INDEX.

Ac	Entry	Return (skip)
0	Logical Unit number	d(INDEX block in BSA)
1	x	B(terminator)
2	B(Filename)	Logical Unit number
3	x	a(INDEX entry)
C	x	x
BSA	x	INDEX block
HBA	x	H(INDEX)

If the Filename supplied is of the form number/Filename then (A0) is ignored and only the Logical Unit given by the number in the Filename will be searched. Otherwise, only the Logical Unit given by (A0) will be searched.

FFILE searches the INDEX on the selected Logical Unit, comparing the Filename given at B(A2) with each INDEX entry. There are two possible returns as follows:

Non-skip return if file not found. In this case, (A3) indicates the reason as follows:

(A3)=0 if illegal Filename
(A2)=Logical Unit number

(A3)=1 if Logical Unit not active
(A0)=a(Logical Unit table entry)
(A2)=Logical Unit number

(A3)=2 if file not found, INDEX is full, and not enough room on Logical Unit to add a block to the INDEX
(A1)=byte address of terminator
(A2)=Logical Unit number

(continued)

(A3) ≥ BSA if Filename is legal and Logical Unit is active, but file not in INDEX; FFILE will allocate a block to the INDEX if necessary

(A0) = d(INDEX block in BSA)
(A1) = byte address of terminator
(A2) = Logical Unit number
(A3) = a(empty INDEX entry)
(BSA) = an INDEX block
(HBA) = INDEX header

Skip return if file is found; registers and buffers as shown in table

Subroutine: FIND OPEN FILE, INITIALIZE (Group 1)

Calling Sequence: CALL
FOFI

Use: Initializes a search for an open file or Logical Unit.

Ac	Entry	Return
0	Logical Unit number	number of ports
1	Real Disc Address (see text)	unchanged
2	x	a(first RTA)
3	x	FOFC entry address
C	x	1

FOFI initializes pointers and counters for FOFC. Specifically, FOFI sets up FOFC to start looking at the data channel number minus four of port number zero, and it sets a counter to total number of active ports. If FOFI is called with zero in A1 then FOFC will check for any file open on the specified Logical Unit.

FOFI and FOFC are both core-resident.

Subroutine: FIND OPEN FILE, CONTINUE (Group 1)

Calling Sequence: CALL FOFC or JSR 0, 3

Use: Determines whether a file or Logical Unit is open.

Ac	Entry	Return (skip)
0	x	a (RTA) of port where open
1	x	x
2	x	a (DFT entry where open) - CHM4
3	x	FOFC entry address
C	x	x

FOFI must be called to identify a file before FOFC is called. The second calling sequence (JSR 0, 3) may be used only if A3 is unchanged since the last call to FOFI or FOFC. FOFC will scan the Data File Table of each port to determine whether the file is open by any user. If (A1) was zero when FOFI was called, then FOFC will look for any file open on the specified Logical Unit. There are two possible returns as follows:

Non-skip if no open file is found
A1=number of data channels per port
A2=a (RTA of last port)
A3=return address

Skip if the file (or Logical Unit) is found to be open; registers as shown in table

After a skip return, FOFC may be called again to determine whether the file (or Logical Unit) is also open on another channel. Calls to FOFC may be repeated without calling FOFI until FOFC does a non-skip return.

Subroutine: OPEN (Group 1)

Calling Sequence: CHANNEL
OPEN

Use: Opens a file or a device on a channel.

Ac	Entry	Return(8-skip)
0	channel number	B(terminator)
1	B(Filename)	d(file header)
2	a(Control Block)	a(channel)
3	x	x
C	x	x
BSA	x	x
HBA	x	x

where the Control Block is as follows:
word #0: desired file type, or -1 for any type
word #1: Logical Unit number, or -1 if the regnant user's Logical Unit is to be used

OPEN opens a file on channel #(A0) providing there are no restrictions; e.g., user call errors, protection, etc. Only runnable processors may be opened on channel -1. A default file type -1 is allowed to open any file whether or not it is a data file. A default file type 37 will allow any file of type 30 through 36 to be opened. Opening a type 36 file (peripheral driver) will cause a JSR to the driver's INIT routine. If a file is write protected but not read protected it will be opened, but the write locked status of that channel is set. If the file is opened then CHARGE is called to charge the user for access to the file.

There are nine possible returns as follows:

Non-skip if illegal channel number
BSA and HBA are unchanged

1-skip if channel in use
A0=unchanged
A1=FDA of channel
A2=a(channel)
BSA and HBA are unchanged

(continued)

2-skip if illegal Filename, inactive Logical Unit, or INDEX is full and not enough room on Logical Unit to add block to INDEX

A1=B(terminator)
(A3)=0 if illegal Filename
(A3)=1 if Logical Unit inactive
(A3)=2 if INDEX full
BSA is unchanged
HBA=INDEX header

3-skip if no such file

A0=d(an INDEX block)
A1=B(terminator)
A3=a(empty INDEX entry)
BSA=the INDEX header
HBA=INDEX header

4-skip if file is being built or replaced

A0=status word of file
A1=B(terminator)
BSA= the INDEX block
HBA=file header

5-skip if wrong TYPE, or channel= -1 and file is not runnable

A0=file's type if channel -1 was selected, or
A0=requested type if channel≥0
A1=B(terminator)
BSA=the INDEX block
HBA=file header

6-skip if file was read protected

A0=user's privilege level
A1=B(terminator)
BSA=the INDEX block
HBA=file header

7-skip will not occur (this is a return for OPENUPDATE or OPENLOCK)

8-skip if file is successfully opened; registers as shown in table

Subroutines: OPEN FOR UPDATE (Group 1)
OPEN FOR REFERENCE
OPEN AND LOCK

Calling Sequence: CHANNEL OPENUPDATE or CHANNEL OPENREF or CHANNEL OPENLOCK

Use: Opens a file or a device for a special purpose.

Ac	Entry	Return (8-skip)
0	channel number	B(terminator)
1	B(Filename)	d(file header)
2	a(Input Block)	a(channel)
3	x	x
C	x	x
BSA	x	x
HBA	x	x

where the Input Block is as follows:

word #0: desired file type, or -1 for any type
word #1: Logical Unit number, or -1 if the regnant user's Logical Unit is to be used

If opening a peripheral driver, OPENLOCK will do a JSR to the driver's INIT routine, but OPENREF and OPENUPDATE will not.

OPENREF does not change the Last Accessed Date in the file's header or charge the user for access to the file, but it unconditionally sets the write locked status of the channel.

There are nine possible returns from these routines. All are the same as for OPEN with the following exceptions:

7-skip return from OPENUPDATE if the file or device is write protected
A1=B(terminator)
BSA=the INDEX block
HBA=file header

7-skip return from OPENLOCK if the file is already open elsewhere or is write protected
A0=-1
A1=B(terminator)
BSA=the INDEX block
HBA=file header

Subroutine: CHECK PROTECTION (Group 1)

Calling Sequence:	CALL CHKRP	or	CALL CHKWP	or	CALL CHKCP

Use: Determines whether a file is protected

Ac	Entry	Return
0	ACNT word from file	x
1	TYPE word from file	x
2	x	x
3	x	return address
C	x	x

Call CHKRP to check read protection, CHKWP to check write protection, or CHKCP to check copy protection. Access is granted if:

a) file's account number is same as user's account number, or

b) user is privilege level three or has a privilege level higher than the file, or

c) the protection specified in the file does not prohibit the type of access requested by this user.

There are two possible returns as follows:

Non-skip if file is protected

Skip if access is granted

Subroutine: CHARGE FOR FILE ACCESS (Group 1)

Calling Sequence: CALL
CHARGE

Use: Charges a user for access to another user's file.

Ac	Entry	Return
0	Logical Unit number	x
1	d(file header)	x
2	a(buffer)	x
3	x	x
C	x	x

BSA and HBA are unaffected depending on (A2) entry.

(A2) entry must be a 256_{10} word buffer to be used by CHARGE.

CHARGE updates the Last Accessed Date (LDAT) cells and increments the Number of Times Accessed (NTAC) cell in a file's header. If the user is on a different account than the file and there is a non-zero cost for the file, then the cost is added to the "total charges" (CHGS) cells in the file's header and also added to the "net accrued charges" cells in the user's entry in the ACCOUNTS file.

Return is non-skip.

Subroutine: CLOSE CHANNEL (Group 1)

Calling Sequence: CHANNEL
CLOSE

Use: Closes a file or device which is open on a channel.

Ac	Entry	Return (2-skip)
0	channel number	x
1	x	x
2	x	x
3	x	x
C	x	x
BSA	x	x
HBA	x	x

CLOSE closes the file open on channel #(A0). If the file's delete bit is set and the file is not open elsewhere then the file is deleted. If the file's build bit (bit 15) is set, that bit is reset, and if an old file was being replaced then the new file replaces the old one in the INDEX, and the old file's blocks are deallocated unless the old file is open elsewhere.

There are three possible returns as follows:

Non-skip if illegal channel number
A0=unchanged
BSA=unchanged
HBA=unchanged

1-skip if channel not in use
A0=unchanged
A2=a(channel)
BSA=unchanged
HBA=unchanged

2-skip if channel closed; registers as shown in table

Subroutine: CHECK CHANNEL (Group 1)

Calling Sequence: CALL
CHKCHANNEL

Use: Determines whether a channel is in use.

Ac	Entry	Return
0	channel number	unchanged
1	x	see text
2	x	see text
3	x	return address
C	x	x

Examines the regnant user's channel #(A0).

There are three possible returns as follows:

Non-skip if illegal channel number
A1=number of channels available

1-skip if channel not in use
A1=0
A2=a(channel)

2-skip if channel is in use
A1=d(file header) (bit 15 may be set)
A2=a(channel)

Subroutine: CLEAR CHANNEL (Group 1)

Calling Sequence: CHANNEL
CLEAR

Use: Clears a channel.

Ac	Entry	Return(skip)
0	channel number	x
1	x	x
2	x	x
3	x	x
C	x	x
BSA	x	x
HBA	x	x

CLEAR CHANNEL clears the channel #(A0) of the regnant user's port. If the file open on (A0) is marked to be deleted and is not open elsewhere then CLEAR deallocates the file's disc blocks. If a file is marked as being built then CLEAR deallocates the file's disc blocks; also, if an older file was being replaced, then CLEAR resets the replace bit of the old file.

There are two possible returns as follows:

Non-skip if illegal channel number
A0=unchanged
A2=a(channel)
BSA=unchanged
HBA=unchanged

Skip if channel cleared; registers as shown in table

Subroutine: CLEAR ALL CHANNELS (Group 1)

Calling Sequence: CALL
ALLCLEAR

Use: Clears all channels of the regnant user's port.

Ac	Entry	Return
0	x	x
1	x	x
2	x	x
3	x	x
C	x	x
BSA	x	x
HBA	x	x

ALLCLEAR uses CLEAR to clear all of the regnant user's data channels (all channels with non-negative numbers).

Return is non-skip.

Subroutine: GET RECORD, WRITE (Group 1)

Calling Sequence: CALL
GETRW

Use: Locates and reads a selected record of a file for writing data.

Ac	Entry	Return (2-skip)
0	x	x
1	record number (see text)	d(data block)
2	a(channel)	B(record)
3	x	x
C	x	x
BSA	x	data block
HBA	x	File header
EXA	x	extender if file is extended

GET RECORD, WRITE uses the STS word of the channel to determine if the open file is formatted, unformatted, or contiguous. From this information, GETRW determines the disc address of the appropriate data block and reads that block into BSA. GETRW then determines the location in that block of the desired record and generates a byte pointer to it.

If a -1 is supplied in A1 as the record number, GETRW will look up the next record by using information in the channel. For formatted files, the record number will be (FSZ)+1. For contiguous files, the record address will be (CBN)*400(octal)+$_8$(STS)$_0$+2*(WPR). For text or unformatted files, the record address will be (CBN)*400(octal)+$_8$(STS)$_0$.

If a -2 is supplied in A1 as the record number, GETRW will look up the present record. For formatted files, the record number is (FSZ). For all others, the record address will be (CBN)*400(octal)+$_8$(STS)$_0$. Note that for a text file this is the same as for -1 in A1.

If the block containing the record requested does not exist, GETRW will allocate the proper block. GETRW will then write all zeroes into that block.

There are three possible returns as follows:

Non-skip if record is locked
(A0) = 0
(A2) = a(channel)

1-skip if record not allocated and disc or account full
(A0) 0 if account is full
(A0) 0 if disc is full

2-skip if record found and block read; registers as shown in table

Subroutine: GET RECORD, READ (Group 1)

Calling Sequence: CALL
GETRR

Use: Locates and reads a selected record of a file for reading data.

Ac	Entry	Return (2-skip)
0	x	x
1	record number (see text)	d (data block)
2	a(channel)	B (record)
3	x	x
C	x	x
BSA	x	data block
HBA	x	File header
HXA	x	extender if file is extended

GET RECORD, READ uses the STS word of the channel to determine if the open file is formatted, unformatted, or contiguous. From this information, GETRR determines the disc address of the appropriate data block and reads that block into BSA. GETRR then determines the location in that block of the desired record and generates a byte pointer to it.

If a -1 is supplied in A1 as the record number, GETRR will look up the next record by using information in the channel. For formatted files, the record number will be (FSZ)+1. For contiguous files, the record address will be (CBN)*400(octal)$+_8$(STS)$_0$+2*(WPR). For text or unformatted files, the record address will be (CBN)*400(octal)$+_8$(STS)$_0$.

If a -2 is supplied in A1 as the record number, GETRR will look up the present record. For formatted files, the record number is (FSZ). For all others, the record address will be (CBN)*400(octal)$+_8$(STS)$_0$. Note that for a text file this is the same as for -1 in A1.

There are three possible returns as follows:

Non-skip if record is locked
(A0) = 0
(A2) = a(channel)

1-skip if record not written

2-skip if record found and block read; registers as shown in table

Subroutine: WRITE ITEM (Group 1)

Calling Sequence: CHANNEL
WRITITEM

Use: Writes an item into a data file or to a device.

Ac	Entry	Return (8-skip)
0	channel number	x
1	x	x
2	a(ICB)	x
3	x	x
C	x	x
BSA	x	data block (if file write)
HBA	x	file header (if file write)
HXA	x	file extender (if extended file write)

where ICB is the "Item Control Block" as follows:

word #0 record number
word #1 item number (origin 0) or byte displacement
word #2 item type
word #3 item length (#words or bytes)
word #4 a(source) or a byte address for a string

WRITE ITEM writes an item into a data file or to a peripheral device opened on the selected channel. The number of words or bytes transferred will be the smaller of the item length and the source length. If the item length is less than the source length, the transfer is truncated. If the item length is greater than the source length then a non-string will be padded with zeroes, and a string item will be terminated with a single zero byte. Word #1 of the ICB is stepped to point to the next item in the record after a successful write.

There are nine returns as follows:

Non-skip if illegal channel number
BSA=unchanged
HBA=unchanged

1-skip if channel not open
A0=unchanged
A2= a(channel)
BSA=unchanged
HBA=unchanged

(continued)

2-skip if file not formatted
A1=d(file header)
BSA=unchanged
HBA=file header (if file access)

3-skip if file is write protected
BSA=unchanged
HBA=file header (if file access)

4-skip if disc or user's account is full
A0=# of blocks the disc needs if positive, or
A0=-# of blocks user's account needs if negative
BSA=unknown
HBA=file header (if file access)

5-skip if record is locked
A0=0
A1=recommended pause (tenth-seconds)
BSA=unknown
HBA=file header (if file access)

6-skip if item number is illegal
Registers are indeterminate
BSA=data block (if file access)
HBA=file header (if file access)

7-skip if item types don't match
A0=desired type
A1=actual type
A2=a(Item Control Block)
BSA=data block (if file access)
HBA=file header (if file access)

8-skip if item is written; registers as shown in table
The item number in ICB is incremented if file access
BSA=data block (if file access)
HBA=file header (if file access)

Subroutine: READ ITEM (Group 1)

Calling Sequence: CHANNEL
READITEM

Use: Reads an item from a data file or from a device.

Ac	Entry	Return (8-skip)
0	channel number	#words or bytes transferred
1	x	x
2	a(ICB)	x
3	x	x
C	x	x
BSA	x	data block (if file read)
HBA	x	file header (if file read)
HXA	x	file extender (if extended file read)

where ICB is the "Item Control Block" as follows:

word #0	record number
word #1	item number (origin 0)
word #2	item type
word #3	desired length (#words or bytes)
word #4	a(destination) or a byte address for a string

READ ITEM accesses an item from a data file or from a peripheral device opened on the selected channel. The amount of data transferred will be the smaller of the item length and the user's destination size. If the item length is greater than the destination, the item is truncated. If the item length is less than the destination length, a non-string item will be padded with zeroes, and a string item will be terminated with a single zero byte.

The returns from READ ITEM are the same as for WRITE ITEM except for the 4-skip return:

4-skip if record not written
Registers are indeterminant
BSA=unknown
HBA=file header

Subroutine: FIND ITEM (Group 3)

Calling Sequence: CHANNEL
FINDITEM

Use: Locates an item in a data file by its contents.

(to be added)

Subroutine: UNLOCK RECORD (Group 1)

Calling Sequence: CALL
UNLOCK

Use: Unlocks a record on a specified channel.

Ac	Entry	Return (skip)
0	channel number	STS of channel
1	x	FDA of channel
2	x	a(Data File Table entry)
3	x	x
C	x	x

UNLOCK unlocks the record by clearing bit 15 of the STS cell in the data channel.

There are two possible returns as follows:

Non-skip if illegal channel number or channel not open
(A1)=number of channels available if illegal channel number, or
(A1)=0 if channel not open
(A2)=a(channel)

Skip if successful; registers as shown in table

Subroutine: WRITE DISC BLOCK (Group 1)

Calling Sequence: JSR @.WBLK

Use: Writes one block (256 words) from core onto a disc.

Ac	Entry	Return
0	Logical Unit number	unchanged
1	Real Disc Address	unchanged
2	core address	unchanged
3	x	x
C	x	x

WBLK checks for certain software errors, then checks whether the Logical Unit and disc address given are the same as that of a block in BSA, HBA, HXA, or SSA. If the same as the block in BSA, the BSA change flag (BSACF) is cleared. If the same as any of the five buffer areas, that buffer's disc address flag is cleared.

If the core address is BSA, HBA, HXA, or SSA, then (A0) and (A1) are stored in the corresponding disc address flags. If the core address is HBA, the DHDR cell in HBA must equal (A1), and the UNIT cell in HBA must equal (A0).

WBLK does a non-skip return if successful or branches to FAULT if any software error or disc write error is detected. In the case of a disc write error, 16 attempts are made to write the block before a FAULT is indicated.

Software errors checked for include:

a) Core address greater than BSA and not exactly equal to HBA, HXA, or SSA, and not wholy within ABA.
b) Core address less than BPS and not equal to 200 octal.
c) Invalid disc address.
d) Core address equal to HBA, and DHDR cell in HBA not equal to (A1) or UNIT cell in HBA not equal to (A0).
e) Disc address zero (attempt to overwrite BZUP).

Subroutine: READ DISC BLOCK (Group 1)

Calling Sequence: JSR @.RBLK

Use: Reads one block (256 words) from a disc into core.

Ac	Entry	Return
0	Logical Unit number	unchanged
1	Real Disc Address	unchanged
2	core address	unchanged
3	x	x
C	x	x

RBLK checks for certain software errors, then checks whether the core address is BSA, HBA, HXA, or SSA. If it is one of these buffer areas, RBLK checks whether the desired disc block is already in core where desired, and returns without actually reading if it is.

The core address is then checked to see if BSA will be overlayed; if so, the BSA change flag is checked, and BSA is first written back on the disc if it has been changed. Finally, the selected disc block is read into core at the address in A2.

RBLK does a non-skip return if successful or branches to FAULT if any software error or disc read error is detected. In the case of a read error, 16 attempts are made to read the block before a FAULT is indicated.

Software errors checked for include:

a) Core address greater than BSA and not exactly equal to HBA, HXA, or SSA, and not wholy within ABA.
b) Core address less than BPS and not equal to 200 octal.
c) Invalid disc address.

Subroutine: CHECK "BSA CHANGED" FLAG (Group 1)

Calling Sequence: CALL
CBSA

Use: Allows new information to be stored in BSA.

Ac	Entry	Return
0	x	x
1	x	unchanged
2	x	unchanged
3	x	see text
C	x	x

If the BSA CHANGED flag (BSACF) is non-zero then the block in BSA will be written on the disc at the disc address in DBSA, and BSACF will be zeroed.

If CBSA is called from RBLK, then A3 will contain the original return address from the call to RBLK. Otherwise, (A3) is undetermined.

Subroutine: FIND LOGICAL UNIT TABLES (Group 1)

Calling Sequence: JSR @.FLUT

Use: Finds the LUFIX, LUVAR, and LUT pointers for a Logical Unit.

Ac	Entry	Return (skip)
0	Logical Unit number	unchanged
1	x	a(LUT entry)
2	x	a(LUFIX)
3	x	a(LUVAR)
C	x	0

The Logical Unit Table (LUT) is searched for the Logical Unit number given in A0. There are two possible returns as follows:

Non-skip if Logical Unit is not active
A0=unchanged

Skip if Logical Unit found; registers as shown in table

Subroutine: CONVERT LOGICAL TO REAL DISC ADDRESS (Group 1)

Calling Sequence: CALL
CLRA

Use: Builds a Real Disc Address from its logical components.

Ac	Entry	Return
0	a(LUFIX)	x
1	x	Real Disc Address
2	a(logical address table)	unchanged
3	x	logical sector
C	x	x

CLRA uses the conversion factors from the LUFIX at (A0) to convert the logical cylinder at (A2), the logical track at (A2)+1, and the logical sector at (A2)+2 into a Real Disc Address which is returned in A1. The logical address table is unchanged.

Subroutine: CONVERT REAL TO LOGICAL DISC ADDRESS (Group 1)

Calling Sequence: CALL
CRLA

Use: Converts a Real Disc Address to its logical components.

Ac	Entry	Return
0	a(LUFIX)	x
1	Real Disc Address	logical sector
2	a(logical address table)	unchanged
3	x	logical track
C	x	x

A Real Disc Address given in register A1 is converted to a logical disc address and stored in the three word logical address table at (A2). CRLA will put the logical cylinder at (A2), the logical track at (A2)+1, and the logical sector at (A2)+2.

No checking is done for an illegal address. The LUFIX pointer in A0 is used to find the conversion factors.

Subroutine: INCREMENT REAL DISC ADDRESS (Group 1)

Calling Sequence: CALL
IRDA

Use: To determine the n^{th} legal Real Disc Address after a given Real Disc Address.

Ac	Entry	Return(skip)
0	Logical Unit number	first unused Real Disc Address
1	any Real Disc Address	incremented Real Disc Address
2	increment value	a(Logical Address)
3	x	a(LUVAR)
C	x	x

Determines the n^{th} legal Real Disc Address after the Real Disc Address given in A1, where n is given in A2. Information from the LUFIX and LUVAR of the specified Logical Unit is used to control the method of determination.

On return, (A2) points to a table containing the logical cyclinder, logical track, and logical sector of the incremented disc address. There are two possible returns as follows:

Non-skip if the incremented Real Disc Address has not been determined
In this case, (A0) indicates the reason as follows:
(A0)=unchanged if Logical Unit is inactive
(A0)<0 if result is not a legal Real Disc Address

Skip if the desired Real Disc Address has been determined; registers as shown in table

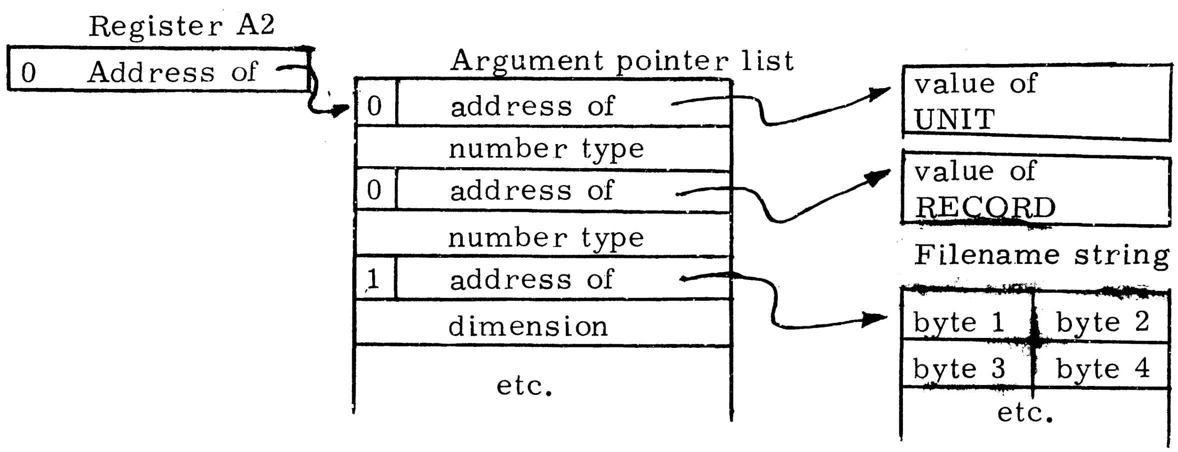
Subroutine: READ FILE HEADER INFORMATION (Group 2)

Calling Sequence: CALL
RDFHI

Use: To determine a disc file's characteristics.

Ac	Entry	Return (skip)
0	x	x
1	x	x
2	a (Argument pointer list)	x
3	x	x
C	x	x

where the Argument pointer list contains pointers to UNIT, RECORD, FILENAME, ACCOUNT #, TYPE, SIZE, STATUS, COST, INCOME, CREATION DATE, LAST ACCESS DATE, AND D(FILE'S HEADER) in the form:



RDFHI is accessable as CALL 97 from a BASIC program. RDFHI will look in INDEX for the file at the selected record. If an empty INDEX record is encountered, the next record will be examined. If a negative record number is specified or the selected record is beyond the INDEX, -1 is returned as the record. At each call, the record number is incremented to the next entry. If a file exists in the record, the header is read, and pertinant information is stored as the argument values. If the value of the INCOME argument is zero at the time of the call, the file's income is cleared, and the header is written back. There are two possible returns as follows:

Non-skip if:

1) Not logged onto an account at least priv 2, and group 0
2) Too small a number type to store a parameter
3) String Dimension<14

Skip if record read

Subroutine: DIRECTORY (Group 3)

Calling Sequence: CALL DIRECTORY

Use: Sets up and initializes the directories for an indexed contiguous data file.

Ac	Entry	Return (skip)
0	a(argument list)	x
1	directory number	status (see text)
2	channel pointer	x
3	x	x
C	x	x
BSA	x	x
HBA	x	file header

where the argument list at (A0) is as follows:

word #0	(not used)
word #1	(not used)
word #2	address of numeric item #1
word #3	number type for item #1
word #4	address of numeric item #2
word #5	number type for item #2

If (A1) is non-zero then it specifies a directory number which must be exactly one greater than the last directory already specified for the file. (A1) must equal one if no directories have yet been specified. The key length of the specified directory is set equal to the integer value of item #1.

If (A1) is zero then an initializing operation is performed. The number of data records given in item #1 is assumed for the purpose of calculating the size of each directory. Each directory will consist of three levels: a one-block master level, a coarse level, and a fine level. All spare blocks in each level of each directory are linked together on a free block chain. The file header is then marked as "indexed" and written back on the disc.

(continued)

The size of each directory is computed as follows:

$$\text{number of keys per block} = \frac{254}{\text{key length (\# words)} + 1}$$

$$\text{size of fine level} = \frac{\text{\# data records} * 2}{\text{\# keys per block} + 1} \quad \text{blocks}$$

$$\text{size of coarse level} = \frac{\text{\# fine level blocks}}{\text{\# keys per block} - 1} \quad \text{blocks}$$

The number of blocks in the coarse level must not exceed the number of keys per block since this would cause the master level to exceed one block.

There are two possible returns as follows:

Non-skip if the file is write protected

Skip if the file is not write protected; register A1 indicates the status as follows:

(A1)	Status
0	Successful operation
6	Directory number not in sequence
7	File is not contiguous
10	File is already indexed
11	Item #1 is negative or too large
12	Too many directories
13	Master level of directory exceeds one block
14	Directories exceed file size

Subroutine: SEARCH (Group 3)

Calling Sequence: CALL
SEARCH

Use: Searches a specified directory of an indexed contiguous data file and inserts or deletes an index entry if required. Also maintains a free data record chain.

Ac	Entry	Return (skip)
0	a(argument list)	x
1	mode, directory #	status (see text)
2	channel pointer	x
3	x	x
C	x	x
BSA	x	x
HBA	x	x
HXA	x	x
ABA	x	x

where the argument list at (A0) is as follows:

word #0	address of a string item v$
word #1	dimension of string v$
word #2	address of numeric item v1
word #3	number type for item v1
word #4	address of numeric item v2
word #5	number type for item v2

Register A1 contains the mode (m) in the top byte and the directory number (d) in the lower byte, and register A2 points to the data channel where the file is open. The variable list is defined as follows:

v$ contains the key for which the search is being made.
v1 receives the record number (result of the search).
v2 receives a status value as follows:

0 No error, search was successful
1 Search was not successful
2 End of directory
3 End of data
4 Wrong variable type
5 Undetermined error
6 File not indexed

Error 5 will occur if the system's Auxiliary Buffer Area (ABA) is less than 1004 words octal, if an illegal command is given, or if the file is not structured as expected; for example, if there are fewer directories than the directory number specified.

(continued)

The available modes of operation are as follows:

m=1, d≠0 — Reads the key length (number of words) of directory d into v1. If directory d does not exist then v1 is set equal to zero.

m=1, d=0 — Performs the operation specified by the value given in v2 as follows:

- v2=0 Reads into v1 the record number of the first real data record.
- v2=1 Reads into v1 the number of available data records on the free record list.
- v2=2 Reads into v1 the record number of an available data record and removes that record from the free record list.
- v2=3 Releases the data record whose record number if given in v1 and puts the record on the free record list.

m=2 Searches directory d for a match with the key value in v$. If a match is found (even if the key in the directory entry is longer then v$). returns the entire key in v$, returns the associated data record number in v1, and sets v2 equal to zero. If not found, leaves v$ and v1 unchanged, and sets v2 equal to one.

m=3 Searches directory d for the first key whose value logically exceeds the value in v$. If found, returns the key value in v$, returns the associated data record number in v1, and sets v2 equal to zero. If not found, leaves v$ and v1 unchanged, and sets v2 equal to three.

m=4 Searches directory d for a match with the key value in v$. If found, returns the associated data record number in v1, and sets v2 equal to one. If not found, and there is directory space available, inserts the key into directory d, references the key to the data record number given in v1, and sets v2 equal to zero. If not found and the insert cannot be made then v2 is set equal to two.

m=5 Searches directory d for a match with the key value in v$. If found, deletes the key from directory d, returns the data record number in v1, and sets v2 equal to zero. If not found, v1 is left unchanged, and v2 is set equal to one.

There are two possible returns as follows:

Non-skip if file is write protected; registers undetermined

Skip if not write protected; registers as shown in table

Subroutine: STORE BYTE (Group 1)

Calling Sequence: JSR @.STBY

Use: Stores one byte at a given address in core.

Ac	Entry	Return
0	byte	byte
1	byte address	word
2	x	word address
3	x	return address
C	x	left/right byte flag

A0 is masked with 377 octal to clear the top half of the word, and the result is stored at the byte address in A1. The other byte in that word is not disturbed. The resulting word is also returned in A1 and its core address in A2.

Subroutine: ACCESS BYTE (Group 1)

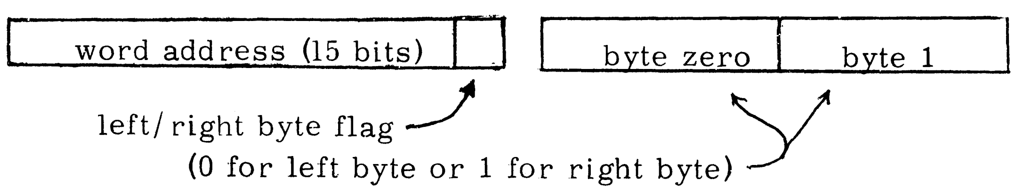
Calling Sequence: JSR @.ACBY

Use: Accesses one byte from a given location in core.

Ac	Entry	Return
0	x	x
1	byte address	unchanged
2	x	byte from B(A1)
3	x	return address
C	x	left/right byte flag

Accesses one byte from the byte address in A1 and returns that byte in the lower 8 bits of A2. The upper 8 bits of A2 will be zero.

A byte address is defined as follows:



Subroutine: IS (A2) A DIGIT? (Group 1)

Calling Sequence: JSR @.IA2D

Use: Determines whether register A2 contains an ASCII code for a decimal digit.

Ac	Entry	Return
0	x	unchanged
1	x	see text
2	byte	unchanged
3	x	return address
C	x	unchanged

There are two possible returns as follows:

Non-skip if (A2) is not an ASCII code for a digit; A1 will contain octal 271 if (A2) $>$ 271 or octal 260 if (A2) $<$ 260

Skip if (A2) is an ASCII digit (i.e., 260 $\leq$ (A2) $\leq$ 271); in this case, A1 will contain octal 260 (ASCII zero) so that a SUB 1,2 instruction will generate the binary value of the digit

Subroutine: IS (A2) A LETTER? (Group 1)

Calling Sequence: JSR @.IA2L

Use: Determines whether register A2 contains an ASCII code for a letter.

Ac	Entry	Return
0	x	332
1	x	301
2	byte	see text
3	x	return address
C	x	see text

There are two possible returns as follows:

Non-skip if (A2) is not an ASCII code for an upper case letter; in this case, A2 and the carry flip-flop are unchanged

Skip if (A2) is an upper case ASCII letter (i.e., 301 $\leq$ (A2) $<$ 332); in this case, the carry flip-flop will be toggled, and 301 will be subtracted from A2; thus, the letter A will be represented by zero, B by one..., and Z by 31 octal

Subroutine: MOVE WORDS (Group 1)

Calling Sequence: CALL
MOVE

Use: Moves the contents of a group of words in core to another area in core.

Ac	Entry	Return
0	first source address	x
1	last source address	x
2	first dest. address	see text
3	x	see text
C	x	x

Registers A0 and A1 point to the first and last words of the area to be moved. Register A2 points to the first word of the destination area, which is the same size as the source area. A2 may point to any location in core, including any location within the source area. The data is moved to the destination area in such a way that the destination area is an exact copy of the source area as it was before the move, even if the source and destination areas overlap.

If (A0) < (A2) then the last source word is moved first, then each preceeding word is copied until the first word has been moved. On return, the registers will contain:

A2 = first destination address
A3 = first source address

If (A0) > (A2) then the first source word is moved first. On return, the registers will contain:

A2 = last destination address
A3 = last source address

Note: if (A0) = (A2) there is an immediate return with no change to the contents of any register.

Subroutine: MOVE BYTES (Group 1)

Calling Sequence: CALL
MOVBYTES
Terminator code

Use: Moves a block of bytes in core.

Ac	Entry	Return
0	B(beginning of source)	B(last dest byte transferred)
1	B(end of source)	number of bytes not transferred
2	B(beginning of destination)	last byte transferred
3	x	x
C	x	x

MOVE BYTE tranfers the byte at the byte address given in A0 to the byte address given in A2. Both byte addresses are incremented, and the move continues in this manner until terminated by one of three conditions:

a) The source byte address of the byte just transferred equals (or exceeds) the byte address given in A1,

b) The byte just transferred was zero, or

c) The byte just transferred is identical to the terminator code following the CALL MOVBYTES.

If on entry (A0) > (A1) then no bytes will be transferred, and registers A0 and A2 will be unchanged.

In any case the return is to the location after the terminator code. There are no error indications.

Subroutine: CONVERT PORT NUMBER TO RTA POINTER (Group 1)

Calling Sequence: CALL
CPNRP

Use: To locate the Resident Table Area for a given port.

Ac	Entry	Return
0	port number	a(port's RTA)
1	x	0
2	x	size of RTA
3	x	return address
C	x	x

There are two possible returns as follows:

Non-skip if (A0) is not a legal port number
A0=unchanged
A1=number of active ports
A2=unchanged
A3=return address

Skip if RTA located; registers as shown in table

Subroutine: CONVERT RTA POINTER TO PORT NUMBER (Group 1)

Calling Sequence: CALL
CRPPN

Use: To determine the port number for a given Resident Table Area.

Ac	Entry	Return (skip)
0	a(any RTA)	port number
1	x	port number
2	x	size of RTA
3	x	return address
C	x	x

There are two possible returns as follows:

Non-skip if (A0) is not an RTA pointer
A2=size of RTA

Skip if port number determined; registers as shown in table

Subroutine: CONVERT DRATSAB TO ASCII (Group 3)

Calling Sequence: CALL
CDTA

Use: Converts a string of bytes in DRATSAB code into the corresponding ASCII codes and stores the results in a specified destination.

Ac	Entry	Return
0	Hollerith/BASIC flag	number of bytes transferred
1	B(source string)	non-zero if error, data not transferred
2	a(ICB)	x
3	x	skip distance for return
C	x	x

where ICB is the Item Control Block described under READ ITEM. CDTA first checks that the item type is a string, and then accesses each DRATSAB code in the source string, converts it to the equivalent ASCII code, and stores the result in the destination string. This continues for each byte in the string starting at the byte address in A1 and ending when either a RETURN character or an END OF RECORD code (300 or 310, respectively, in DRATSAB code) is converted.

DRATSAB code is compressed from the twelve row punched card code as follows:

8	9	12	11	0	1-7

In the above DRATSAB byte, each number indicates the card row stored in that bit. There may be only one punch in rows one through seven; the number of the punched row is converted to binary and placed in the lower three bits of the DRATSAB code.

Register A0 must be zero if the cards are punched in Hollerith standard key punch codes or non-zero if the cards are marked as shown on the EDS BASIC Card Programmer.

The following control codes are translated the same for either Hollerith or BASIC cards:

Card rows	DRATSAB	Meaning
8-9	300	RETURN
8-9-5	305	CTRL E
8-9-0	310	End of Card

Subroutine: SEND SIGNAL (Group 1)

Calling Sequence: CALL
SIGPAUSE

Use: Sends a signal to a user on another port or to a later program segment on the same port.

Ac	Entry	Return
0	1	x
1	Sender's RTA pointer	x
2	a(parameter list)	x
3	x	x
C	x	x

Register A2 must point to a three-word parameter list of the form:

word0: Destination port number or RTA pointer
word1: Signal value#1
word2: Signal value #2

A signal value may be any 16 -bit binary word. There are two possible returns as follows:

Non-skip if signal buffer is full or if there is no such destination port

Skip if signal was successfully stored in the signal buffer

Subroutine: RECIEVE SIGNAL (Group 1)

Signal Sequence: CALL
SIGPAUSE

Use: Receives a signal if any have been sent to the regnant user's port.

Ac	Entry	Return
0	2	x
1	x	x
2	a(parameter list)	a(sender's RTA)
3	x	x
C	x	x

Register A2 must point to a three-word parameter list. The contents of the list are ignored. If a signal is received, it will be stored in the list in the form:

word 0: Port number of sender
word 1: Signal value #1
word 2: Signal value #2

There are two possible returns as follows:

Non-skip if no signal was received; in this case, the parameter list is unchanged

Skip return if a signal was received; the signal will be in the parameter list as shown above

Subroutine: PAUSE (Group 1)

Calling Sequence: CALL
SIGPAUSE

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

Ac	Entry	Return
0	3 or 4 (see text)	x
1	delay in tenth-seconds	x
2	x	x
3	x	x
C	x	x

The regnant user's task will be bumped, and the Pause Delay Counter (PDC) in his RTA will be set to the value in A1. If (A0)=3, he will be put in the task queue only after a delay of (A1) tenth-seconds. If (A0)=4, he will be put in the task queue after the delay or when any signal is sent to him, whichever occurs first (immediate return without bump if a signal is waiting for this port).

In any case, the return is non-skip.

Subroutine: COMPARE STRINGS (Group 1)

Calling Sequence: CALL
CSTR

Use: Tests whether two strings are equivalent.

Ac	Entry	Return
0	B(string-one)	B(terminator of string one)
1	B(string two)	B(terminator of string two)
2	x	Last character of string two
3	x	zero if strings are equivalent
C	x	x

CSTR compares two alphameric strings until differing bytes are found or terminating characters are found in one or both. It returns zero in A3 if the strings are equivalent.

Subroutine: PASSWORD COMPARE (Group 1)

Calling Sequence: CALL
PASSCOMPARE

Use: Tests whether the user supplied the correct password.

Ac	Entry	Return (skip)
0	x	x
1	B(password)	x
2	x	terminating character
3	x	x
C	x	x

PASSC compares a password pointed to by the byte address in A1 against the string in thr regnant user's I/O buffer. For correct comparison, the string in the I/O buffer must terminate with a RETURN or CTRL E at the same point that the password is terminated by a zero byte.

There are three possible returns as follows:

Non-skip if no password given
A0=205 (CTRL E)
A2=first non-space character of input
Next byte accessed will be the first non-space

Non-skip if incorrect password given
A0=mismatched byte of password
A2=mismatched byte of input
Next byte accessed is next after mismatched byte

Skip if correct password given; registers as shown
Next byte accessed is next after end of password (a trailing CTRL E will be scanned off as part of the password)

Subroutine: ACCOUNT LOOKUP (Group 1)

Calling Sequence: CALL
ACNTLOOKUP

Use: Finds a user's account entry in the ACCOUNTS file via the Account I.D., account number, or entry position.

Ac	Entry	Return (skip)
0	see text	account entry position
1	see text	d(ACCOUNTS block)
2	a(256 word buffer)	a(account I.D.)
3	x	x
C	x	x
BSA	x	x
HBA	see text	unchanged

ACNTL looks up an account in the ACCOUNTS file on Logical Unit zero. It uses a caller-supplied disc buffer to read in the ACCOUNTS file and returns with information so that the caller may modify or create a user's account.

(A0) prescribes the method by which the account is looked up as follows:

If (A0)=0, then (A1)=account entry position: i.e. 1 for first account, 2 for second, etc.

If (A0)=-1, then (A1)=B(Account I.D.) which is an ASCII string of not more than twelve characters.

If (A0)=1, then (A1)=account number (user number is in bits 7-0, group number in bits 13-8; bits 15 and 14 are ignored).

If (A0)>1, then (A1)=account number as above and HBA must have a file's header. ACNTL will lookup on the Logical Unit of the file. This mode should be used for all disc block usage updates.

There are two possible returns as follows:

Non-skip if account does not exist
(A0)=available account record number (zero if none available)
(A1)=d(ACCOUNTS block)
(A2)=a(available account entry in core)

Skip if account found; registers as shown

Subroutine: CHANGE OR CHECK FLAG (Group 1)

Calling Sequence: FLAGCHANGE
command+displacement+skip
mask

Use: To change and/or check the state of a specified bit (or bits) in a flag word.

Ac	Entry	Return
0	x	x
1	x	x
2	table pointer	pointer to flag word
3	x	address of command word
C	x	x

The "command" in the calling sequence may be omitted if it is desired to check the state of a flag without changing it, or it may be one of the following words:

SET	Set the masked bit(s) to one.
RESET	Reset the masked bit(s) to zero.
TOGGLE	Toggle the masked bit.

The "displacement" in the calling sequence is the number of words from the pointer in A2 to the desired flag word. The "skip" in the calling sequence may be omitted for an unconditional non-skip return, or it may be one of the following words:

SKIPZ	Skip if all masked bits are zero.
SKIPO	Skip if any masked bit is one.

Typical usage of FLAGCHANGE is shown by the following examples:

```
LDA     2,.SLT          LDA     2,.RTA
FLAGCHANGE              FLAGCHANGE
SET+FLAG                TOGGLE+FLW.+SKIPO
2000                    4
```

The first example will set bit 10 of the FLAG word in the table pointed to by .SLT and will non-skip return.

The second example will toggle bit two of the FLW word in the master port's RTA and will skip return if the result in bit two is a one, or non-skip return if the result in bit two is a zero.

Note: For a TOGGLE command, only one bit in the mask may be a one. Interrupts are disabled for a short time, then re-enabled.

Subroutine: BUMP REGNANT USER (Group 1)

Calling Sequence: JSR @.BUMP or LDA 3, address
JMP @.BUMP

Use: Bumps the regnant user from core.

Ac	Entry	Return
0	x	x
1	x	x
2	x	x
3	see text	x
C	x	x

Any processor which does not complete its task within one-half second or less must periodically check the Run Time Limiter (RTL). If RTL becomes zero or negative (indicating end of time slice) then BUMP should be called.

Specifically, the processor should periodically execute the instruction sequence.

```
LDA     0, RTL
NEGL#   0, 0, SNC
JSR     @.BUMP
```

The processor's Swap-Out and Swap-in subroutines must save and restore any information . will be required for the next time slice. Such information is usually stored in the user's active file or in the FMAP cells of the active file header.

The second calling sequence may be used if control is to be given to the processor at a different location when brought in for the next time slice. Otherwise, control will resume at the instruction immediately following the JSR @.BUMP.

Subroutine : LOAD USER'S ACTIVE FILE (Group 1)

Calling Sequence: CALL
LUSR

Use: Loads the regnant user's active file into core.

Ac	Entry	Return (skip)
0	x	x
1	x	x
2	x	x
3	x	x
C	x	x

LUSR is usually called by a processor's swap-in subroutine. There are two possible returns as follows:

Non-skip if the file type of the active file is not the same as the file type of the processor

A0=file type of active file
A1=d(active file header)
A2=a(HBA)

Skip if file types match; the active file has been loaded into core

In either case, HBA will contain the active file header.

Subroutine: EXIT (Group 1)

Calling Sequence: CALL
EXIT

Use: Exits from a processor.

Ac	Entry
0	x
1	x
2	x
3	x
C	x

When processor has finished its task or has been aborted for any reason (such as a non-recoverable error or an ESCAPE) then the processor must jump to EXIT to return the port to system control mode; i.e., to cause a # symbol to be printed and to set up SCOPE as the user's processor.

If an output is currently in progress, it will be allowed to finish before the # symbol is printed.

Subroutine: START IPL (Group 1)

Calling Sequence: JMP @.STPL

Use: Aborts all system operations and perform an Initial Program Load.

Ac	Entry
0	x
1	x
2	x
3	x
C	x

START IPL will be called by RECOVER if the Recover Inhibit Flag is set.

Subroutine: FAULT (Group 1)

Calling Sequence: JSR @.FALT or JSR @.FALT
n*K+NOP

Use: Aborts a process due to an illegal condition or a hardware failure.

Ac	Entry
0	x
1	x
2	x
3	x
C	x

FAULT is a DISCSUB, but it has a special calling routine in core to prevent nesting if it is called by another DISCSUB.

FAULT prevents swapping while it types out a status message giving the trap number and location and the contents of all registers and the carry flip-flop. If the call was from another DISCSUB, then the location in the DISCSUBS file is also given. This status message is printed on the regnant user's terminal or on the master terminal if there is no regnant user.

If an expression of the form n*K+NOP follows the JSR @.FALT then the value of n, where n is any value from zero to 177 octal, is printed in decimal as the trap number. If the expression n*K+NOP is not given then trap number zero is assumed.

FAULT waits for the type out to finish rather than calling WONA. Only the interrupt handling tasks will be processed during the type out. After completion, FAULT aborts the regnant task and transfers control to the system via the RECOVER routine.

Refer to "Trap Messages" in the Manager's Reference Manual for a discussion of the trap message itself. Refer also to Appendix 2 of the same manual for a list of currently assigned trap numbers.

Subroutine: BINARY MULTIPLY (Group 1)

Calling Sequence: JSR @.BMUL

Use: Multiplies two unsigned 16-bit binary integers to produce a 32-bit product.

Ac	Entry	Return
0	Multiplier	Product
1	x	unchanged
2	Multiplicand	unchanged
3	x	Product overflow
C	x	x

Return is non-skip. The 32-bit product is returned in registers A0 and A3 with the most significant half in A3.

Subroutine: BINARY DIVIDE (Group 1)

Calling Sequence: JSR @.BDIV

Use: Divides two unsigned 16-bit binary integers to produce a 16-bit quotient and a 16-bit remainder.

Ac	Entry	Return
0	Divisor	unchanged
1	Dividend	Remainder
2	x	0
3	x	Quotient
C	x	x

Return is non-skip. The quotient returned in A3 will be equal to the integer part of dividend/divisor, and the remainder in A1 will be less than the divisor.

Subroutine: CONVERT DATE TO ASCII (Group 1)

Calling Sequence: CALL
CNVDA

Use: Converts the system's representation of the date and time to an ASCII string.

Ac	Entry	Return (skip)
0	see text	x
1	B(dest) see text	B(terminator)
2	a(clock) see text	x
3	x	x
C	x	x

If (A1) $\neq 0$ it will be used as the byte address to store the string, and (A0) must contain the dimension of the destination string.

If (A1) = 0 then the string will be stored in the regnant user's I/O buffer, and (A0) is ignored.

If (A2) = 0 then the system's clock is used to determine the current date and time, which is then converted to a string of ASCII characters in the form

JUN 16, 1973 14:25:08

The string is exactly 22 characters long, plus a zero byte as a terminator. The time, which is based on a 24 hour clock, is given in hours, minutes, and seconds.

If (A2) $\neq$ 0 then it must be a pointer to the date and time to be converted, which must be in the form:

(A2) →	
	hours after 1 January 1973
	part of hour in tenth-seconds

Both values are in binary; the hours value must assume that all months have 31 days.

There are two possible returns as follows:

Non-skip if illegal time value (second word > 35999 decimal).

Skip if conversion successful

Subroutine: CONVERT ASCII TO DATE (Group 1)

Calling Sequence: CALL
CNVAD

Use: Converts an ASCII string representing a date and time to a pair of binary words.

Ac	Entry	Return (skip or non-skip)
0	x	hours after 1 January 1973
1	B (string)	part of hour in tenth-seconds
2	x	terminating code
3	x	B (next byte of string)
C	x	x

The string representing the date and time may be in either of two forms as follows:

FEB 20, 1973 11:09:56

or

73, 20, 2, 11, 09, 56

where the second string is the same form as requested when an IPL is performed. In either case, the first two digits of the year, the "seconds" value, and all leading zeroes are optional. Spaces, commas, and colons are interchangeable as field separators, and the string may be terminated with a zero byte, a RETURN code, or any other character that is not acceptable as part of the date/time representation.

Both values returned are binary integers. The "hours" value assumes that all months have 31 days.

There are two possible returns as follows:

Non-skip if the string is not an acceptable representation of the date and time; any value already converted is returned in the registers

Skip if the complete date and time have been converted (with the possible exception of the "seconds" value, which is assumed to be zero if not given)

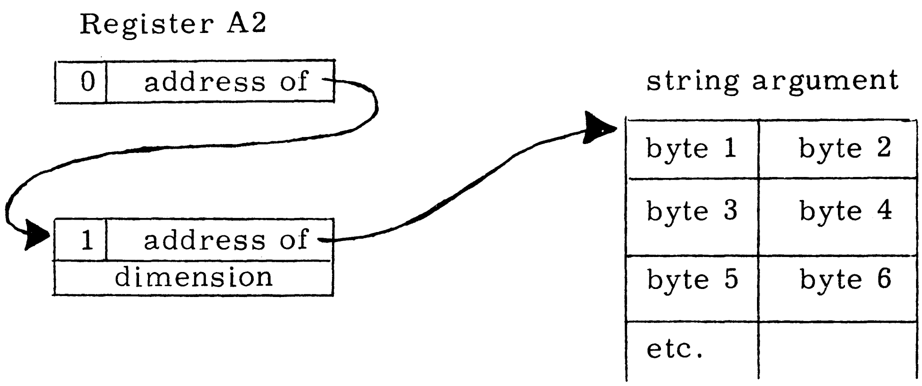
Subroutine: CONVERT DATE AND TIME (Group 1)

Calling Sequence: CALL
CNVDT

Use: Reads or sets the system's real time clock.

Ac	Entry	Return (skip)
0	x	x
1	x	x
2	a(argument pointer)	x
3	x	x
C	x	x

where the argument pointer list contains pointers to a string pointer in the form:



There are two modes of operation as follows:

1) If byte 1 of the string is zero then CNVDA is called to read the system's real time clock and convert it to a string as described in the write up for that subroutine. This mode will always skip return.

2) If byte 1 is non-zero then a non-skip return will occur unless the user is logged on to the Manager or the System account, in which case CNVAD is called to convert the string to two binary words as described in the write up for that subroutine. If CNVAD does a skip return with a RETURN or a zero byte as the terminating code then the result is placed in the system's real time clock, all active users' log-on times are adjusted to avoid erroneous connect times, and CNVDT does a skip return. Return will be non-skip if any error is detected, and the system's clock is not affected.

CNVDT is accessable as CALL 99 from a BASIC program.

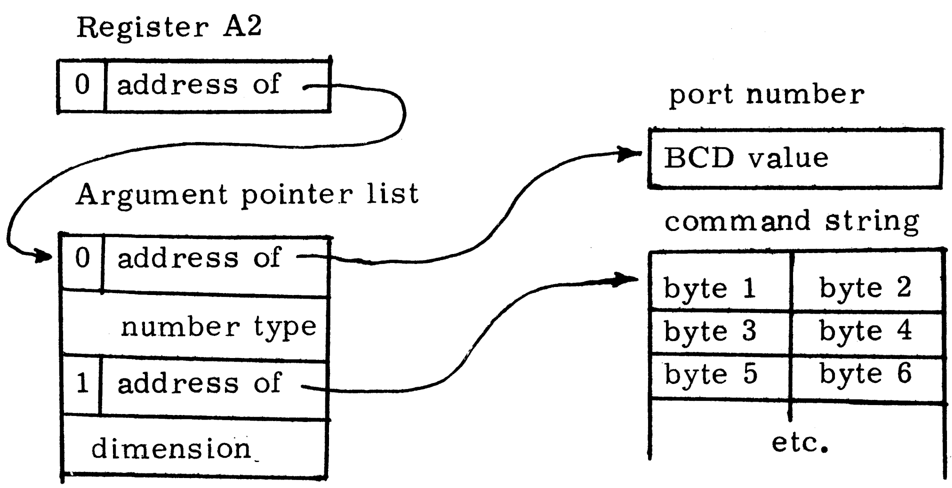
Subroutine: SYSTEM COMMAND TRANSMITTER (Group 2)

Calling Sequence: CALL
SYSCO

Use: Transmits a system command to another port.

Ac	Entry	Return (skip)
0	x	x
1	x	x
2	a (argument pointer list)	x
3	x	x
C	x	x
BSA	x	x
HBA	x	x
HXA	x	x

where the argument pointer list contains pointers to a port number and a system command string in the form:



SYSCO transmits the system command given to the specified port as if a user at that port had typed the same system command in response to the system's # prompt character.

The destination port must be an interactive port, but it need not have a terminal connected to it. If there is no terminal associated with the specified port, the user should be cautioned that any non-channel input or output will freeze that port's operation. The port must be in the system command mode in order to accept a new system command. The caller may place a port in the system command mode by supplying a "\" character (shift L) as the command string.

SYSCO is accessable as CALL 98 from a BASIC program.