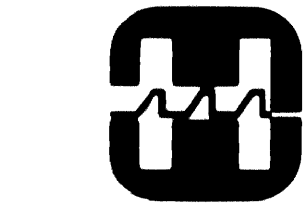
VULCAN SYSTEM SERVICES

REFERENCE MANUAL



HARRIS

COMPUTER SYSTEMS

VULCAN SYSTEM SERVICES

REFERENCE MANUAL

Original Issue
August, 1977

Revision A
December 1977

Revision B
May, 1978

Revision C
December, 1978

Revision D
February, 1980

Revision E
January, 1981

Revision F
January, 1982

HARRIS CORPORATION

COMPUTER SYSTEMS DIVISION
2101 W. CYPRESS CREEK RD.
P.O. BOX 6200
FT. LAUDERDALE, FL 33310

List of Related Publications

0860002	VULCAN Job Control Reference Manual
0860004	VULCAN I/O Services Reference Manual
0860005	VULCAN Operator Communications (OPCOM) Reference Manual
0860006	VULCAN GENASYS Reference Manual
0862001	VULCAN Terminal User's Guide
0862003	VULCAN Concepts and Features
0864001	VULCAN Utilities Reference Manual
0861001	Harris COBOL Reference Manual
0861002	Harris BASIC-V Reference Manual
0861003	Harris APL Reference Manual
0861004	Harris FORTRAN 77 Reference Manual
0861005	Harris RPG II Reference Manual

Please note that model numbers and product availability are subject to change. All model numbers mentioned in this manual are supported by the VULCAN operating system software although the product to which the model number refers may not be currently available as a standard product. Please consult your sales representative concerning product availability.

Printed in U.S.A.

LIST OF EFFECTIVE PAGES

TOTAL NUMBER OF PAGES IN THIS PUBLICATION IS: 161
CONSISTING OF THE FOLLOWING:

Page No.	Change No.	Page No.	Change No.	Page No.	Change No.
Title	Rev. F	5-1	Rev. C	14-1 thru 14-4	Rev. F
A	Rev. F	5-2, 5-3	Rev. B	14-5 thru 14-7	Rev. E
i	Rev. F	5-4	Original	14-8, 14-9	Rev. F
ii	Rev. F	5-5	Rev. A	14-10, 14-11	Rev. E
iii, iv	Rev. E	5-6	Original	14-12	Rev. F
v, vi	Rev. F	5-7 thru 5-9	Rev. C	14-13	Rev. E
1-1	Rev. F	5-10	Rev. A	14-14	Rev. E
1-2 thru 1-6	Rev. C	5-11	Rev. C	14-15, 14-16	Rev. F
2-1	Rev. C	5-12	Rev. D	I-1 thru I-4	Rev. F
2-2, 2-3	Rev. B	5-13 thru 5-18	Original		
2-4	Rev. F	6-1, 6-2	Rev. C		
2-4.1	Rev. B	7-1	Rev. E		
2-5 thru 2-8	Rev. F	7-2	Original		
2-9 thru 2-12	Rev. B	8-1, 8-2	Rev. F		
2-12.1	Rev. B	8-3, 8-4	Rev. C		
2-13	Rev. D	9-1	Rev. C		
2-14	Rev. A	9-2 thru 9-3	Rev. F		
2-14.1, 2-14.2	Rev. F	9-4	Rev. E		
2-15	Rev. F	9-5	Rev. A		
2-16 thru 2-17	Rev. B	9-6	Rev. F		
2-18 thru 2-20	Rev. C	9-7, 9-8	Rev. C		
2-21	Rev. F	9-9	Rev. D		
2-23, 2-24	Rev. C	9-10 thru 9-11	Rev. F		
3-1	Rev. C	9-12 thru 9-16	Rev. A		
3-2	Rev. E	10-1	Rev. C		
3-3, 3-4	Rev. A	10-2 thru 10-14	Original		
3-5, 3-6	Rev. F	11-1, 11-2	Rev. C		
3-6.1	Rev. E	11-3, 11-4	Rev. D		
3-6.2 thru 3-6.4	Rev. D	12-1 thru 12-7/12-8	Rev. F		
3-7	Rev. E	13-1 thru 13-6	Rev. E		
3-8	Rev. B	13-6.1 thru 13-7	Rev. F		
4-1	Rev. C	13-8 thru 13-10	Rev. E		
4-2	Original				
4-3	Rev. F				
4-4	Rev. E				

Insert Latest Revision Pages. Destroy Superseded Pages.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CONT'D)

TABLE OF CONTENTS (CONT'D.)

LIST OF TABLES

CHAPTER 1
GENERAL SERVICES

1.1 INTRODUCTION

System Services provide the interface between user programs and the VULCAN system. All are implemented through use of the Branch and Link Unrestricted (BLU) instruction, which is the only way a user program can enter or access the monitor. FORTRAN calls are implemented with library routines which make the BLU calls.

VULCAN BLUs are implemented by using the spare 7 bits of the instruction to define subtypes of BLUs. This provides a maximum of 4096 different BLUs, instead of the hardware limit of 32. Software interprets this subtype into separated functions. Currently about 96 system services may be accessed, which are implemented through the use of only about 17 BLU entries.

VULCAN reserves the first 28 (BLU locations '00 thru '33) of the 32 BLU locations for standard supported system service calls. The next 2 BLU locations (BLU locations '34 and '35 are reserved for the use of VULCAN software tools. The last BLU locations (BLU locations '36 and '37) are reserved for use by the customer site for any system service which that site may implement.

All of the System Services described in this manual are defined in terms of assembly language calls. Where they exist, the FORTRAN call is also included. It should be noted that unless otherwise stated, all system services destroy all registers when called.

On return from every system service the I register is set to the current temporary storage pointer. The C register always reflects the condition of the A register.

1.2 VULCAN EQUIVALENCES (EQIVS)

Displacements into parameter lists may be referenced as labels instead of as constants. These labels are defined in the file 0000SORC*V.EQIV. Use of the labels, hereafter called equivalences or EQIVs, increases the compatibility of user-written software with future versions of the VULCAN operating system and is therefore highly encouraged.

The particular EQIVs for a service are defined in the section of this manual pertaining to the individual services. Naming conventions for EQIVs are given below:

AA!YYY — is used as a displacement into a parameter list for a service. AA is a mnemonic indicating which service is being used (e.g., LF implies $LFINFO), and YYY is a mnemonic indicating the displacement within the parameter list (e.g., LF!FC indicates the $LFINFO function code).

AA!YYY! — is a mnemonic representing the value of a bit (e.g., LF!OPN! references the value of a bit within a word of the parameter list for the $LFINFO service).

AA!YYY@ — is a mnemonic representing the bit number (the consecutive position of a bit within a word). For example, LF!OPN@ references the number of a bit within a word of the parameter list for the $LFINFO service).

AA:YYY — is used to represent the value of a function code (e.g., LF:LFN represents the $LFINFO function code to query an LFN).

EQIVs of this form are typically used as follows:

```
          TOA     AA:YYY
          TAM     PARLIST+AA!FC
          TLO     PARLIST
          BLU     $service-name
          ....
PARLIST   BLOK    n
```

Certain blocks of data that are passed by system services and that are data structures other than a parameter list are referenced by EQIVs with the following general format:

AA/YYY — where AA indicates the table structure and YYY indicates a displacement into the table structure (e.g., M/NAME is the areaname field of a QDD).

EQIVs of the format AA!YYY and AA/YYY are frequently suffixed with the characters ! and @ to indicate certain portions of a word. The ! suffix is used to indicate a mask of the field. The @ suffix is used to indicate the bit number of the low order bit of the field. For example,

```
TMA     PL+LF!OPN
DMA     =LF!OPN!
RLA     LF!OPN@
```

can be used to (1) pick up the word containing the field LF!OPN, (2) mask out all bits not in the field LF!OPN, and (3) shift the field LF!OPN to bit 0.

Table lengths are indicated by the form "AA/". Parameter list lengths are indicated by the form "AA!". In certain cases where a parameter list or table may be a different length depending upon circumstances, the form:

```
AA/.XXX
AA!.XXX
```

is used to designate the table size (XXX indicates the circumstances).

1.3 $EXIT

The Exit service provides a mechanism for normal program termination. A call to $EXIT removes the program from the active program list and returns control to the operating system. The calling sequences are:

Assembly Language	FORTRAN
BLU $EXIT	CALL EXIT

If the exiting program was chained by another program using the $CHAIN service, then the execution of the calling program is resumed. If the exiting program was running at a control-point or at an interactive terminal and was not chained to by another program, then Job Control is loaded to read subsequent commands.

1.4 $SEXIT

Special Exit is the means by which a control point or interactive terminal terminates the job stream. It should not be called by user programs. If called by user programs, the call is converted into a regular Exit call.

1.5 $ABORT

A user program may terminate abnormally by calling the Abort service, as follows:

```
BLU          $ABORT
```

The address of the instruction following the abort call is output on the abort message unless the program is interactive or control-point, and the Job Control MODE EM=OFF is in effect.

1.6 $SYSERR

A user program may output an error message with the System Error service. The calling sequence is:

```
TLO          address
TOA          X-value
TOE          error-code
BLU          $SYSERR
```

The K register holds the address to output or is suppressed by being set to -1. The address is output in octal. The A register holds the value written in the message as "X=value". It may be suppressed by setting A to zero. If non-zero and less than 1000000, A is output as a decimal number; otherwise, it is assumed to be ASCII and written as three ASCII characters. The E register contains the error code, which is used as a displacement into the system error message disc area (*VULCMESS) to obtain the meaning of the error number. The error message is output only for interactive or control-point programs which have the Job Control MODE EM=ON set.

No registers are saved on this call, and return is to the location following the BLU instruction. For example, assume:

a. Record 5678 of *VULCMESS is "EXAMPLE ERROR"

b. Mode EM=ON is in effect

c. Program is interactive and called TESTPROG

then:

```
TOK     '4156
TOA     67
TOE     5678
BLU     $SYSERR
```

will output:

```
TESTPROG ER 5678 @ 04156 X = 67: EXAMPLE ERROR
```

If A was zero and K was negative, the output would be:

```
TESTPROG ER 5678: EXAMPLE ERROR
```

1.7 $SYSABT

A user program may output an abort code and message using the System Abort service. The calling sequence is:

```
TOK     address
TOA     parameter
TOE     abort-code
BLU     $SYSABT
```

The definition of the register contents is identical to the $SYSERR service. The $SYSABT service combines a $SYSERR call with an $ABORT call. There is no return from this call. An abort message will be output from the message area (*VULCMESS), depending on the E register value, regardless of the program type or whether the EM=ON mode was set.

1.8 $DELAY

The Delay service allows a user program to wait for a specified length of time before continuing to execute. The calling sequences are:

Assembly Language	FORTRAN
TOK n BLU $DELAY	CALL TOADS ("WAIT",n)

where n is the number of 120 Hz clock ticks (100 Hz on 50 cycle power supply systems) to delay.

Upon return, the V, H, E and A registers are restored to their contents on the original call and control is returned to the location following the BLU.

For example, to delay 3 seconds:

```
TOK    360          or        CALL TOADS ("WAIT",360)
BLU    $DELAY
```

1.9 $HOLD

The Hold service suspends a program from execution until released by the operator. Prior to the suspension a message can be written from the program to the operator terminal. The assembly language call is:

```
                TMK         PARLIST
                BLU         $HOLD
                ...
PARLIST         DATA        'nnaaaaaa
MES             DATA        "message-text"
```

The size of the message in words (three characters per word) must be specified in octal by "nn", and the address of the message must be specified in octal by "aaaaaa". The actual text of the message is enclosed in quotes. For example:

```
                TMK         PARLIST
                BLU         $HOLD
                ...
PARLIST         '05         MES
MES             DATA        "EXAMPLE MESSAGE"
```

writes to the operator terminal the program name followed by EXAMPLE MESSAGE and then suspends the program.

The FORTRAN call for the $HOLD service is:

```
                PAUSE       message-text
```

CHAPTER 2
FORMAT SCANNER

VULCAN has as a non-resident handler, a standardized format scanner. This scanner is used by Job Control, Operator Communications, Vulcanizer, other system routines, and user programs.

2.1 DEFINITIONS

The following definitions apply to all scanner services except where specifically noted.

2.1.1 Scanner Services Types

There are three types of scanner services: control services, single character services, and parameter services. Control services do not scan characters or parameters but control the operation of the scanner. Single character services return only a single character. Parameter services can potentially return one or more characters, depending on the text.

2.1.2 Delimiters

All blanks and commas are counted as delimiters for parameter services unless enclosed in quotes. A string of blanks is equivalent to one blank. Blanks after or preceding a comma are ignored. If a parameter is followed by one or more spaces and then a comma, then the comma is considered to be the true delimiter of the parameter. Other delimiters may be set by the $STCHAR or $DLIM functions; these delimiters function the same as commas.

Examples:

Text Scanned	Parameters	True Delimiter of A
A,B	A and B	comma
AøB	A and B	space
AøøB	A and B	space
Aø,øB	A and B	comma
A,,B	A,null,and B	first comma
Aø,ø,øB	A,null,and B	first comma

2.1.3 Quotes

Double quotes (") may be used to enclose a string of characters, including delimiters. Delimiters are not usually significant in a string. An exception to this is that a * is still significant when a string of the format qualifier*areaname is scanned by the function $AREANM. Numeric digits are treated as text if they are within a string, and an error return will occur if they are scanned by a number function. If a double quote is desired in a string, it should be entered as a pair of double quotes.

Examples:

Parameter Scanned	Actual Argument	
,ABCD,	ABCD	
,A''''BCD,	ABCD	null string
,"A""BCD",	A"BCD	
,A"""""BCD,	A"BCD	
,"AøB",	AøB	
,"A,B,C",	A,B,C	
,"","",	","	one parameter
,"","","",	" and "	two parameters

2.1.4 Displacement

Displacement represents the current scan position and is normally equivalent to the column number — i.e., the first character in the buffer is at displacement zero, the second character is at one, etc. When registers or quotes are present, the displacement is affected as shown in the following examples:

Displacement Text	012 345 AB#N1,CD	A numeric register is counted as one displacement.
Displacement Text	012 678 AB#R1,CD	If #R1 is a replacement or string register containing the value XYZ, the characters in the register are counted for displacements 3,4, and 5.
Displacement Text	01 2 34 5 A,"""","""",	When quotes are present, displacement is calculated as if only the actual argument was present. See the above discussion of quotes.

The scan displacement after a call varies among the scanner services. For single character services, normally only $CHAR will change the displacement. However, if a numeric register name, a bad register name, or the name of an undefined register is scanned by $CHAR or $NXCHAR, the displacement is set to the first character following the register name. For parameter services, the scan is positioned to pick up the next parameter; one character past the true delimiter of the last parameter scanned. After an illegal parameter error return, the scan position is undefined. When an error return occurs from a parameter service, a call to $BKPARM may be used to position the scanner to reexamine the last parameter.

2.1.5 End of Line

The scanner returns an end of line status whenever the end of buffer is found or when a space-dollar-space sequence is found. For example, if the following line were being scanned:

```
AA,BB $
      ↑
```

no scanner function will move the displacement beyond the position pointed to by the arrow; rather, each call would result in an end of line error return.

2.2 NUMBERS

All numeric values are assumed to be decimal unless they are preceded by a single quote, which indicates they are octal. Negative numbers are preceded by a minus sign. A number range may be specified by placing a dash between the numbers; the second number in a range must not be smaller than the first.

Number Scanned	Decimal Value
77	77
'77	63
-77	-77
-'77	-63
1-3	Base of 1, range of 2
'4-14	Base of 4, range of 8
-1-3	Base of -1, range of 2

2.3 QUALIFIERS AND AREA NAMES

Standard VULCAN disc area qualifiers and area names may be input in truncated ASCII with $QUAL and $AREANM respectively. A qualifier is of the form "nnnnxxxx" where "nnnn" are 1 to 4 digits specifying an account number, and "xxxx" are 1 to 4 characters, starting with an alphabetic character, specifying an identifier. Leading zeros are prefixed to the account number as needed. Trailing blanks are suffixed to the identifier as needed. The format of an area name is "qualifier*areaname", where "areaname" is 1 to 8 characters of which the first is alphabetic.

Examples:

Input	Qualifier	Areaname
1234ABCD*XYZ	1234ABCD	XYZ
1A*XYZ	0001A␣␣␣	XYZ
XYZ	user's default qualifier	XYZ
*XYZ	0000SYST	XYZ
"A␣B␣*XYZ"	0000A␣B␣	XYZ
ABCDEFGHI	error (too many characters)	
ABCDE*X	error (identifier is too long)	

2.4 REGISTERS

Job Control registers (software registers) should not be confused with hardware registers. Job Control registers are local to an individual user and to a particular terminal session or control point job. During any session or job, user registers are available to any processor invoked.

2.4.1 Register Usage

Whenever a # sign is scanned, the characters immediately following the # sign, until the next delimiter, are treated as a register name, except in the following situations:

- If the # is within a quoted string or string register
- If the program calling the scanner is a real time or monitor program, in which case, registers are not allowed.
- If the user has turned register mode off using the Job Control command $MO,RG=OFF

A register name consists of a # followed by from 1 to 3 letters or decimal digits in any order. If more than 3 characters are used, the excess is ignored. A $ sign is used as a special delimiter which delimits a register name but not its contents. Any character other than a letter or a decimal digit is treated as a delimiter of the register name.

When a register specification is encountered by the scanner, the contents of the register are used in place of the register specification. The register type and register contents must both satisfy the specifications of the scanner function which scans the register. For example, a $NUMBER request on a register containing "ABC" will return an error condition.

Registers may be defined through the $STREG scanner function as well as by the Job Control command SR (Set Register). Registers may also be manipulated through the use of the $GTREG scanner function and by the Job Control commands RR (Remove Register) and DR (Display Register). The maximum number of user registers may be defined by the Job Control MODE command, MO,NR = nn where nn is the number of user registers.

2.4.2 Types of Registers

There are three types of registers: numeric, string, and replacement. Numeric registers are, of course, used for numbers. String registers are used for text. Replacement registers may be used for either numbers or text.

In scanner services where a register type must be specified, use the register type number for a register as shown in Table 2-1.

Table 2-1. Table of Register Types

Register	Register Type Number
String register	1
Numeric register	2
Replacement register	3

2.4.2.1 Numeric Registers

Numeric registers may contain only a numeric value or range and return a valid parameter only for the functions $NUMBER, $ONENUM, $SPNUMB, $TEXNUM, or $NUMTEX.

Examples:

Text	Numeric Register Value	Function	Equivalent Text
,#N1,	#N1=77	$NUMBER	,77,
,-#N1,	#N1=77	$NUMBER	,-77,
,#N1$-#N2,	#N1=7 #N2=10	$NUMBER	,7-10,
,#N1,	#N1=7-10	$NUMBER	,7-10,
,#N1,	#N1=7-10	$ONENUM	Error. Cannot have range.
,#N1,	#N1=77	$TEXT	Error. Not legal for $TEXT
,AB#N1,	#N1=77	$TEXNUM	,AB77,
,#N1$AB,	#N1=77	$NUMTEX	,77AB,

2.4.2.2 Replacement Registers

The contents of a replacement register are literally substituted in place of the register name. Replacement registers may contain more than one argument, they may be nested, they may contain other register names including other replacement registers, or they may be null, in which case they are ignored by the scanner. Note: certain undefined conditions may result by having a quote character be the last character of a replacement register or having a quote be the first character following a replacement register.

Examples:

Text	Replacement Register Value	Equivalent Text
ABC#R1,DE	#R1=123	ABC123,DE
AAA#R1ϕBBB	#R1=,ϕ#R2 #R2=123	AAA,ϕ123ϕBBB
#R1$#R2	#R1=AAA #R2=BBB	AAABBB
#R1#R2	#R1=AAA #R2=BBB	AAABBB
#*		Error, Bad register name

2.4.2.3 String Registers

A string register is substituted in the same manner as a replacement register with an implied set of quotes around the contents. Double quotes (") in a string register are scanned as normal characters; i.e., they are not interpreted in any special manner. Scanning a string register causes an invalid parameter error return from any function which scans a numeric value, since numeric values within a string are treated as text. Delimiters and # signs have no special meaning in string registers. String registers may be null, in which case they are ignored by the scanner.

Examples:

Text	String Register Value	Function	Returned Value
#S1,	#S1=ABC	$TEXT	ABC
#S1,	#S1="ABC"	$TEXT	"ABC"
#S1,	#S1=""""	$TEXT	""""
#S1,	#S1=3	$NUMBER	Error. Invalid parameter
AAA#S1ϕBBB	#S1=ϕ#S2 #S2=123	$LTEXT	AAAϕ#S2ϕBBB

2.5 RETURNS FROM SCANNER SERVICES

This section gives a general description of the returns from the scanner services. See the documentation of the individual services for additions or exceptions.

Hardware registers:

I=	Pointer to user temporary space.
J=	Saved (return address).
K,E=	Service return values, as documented.
A=	Error code if negative, or function return value, as documented.
V=	Not saved, or delimiter code, as documented.
H=	Saved.

Register A error codes:

-1=	Invalid parameter, bad register name, undefined register, or inappropriate numeric register.
-2=	EOL; end of buffer or space-dollar-space.
-3=	Null parameter.

The delimiter code is returned in register V by parameter services only. The code represents the true delimiter of the parameter, not necessarily the delimiter immediately following the parameter. The delimiter codes are:

0	blank
1	comma
2	EOL
3	First special delimiter (from $DLIM or $STCHAR)
4-14	2nd through 12th special delimiters (from $DLIM)

2.6 SCANNER CONTROL SERVICES

2.6.1 $SCINIT

The Scan Initialize service is used to initialize the scanner and to pass a buffer to be scanned. Only one buffer may be processed by the scanner at a time. The calling sequence is:

```
        TLO     PARLIST
        BLU     $SCINIT
```

where the parameter list is a two word list:

```
PARLIST DATA    buffer-length-in-words
        DAC     buffer-address
```

There are no error returns from the $SCINIT service.

2.6.2 $GTHEAD

The Get Header service captures and saves the major elements of the current scanner state. Using the $GTHEAD service in conjunction with the $STHEAD service enables the user to concurrently scan multiple buffers.

$GTHEAD saves the buffer pointers and any special delimiters set by $STCHAR or $DLIM. Note that the user scan state as set by $STMODE is not saved.

$GTHEAD copies four words of information into a user-supplied buffer. If no special delimiters are in effect, the last word will contain a zero. If special delimiters have been set, the last word of the buffer will point to a small block of DCM (presently six words but subject to change) allocated by the service on behalf of the user. If, for any reason, the

service is unable to allocate the DCM, the last word will contain a minus one indicating special delimiters were not saved. The calling sequence is:

```
TLO     PARLIST
BLU     $GTHEAD
```

where PARLIST is the address (label) of the first word of the four-word user-supplied buffer.

An example showing how to use $GTHEAD is provided following the description of the $STHEAD service.

2.6.3 $STHEAD

The Set Header service restores the major elements of the scanner state as captured by the $GTHEAD service. The calling sequence is:

```
TLO     PARLIST
BLU     $STHEAD
```

where PARLIST is the address (label) of the four-word user buffer utilized by the corresponding $GTHEAD call.

EXAMPLE:

To save the scanner state, re-initialize to another buffer, process the new buffer and then resume scanning the original buffer, use the sequence:

```
        TLO     BUF
        BLU     $GTHEAD         save pointer.
        TLO     PL              initialize to
        BLU     $SCINIT         new buffer.
        ...     process new buffer
        TLO     BUF             restore original
        BLU     $STHEAD         pointers.
        ...     continue scanning first buffer
BUF     BLOK    4
PL      DATA    word-count
        DAC     buffer-address
```

Note that the $GTHEAD and $STHEAD services allocate DCM to store extended scanner states. There are, however, circumstances under which the scan state cannot be properly restored. They include:

1. buffers on which the scan mode has been changed by the $STMODE service in the middle of scanning the buffers.
2. buffers on which the $REGNAM service has been called.

2.6.4 $GTDISP

The Get Displacement service enables the current displacement to be saved, thus, allowing a return to this position. The calling sequence is:

```
BLU          $GTDISP
```

The current displacement is returned as a numeric value in register K.

2.6.5 $STDISP

The Set Displacement service allows a new value to be set as the current scanner displacement. This is normally a value from a previous $GTDISP call but may be any displacement within the range of the buffer. The calling sequence is:

```
TMK          displacement
BLU          $STDISP
```

For example, to reset the scanner to the start of the current buffer, set the displacement to zero. A return with A=-2 means the displacement is past the end of the buffer.

2.6.6 $NXPARM

The Next Parameter service skips over redundant blanks and sets the scan position so that the next character picked up is the first non-blank encountered. If the current displacement is already at a non-blank character, no change is made. The calling sequence is:

```
BLU          $NXPARM
```

The returned register settings are like those for $CHAR but the E register is undefined. The A register is set to -1 if a numeric register is encountered.

2.6.7 $BKPARMA

The Back Parameter service resets the scan position back to the start of the most recently input parameter. This refers only to parameters input by $NUMBER, $SPNUMB, $NUMTEX, $TEXNUM, $TEXRAN, $ONENUM, $TEXT, $LTEXT, $AREANM, and $QUAL. Any calls to other scanner services will not affect the buffer displacement used by $BKPARM. For example, following a call to $TEXT, it is determined that the parameter was greater than 6 characters. To obtain the entire parameter. a call to $BKPARM, then a call to $LTEXT may be made. The calling sequence is:

```
BLU          $BKPARM
```

2.6.8 $STCHAR

The Set Character service allows a special delimiter character to be given to the Format Scanner. This delimiter is in addition to the standard blank or comma delimiter. A call to $STCHAR will replace any delimiters set by a previous call to $DLIM or $STCHAR.

The calling sequence is:

```
TOK     "character"
BLU     $STCHAR
```

For example, if the following call were made:

```
TOK     "="
BLU     $STCHAR
```

then the equal sign (=) would also delimit parameters. Thus the input text:

```
AB=CD
```

would be input as the two parameters AB and CD.

To remove all special delimiters with a call to $STCHAR, set the special delimiter to blank (" ").

2.6.9 $DLIM

The Delimiter service allows multiple special delimiters to be given to the format scanner. This service performs the same function as $STCHAR, except that more than one delimiter (up to a maximum of 12) may be defined. A call to $DLIM will overwrite any delimiters set by a previous call to $DLIM or $STCHAR. The calling sequence is:

```
        TLO     PARLIST
        BLU     $DLIM
        ...
PARLIST DATA    number-of-special-delimiters
        DATA    list-of-delimiters
```

where number-of-delimiters is 0 through 12. If it is zero, all previously entered special delimiters are removed. List-of-delimiters is 1 to 4 words of delimiters, packed left-justified in any partially filled words.

for example, to set "A", "B", "C", ".", "X", "Y" and "Z" as special delimiters:

```
        TLO     PL
        BLU     $DLIM
        ...
PL      DATA    7
        DATA    "ABC.XYZ"
```

2.6.10 $STREG

The Set Register service is used to alter register contents. This service will always perform the specified function regardless of the "Register Mode" flag (see the Job Control $MODE command). This service can be used to generate or eliminate registers.

On return the A register will be set:

A =	0	Function performed as specified
	-1	Invalid register name syntax
	-2	Too many registers currently defined
	-3	Illegal type number
	-5	Byte count greater than 255 or less than 0.

In all cases, if the register previously existed, the old value is eliminated prior to generation of the new value.

```
          TLO     PARLIST
          BLU     $STREG
          ...
PARLIST   DATA    type
          DATA    register-name
          DATA    value-1
          DATA    value-2
```

where type is 1 for a string register, 2 for a numeric register, or 3 for a replacement register (see Table 2-1). Register-name is any valid register name. The appropriate substitution for value-1 and value-2 is based on type. For a string or replacement register, value-1 must be the byte count length of value-2 which is text; a zero-length byte count denotes a null register. For a numeric register, value-1 must be the range and value-2 the base of the numeric value.

To eliminate a register, use the following calling sequence:

```
          TLO     PARLIST
          BLU     $STREG
```

where PARLIST is a two word parameter list:

```
PARLIST   DATA    0
          DATA    register-name
```

If register-name is zero, then all registers will be eliminated. On return the A register will be set to zero if the function was completed as specified, or -1 if the specified register did not exist.

2.6.11 $GTREG

The Get Register service is used to obtain register information or contents. The calling sequence is:

```
          TLO     PARLIST
          BLU     $GTREG
          ...
PARLIST   DATA    function
          DATA    register-name
          DATA    buffer-length
          DAC     buffer-address
```

See Table 2-1 for a list of register types.

Depending on function, some of the parameters are not used. When used, register-name is a standard register name and buffer-length is the length in words of the supplied buffer at buffer-address. If an invalid function number is supplied, the A register will be set to -3 on return. The following functions are available:

Get Register List — A list of all active registers may be obtained when function has a value of 1. The register-name is used to indicate the starting point for copying register information. If the register-name is zero, the list will start with the first entry. If the register-name is not zero, the list will start with the first register after the specified name. For each active register, three words will be copied to the supplied buffer:

Word 0 = Register type (1, 2 or 3)
Word 1 = Register name
Word 2 = Byte count (meaningless if type = 2)

On return, the A and E registers will be set as:

A	=	≥0	Word count actually transferred
		-1	Register-name not in list
		-2	Buffer is too small (as much as possible will be transferred)
E	=		Total number of active registers

Register Information — Information about a single register may be obtained when the function has a value of 2. No buffer need be supplied for this function. On return, the A, E and K registers will be set as:

A	=	0	Function performed as specified
		-1	Register name not in list, E and K undefined
E	=		Byte count (meaningless if type = 2)
K	=		Register type (1, 2 or 3)

Register Contents — Information and contents of a single register may be obtained when function has a value of 3. This function is like function 2 except that the contents of the register are returned in the supplied buffer. The layout of the buffer is identical to the PARLIST used to define the register with $STREG. If the buffer is too small for the register contents, the A register will be set to -2. Otherwise, the A register will be defined as for function 2. If the register is found, the E and K registers will always be set to the byte count and register type, respectively.

2.6.12 $STMODE

The Set Mode service is used to set special scan modes for all calls to the scanner until the modes are reset. The calling sequence is:

```
TOK             modes
BLU             $STMODE
```

where bits set in register K determine which special scan modes are active. These bits are defined as follows:

Bit 0 Delimiter mode. If reset, the true delimiter of a parameter function is as defined in Section 2.1.2 of this manual. If set, the first delimiter found is considered to be the true delimiter of a parameter.

For example, if "PRbbb,F" is scanned by a parameter function with Bit 0 set, the scan displacement will be PRbbb,F. If it is scanned with Bit 0 reset, the scan displacement will be

PRbbb,F.

Bit 1 Register mode. If reset, a "#" is treated as the start of a register specification, as described in Section 2.4 of this manual. If set, then register mode is turned off; a "#" is treated as a normal character. This is useful, for example, if the name of a register is desired instead of its contents.

A call to $STMODE resets any previous modes set by this service. To clear all special modes, set the K register to 0 when calling $STMODE. A call to the $SCINIT service also clears all special modes.

There are no error returns from $STMODE.

2.7 SINGLE CHARACTER SERVICES

2.7.1 $CHAR

The Character service obtains the next character in the buffer, regardless of parameters, delimiters, etc., and then increments the displacement. The calling sequence is:

```
BLU          $CHAR
```

The normal register settings are returned, with the following additions:

A= positive if no errors. Bits 7-0 are the ASCII character, bits 23-8 are zero.

E= 0 or bit 23 is set if the character is within a string

K= the current scan displacement, i.e., of the next character.

2.7.2 $BKCHAR

The Back Character service picks up the preceding character in the buffer (i.e., the last character scanned).

The ASCII character is returned in bits 7-0 of the A register, with the rest of the register set to zero. If the buffer has just been initialized, and there is no previous character, then the A register is set to -2.

The buffer displacement pointer is not modified by this call. Hence, successive Back Character calls will not backspace through the buffer but will merely return the same character. The calling sequence is:

```
BLU          $BKCHAR
```

2.7.3 $NXCHAR

The Next Character service functions like $CHAR except that the buffer displacement pointer is not modified but is returned in the K register.

The ASCII character picked up is returned in bits 7-0 of the A register, with the remainder of A set to zero. If the end of the buffer is reached, A is set to -2. The calling sequence is:

```
BLU     $NXCHAR
```

The following example is a sequence of calls demonstrating the function of the character input routines just described. The underscore indicates the next character to be picked up.

Buffer before Call	Call	Character returned
A̲BCD	BKCHAR	none : (-2)
A̲BCD	NXCHAR	A
A̲BCD	CHAR	A
AB̲CD	CHAR	B
ABC̲D	BKCHAR	B
ABC̲D	BKCHAR	B
ABC̲D	CHAR	C
ABCD̲	NXCHAR	D

2.8 PARAMETER SERVICES

2.8.1 $NUMBER

The Number service is used to input a numeric parameter. The buffer is scanned for the next argument and, if numeric, the binary value is returned to the calling program. The calling sequence is:

```
BLU        $NUMBER
```

Upon return, the A register is set either to the binary value or to an error code:

A	=	≥0	Binary value scanned
		-1	Wrong type of argument was found
		-2	End of buffer
		-3	Null argument (default case)
		-4	Negative number encountered, value in K register, E is still length of range

The E register is set to the range of a pair of numbers. (Note: E can be zero if only one number is input, or if the range is zero, e.g., (7-7). If the user must differentiate between these two cases, the user should first use the $ONENUM service, followed by the $NUMBER service.)

Given the following text inputs, the register values returned after a $NUMBER call would be:

Text	A after call	E after call
,32,	32	0
,32-32,	32	0
,32-41,	32	9
,'32-41,	26	7
,32A,	-1	unspecified
,'32,	26	0
,'32-'41,	-1	unspecified
,,	-3	unspecified
,"32",	-1	unspecified
,-32,	-4 (K= -32)	0
,-30-32,	-4 (K= -30)	2
,32"	-1	unspecified

2.8.2 $ONENUM

The One Number service is useful if a group of numbers would be an incorrect argument. The returns from the $ONENUM service are identical to those produced by $NUMBER except that the A register is set to -1 if a group of numbers is encountered. In this case, the K register is set to the first number of the group. The calling sequence is:

```
BLU        $ONENUM
```

Given the following text inputs, the register values returned after a $ONENUM would be:

Text	A after Call	E after Call
,32,	32	0
,32-32,	-1	unspecified
,32A,	-1	unspecified

2.8.3 $SPNUMB

The Special Number service is used to input a number with embedded blanks. The service is identical to the $ONENUM service except that embedded blanks are ignored, and the scanning is terminated by any non-blank non-digit character. The calling sequence is:

```
BLU     $SPNUMB
```

The return conditions are identical to those of $ONENUM. In addition, the V register can be set:

V='177777 If the number was delimited by a character.
V='177776 If the number was delimited by a quoted string.

Examples:

Text	A after Call
,32b,	32
,3b2,	32
,32A,	32 (pointer set to "A")
,1A2B,	1 (pointer set to "A")
,b3b2bbA,	32 (pointer set to "A")

2.8.4 $NUMTEX

The Number-Text service returns the numeric and non-numeric values of an input text. The calling sequence is:

```
BLU     $NUMTEX
```

Upon return, the A register is set to the numeric value found, and the E register is set to the first three characters found after the numeric portion of the text, and preceding any delimiters.

The numeric portion of the text may be in a numeric register. Negative numbers are handled in the same manner as with the $NUMBER service. In an error condition the normal error codes are returned in the A register.

For example, assume the indicated text is scanned using the $NUMTEX service:

Text	A after Call	E after Call
,32AB,	32	ABb
,3A,	3	Abb
,3ABCD,	3	ABC
,3,	-1	unspecified
,A,	-1	unspecified
,1-3AB,	-1	unspecified
,A3,	-1	unspecified
,,	-3	unspecified

2.8.5 $TEXNUM

The Text-Number service is the reverse of the $NUMTEX service. Non-numeric characters are expected followed by an uninterrupted string of digits until a delimiter is encountered. A numeric register may be used for the numeric portion of the text. The calling sequence is:

```
BLU         $TEXNUM
```

Upon return, the E register is set to the first 3 non-digit characters found, and the A register is set to the integer value following the text. In case of an error, the A register is set to the normal error codes. Negative numbers are handled in the same manner as for the $NUMBER service.

For example, assume the indicated text is scanned using the $TEXNUM service:

Text	A after Call	E after Call
,AB3,	3	AB⌀
,ABCD32,	32	ABC
,A3,	3	A⌀⌀
,3A,	-1	unspecified
,A,	-1	unspecified
,3,	-1	unspecified
,A-1,	-4 (K=-1)	A

2.8.5A $TEXRAN

The Text-Range Service returns a text string and a range of numbers following the text string. The service expects a string of non-numeric characters followed by a number or a

range of numbers. The rules for the numeric range are given for the $NUMBER service. The calling sequence is:

```
BLU         $TEXRAN
```

Upon return, the K register is set to the first 3 non-numeric characters found, the A register is set to the first integer value following the text, and the E register is set to the range of the pair of numbers. Note: this service does not support a negative range. In the case of an error, the A register is set to the normal error codes.

For example, assume the indicated text is scanned using the $TEXRAN service:

TEXT	A after call	E after call	K after call
,A5-6,	5	1	A∅∅
,A32,	32	0	A∅∅
,ABCD5-6,	5	1	ABC
,3A,	-1	unspecified	unspecified
,A,	-1	unspecified	unspecified
,3,	-1	unspecified	unspecified
,A-1,	-1	unspecified	unspecified
,A-1-2,	-1	unspecified	unspecified
,A'6-7	6	1	A∅∅

2.8.6 $TEXT

The Text service picks up the first 6 characters in the next argument and returns them in the E and K registers. For less than 6 characters, E and K are blank filled with the text left justified, first in E, then in K. If the text is longer than six characters, only the first six characters are returned in E and K; however, the scan is positioned after the true delimiter of the entire text as if the entire parameter were scanned. The A register is set to the number of characters in the argument just encountered. The calling sequence is:

```
BLU         $TEXT
```

Upon return, the E register has the first three characters picked up, the K register has the second three picked up, and the A register has the number of characters encountered. In case of error, the A register is set to the normal error codes.

The following examples show the results of $TEXT calls on the given texts.

Text	A	E	K
,ABCDEF,	6	ABC	DEF
,AB,	2	AB∅	∅∅∅
,A,	1	A∅∅	∅∅∅
,,	-3	unspecified	unspecified
,ABCDEFGH,	8	ABC	DEF
,''''	-3	unspecified	unspecified

2.8.7 $LTEXT

The Long Text service is the same as the $TEXT service except that a parameter list is supplied which enables an argument longer than six characters to be returned to the user. The calling sequence is:

```
TLO         PARLIST
BLU         $LTEXT
```

where PARLIST is a two word parameter list:

```
PARLIST     DATA      buffer-word-count
            DAC       buffer-address
```

On return, the A register is set to the number of characters picked up. If this is greater than the buffer size, the supplied buffer will hold only those that can be held. Otherwise, the A register is set to the number of characters stored in the buffer. The remainder of the buffer is blank filled.

In case of error, the A register is set to the normal error codes.

If the following call were made:

```
            TLO       PL
            BLU       $LTEXT
            ...
PL          DATA      3
            DAC       BUF
BUF         BLOK      3
```

then the contents of BUF given the sample text would be:

Text	A	BUF
,ABCDEFGHI,	9	ABC/DEF/GHI
,ABCDEFG,	7	ABC/DEF/Gbb
,AB,	2	ABø/øøø/øøø
,ABCDEFGHIJ,	10	ABC/DEF/GHI
,,	-3	øøø/øøø/øøø

2.8.8 $AREANM

The Areaname service is used to input disc area names, which consist of a qualifier and an areaname. The results are stored in a 4-word user supplied buffer, with the 8-character areaname in truncated ASCII (6-bit characters) occupying the first two words, and the 8-character qualifier occupying the last 2 words. The calling sequence is:

```
            TLO       PARLIST
            BLU       $AREANM
```

where PARLIST is a user buffer at least 4 words long which will contain the results of the scanner input. On return the A register will be set to be the total number of characters picked up by the call. In case of an error, A is set to the normal error codes. The E register is set to the total number of characters specified for the qualifier.

The first character of all areanames must be alphabetic. The first four characters of the qualifier are the account number. The $AREANM service will right justify the account digits, zero fill the qualifier, and left justify and blank fill the areaname. If no qualifier is present in the argument, the default (sign-on) qualifier is returned. An asterisk (*) must be used to delimit the qualifier from the areaname.

Given the specified input text, with the user signed on with the qualifier 1234USER, a $AREANM call will produce:

Text	PARLIST after Call
,1234ABCD*XYZ,	XYZb/bbbb/1234/ABCD
,12AB*XYZ,	XYZb/bbbb/0012/ABbb
,*XYZ,	XYZb/bbbb/0000/SYST
,XYZ,	XYZb/bbbb/1234/USER
,"*LG 71"	LGbb/bb71/0000/SYST

2.8.9 $QUAL

The Qualifier service is used to pick up an argument that is a qualifier. This is effectively the same service as the first half of the $AREANM service. A 2-word buffer must be supplied by the user to store the argument. The calling sequence is:

```
TLO     PARLIST
BLU     $QUAL
```

The qualifier will be returned into PARLIST and PARLIST+1. Up to eight characters will be picked up and returned in truncated ASCII. The A register will be set to the number of characters scanned for this argument. For error conditions the A register will be set to the normal error codes.

For example, assume the following inputs were scanned with the $QUAL service:

Text	PARLIST after Call
,1234ABCD,	1234/ABCD
,12AB,	0012/ABbb
,1A,	0001/Abbb
,A,	0000/Abbb
,1,	(error)
,12345A,	(error)

2.8.10 $REGNAM

The register name service scans the input buffer for a register name. The format is:

```
BLU     $REGNAM
```

Upon return the hardware registers are set as follows:

A: 0 Successful return.
-1 Wrong type of argument.

E: First three ASCII characters of the register name (with trailing blanks filled).

All other hardware registers are set to the normal values.

Prior to calling the $REGNAM service, the scanner must be positioned prior to or at the # character of the register name. For example:

```
A␣␣␣,␣␣␣#REG
  ↑
A␣␣␣#REG
  ↑
```

Care must be taken when scanning the parameter prior to the register name. In certain instances the scan position may inadvertently be placed inside the register; for example,

```
A␣␣␣#REG
```

where A is scanned via a BLU $TEXT and #REG contains "CDE". The scan displacement is positioned at the "C" rather than at or prior to the "#". In this situation the $STMODE service may be used to turn the register mode off prior to scanning the parameter.

2.8.11 $ASNOBJ and $ASNLST

$ASNOBJ and $ASNLST are system services which scan a field that is a description of an I/O entity and which return a parameter list that may be used in a BLU $ASSIGN call. The format of the I/O entity for each service is as follows:

Service	I/O Entitiy
$ASNOBJ	assign-object (attribute-list)
$ASNLST	lfn=assign-object (attribute-list)

where "lfn" is the logical file number to be assigned, "assign-object" is the object of the assign (see Section 2.8.11.1), and "attribute-list" is an optional list of attributes enclosed within parentheses (see Section 2.8.11.2). The "assign-object" is optional for the $ASNLST service.

The calling sequence is:

```
        TLO     PARLIST
        BLU     service
        BNZ     error-check
        TLO     BA
        BLU     $ASSIGN

PARLIST DATA    bit mask of allowable objects
        DATA    bit mask of allowable attributes
        DATA    word count
        DAC     BA
BA      BLOK    word count
```

where "service" is $ASNOBJ or $ASNLST. The minimum buffer length for $ASNOBJ and $ASNLST is defined through the AS!.MIN EQIV to be 12 words.

Standard VULCAN EQIVs are listed in the bit mask descriptions in the following sections.

2.8.11.1 Bit Mask of Allowable Objects

Bits set in the bit mask of allowable objects permit specific assign objects. An error return will occur if any object is scanned that is not allowed by this mask. The bits are defined below.

Bit	EQIV	Object Allowed**	Description
B0	AS!SEL!	lfn=*	Assign to self. An "lfn=*" is the same as an assign to *3. A -10 is returned as the value of the A register.
B1	AS!IFA!	lfn=*lfn	Specifies an indirect follow assign.
B2	AS!IPA!	lfn=%lfn	Specifies an indirect permanent assign.
B3	AS!PDV!	lfn=:pdn	Assign to a physical device.
B4	AS!SPO!	lfn=@pdn	Assign to a spool file.
B5	AS!CST!	lfn=:pdnTn	Assign to a cassette. The "pdn" is a Harris Model 2200 data terminal with cassette. "T1" designates the left cassette. "T2" designates the right cassette.
B6	AS!REM!	lfn=:pdnxx	Assign to a remote site. The "pdn" is an RJE remote site number and "xx" is one of the following: LP – Remote line printer TP – Remote tape punch CP – Remote card punch PL – Remote plotter CS – Remote operator console
B7-B8			Reserved for future use.
B9	AS!ARE!	lfn=qualifier*area	Assign to a disc area.
B10-B21			Reserved for future use.
B22	AS!NOC!	lfn=pdn lfn=pdnxx lfn=pdnTn	If this bit is set, then the colon (:) preceding the specified PDN for Bits 3, 5, and 6 above may be omitted.
B23			Reserved for future use.

** $ASNOBJ is identical to $ASNLST except that "lfn=" is omitted.

2.8.11.2 Bit Mask of Allowable Attributes

Bits set in this mask allow specific attributes to be present. An error return will occur if any attribute is scanned which is not permitted by this mask. The bits are defined below.

Bits Set	EQIV	Attribute Allowed	Description
B0-B21			Reserved for future use.
B22	AS!EX!	EX	An exclusive (EX) assign allows no more assignments to be made to the object until the object has been freed. An exclusive assign is invalid if an assignment has already been made to the object.
B23			Reserved for future use

2.8.11.3 Parameter List Returned by $ASNOBJ or $ASNLST

$ASNOBJ and $ASNLST return a parameter list in the buffer specified in the calling sequence. The format of this buffer is such that the buffer can be used as the parameter list for a call to the $ASSIGN service. The format of the buffer is as follows.

Word	Bit Within Word	EQIV	Description
0		AS!FUNC	(Each bit of this word is described separately as follows.)
0	0-6	AS!FC!	Contains the $ASSIGN function code.
0	7	AS!FEX!	Is always set and indicates that Word 1 is present.
0	8	AS!IND!	Set only for an indirect permanent assign (lfn=%lfn).
0	9-11		Contains the object of the assignment: 0 – The object is a remote line printer. 1 – The object is either a Harris Model 2200 data terminal (left cassette) or a remote tape punch. 2 – The object is either a Harris Model 2200 data terminal (right cassette) or a remote card punch. 3 – The object is a remote plotter. 4 – The object is a remote operator console.
0	12-21		Reserved for future use.
0	22	AS!AEX!	If set, then attribute EX is present.
0	23	AS!QSP!	If set, then a qualifier is present (for function code 0).

Additional words in the buffer for $ASNLST and $ASNOBJ are described below.

Word	EQIV	Function Code	Description
1	AS!FUNX		Function code extension. Always zero.
2	AS!FLFN		Set to zero for $ASNOBJ. Contains the LFN for $ASNLST.
3-4	AS!FARE	0 or 4	Two-word truncated ASCII disc area name.
3	AS!FPDN	1 or 2	The object, a PDN.
3	AS!F3LF	3	The object, an LFN for an indirect assign.
5-6	AS!FQUA	0 or 4	Two-word truncated ASCII qualifier name.
7-11			Reserved for future use.

2.8.11.4 Return Register Settings from $ASNOBJ and $ASNLST

$ASNOBJ and $ASNLST return the following information in the registers:

Register	Value	Description
A	0	Successful return.
	-1	Wrong type of argument (e.g., incorrect syntax, use of an undefined register, incorrect use of a numeric register, etc.). This error code does not include errors detected within the attribute list.
	-2	End of buffer or "space dollar space".
	-3	Null argument.
	-5	Return buffer too small.
	-6	Invalid attribute scanned. Registers E and K are set.
	-7	Mutually exclusive attributes were scanned. Registers E and K are set.
	-8	Duplicate attributes were scanned. Registers E and K are set.
	-9	Wrong type of argument in the attribute list (e.g., a null argument, a numeric register, or an undefined register was detected). Register K is set.
	-10	The assignment is to self (e.g., "lfn=*" for $ASNLST).
	-12	The object present is not allowed. The appropriate bit in the bit mask of allowable objects was not set.
E		For A register values: 0 and -10, E contains the length in words of the used portion of the $ASSIGN parameter list. For A register values: -6, -7 and -8, E contains the first three characters of the last attribute scanned, left-justified and blank filled. For all other A register values, E is undefined.
I		Temporary pointer.
J		Contains the return address.
K		Contains the number of the attribute at which the error was detected. This register is set only for error codes -6, -7, -8, or -9. Otherwise, Register K is undefined.

V — Contains the delimiter code of the true delimiter of $ASNLST or $ASNOBJ.

H — Saved.

2.8.11.5 Example Using $ASNLST

If buffer INBUF contains the following:

```
AS 6=LO
```

then the following sequence of instructions will perform the requested assignment:

```
        TLO     PARL1
        BLU     $SCINIT         Initialize scanner to scan the buffer
        BLU     $TEXT           Scan the command
        CME     ="AS  "
        BNZ     NOTASSIGN       Branch if not the ASSIGN command
        TLO     PARL2
        BLU     $ASNLST         Scan the LFN and the assign object
        BON     ERROR
        TLO     ASBUF
        BLU     $ASSIGN         Make the assignment
                . . .
PARL1   DATA    27              SCINIT parameter list
        DAC     INBUF
INBUF   BLOK    27                              Command input buffer
PARL2   DATA    AS!IPA!+AS!SEL!+AS!ARE!         These assignments are allowed
        DATA    AS!EX!                          Allow the EX attribute
        DATA    AS!.MIN                         Minimum possible return buffer size
        DAC     ASBUF                           Buffer for $ASSIGN call
ASBUF   BLOK    AS!.MIN
```

2.8.12 $USER

The $USER service is used to input a VULCAN user number. A user number can be either one to six ASCII characters beginning with any alphabetic character or, if all digits, up to a twelve digit number. The calling sequence is:

```
BLU $USER
```

Upon return, the registers are set to the standard values. In addition, the E, A and K registers return as follows:

E= — 24 most significant bits of the user number

K = — 12 least significant bits of user number (left justified). Bits 0-11 are zero.

A = —

0	numeric user number found
positive	number of characters in non-numeric user number
negative	standard error code

2.9 SCANNER EXAMPLE

The following is a scanner example which will produce a $VA (Vassembler) statement in Job Control. The basic format is:

$VA.options,input-areaname

Both the options field and the input-areaname field are optional.

```
                TLO     PL              initialize scanner
                BLU     $SCINIT
                TOK     "."             use "." for delimiter
                BLU     $STCHAR
                BLU     $TEXT
                BON     error
                LRD     8
                COB     "$"             ignore leading $
                BNZ     *-2
                LRD     8
                LRD     16
                TOB     "ø"
                CMA     ="VAø"
                BNZ     error
                BLU     $BKCHAR         test delimiter
                COB     "."
                BNZ     NOOPTS
OPTS            BLU     $CHAR           input an option
                BON     ENDOPTS
                COB     "ø"
                BOZ     NOOPTS          end of options
                ...     process option letter
                BUC     OPTS
ENDOPTS         AOA     2
                BOZ     ENDCARD
                BUC     error
NOOPTS          TLO     NAME
                BLU     $AREANM         input areaname
                BOP     process areaname
                AOA     2
                BNZ     error           no areaname
ENDCARD         ...     continue processing
PL              DATA    27
                DAC     BUF
BUF             BLOK    27              input buffer
NAME            BLOK    4
```

CHAPTER 3
LOGICAL ASSIGNMENT SERVICES

3.1 LOGICAL FILE NUMBERS

The logical assignment services allow the user to control and query LFN assignments. LFNs range from 0-255 (8 bits). LFNs 200-255 are not available to users. Systems processors use LFNs 200-219 on a temporary basis. Job Control uses LFNs 220-225.

LFNs 0-199 are available to users. LFNs 0-10 are also used by Job Control and systems processors and should be assigned with caution. Table 3-1 describes the standard usage of these LFNs and their default assignments.

The COBOL user must avoid assigning to LFNs used by COBOL at execution time. For each COBOL SELECT/ASSIGN clause, COBOL uses one LFN in descending order beginning with 199. For example, if the program contains six SELECT/ASSIGN clauses, the user must not assign LFNs 194-199.

Table 3-1. Standard LFN Usage

LFN	Usage
0	Job Stream. Used to read Job Control commands for Control Points and Terminals.
3	Diagnostic Output. Assigned to disc area LO for control points and to terminal for interactive terminal users. Reassignment will prematurely terminate the job or terminal and should be avoided.
4	Used by KEEP/FETCH routines. Must be assigned to the device or disc area to keep to or fetch from.
5	Binary Output. Used for link code from the assembler, FORTRAN compiler, etc. Default assignment is to disc area LR.
6	List Output. Used by compilers and the assembler to produce listings. Default assignment is to disc area LO.
7	Source Input. Used by compilers and the assembler to read source. Default assignment is indirect to LFN 0 (to read from Job Stream).
8	Scratch Output. Used for intermediate output by Assembler. Default assignment is to disc area W1.
9	Dump Output. Used by Job Control. $DLOAD and $DBOOT commands for dump output.
10	Library File Input. Used by the Library File Edit commands to read their modules to add, replace, etc. Default assignment is to disc area LR.

3.2 $ASSIGN

The system service $ASSIGN is used to assign LFNs to physical devices, disc areas, or other LFNs. It is accessible from Job Control or from user programs using the following assembly language sequence:

```
            TLO     PARLIST
            BLU     $ASSIGN
            ...
PARLIST     DATA    function-code
            DATA    lfn
            DATA    2-word areaname (8 characters-truncated
                    ASCII),
                    or physical-device-number,
                    or logical-file-number for indirect
                    assigns
            DATA    2-word qualifier (8 characters-truncated
                    ASCII) may be used if the areaname is
                    supplied above
```

The function-code DATA word is defined as:

Bits 7-0 = 0 Disc Area Assign. Used to make an assignment to a disc area. The areaname is taken from words PARLIST+2 and PARLIST+3. If bit 23 of the function code word is zero, the sign-on qualifier for the user is used. If bit 23 is set, the qualifier is taken from words PARLIST+4 and PARLIST+5.

If this qualifier field is all zero, then the system qualifier 0000SYST is used. If the qualifier field is truncated ASCII blanks, then the following actions are taken:

Areaname is a workfile:	Sign-on qualifier is used.
Areaname is not a workfile:	Default qualifier is used.

If bit 22 of the function code is set, an exclusive assignment is attempted. An area assigned by another program cannot be exclusively assigned; an exclusively assigned area cannot be accessed by another program.

If the assignment is made to a batch/interactive-terminal work disc area which does not exist, the work area will be dynamically generated.

1 Spool Assign. Used to assign output to spool areas destined for interactive terminals. The PDN of the terminal is specified in PARLIST+2. The terminal need not be allocated as a resource in this case. A spool disc area will be dynamically generated.

2 Physical Device Assign. Used to make an assignment to the physical device specified in word PARLIST+2. If the physical device is an output device only a spool disc area is generated and the assignment is made to it. If the physical device is a card reader, the data input queue is searched for the disc area destined for this purpose.

If the assignment is to device 0, all read requests return an end-of-file and all write requests are accepted but ignored (i.e., output is discarded).

If the specified physical device is a Model 2200 Data Terminal, then bits 9 and 10 of the function code are used to assign to cassettes one and two, respectively. These bits should not be on together.

3 Used to make an assignment to a device or area to which another LFN (the object LFN) is known to be assigned. The object LFN is specified in PARLIST+2. If bit 8 of the function code is not set, then this assignment follows the assignment of the object LFN. If bit 8 is set, this assignment is not changed when the object LFN is reassigned or freed.

Any other value — Invalid call — an abort will result.

Upon return from the $ASSIGN call, the A register will be set to reflect the status of the requested assign, and the condition register is set to match the A register. The A register values may be:

A = 0 — Valid assignment.

1 — Referenced object does not exist. For disc area assigns: the disc area is either not there or is not accessible to this user.

For card reader assigns: no input data file for this program.

For other PDN assigns: referenced device does not exist.

3 — Physical device not ready.

4 — Spool assign allowed only to interactive devices.

5 — Resource not allocated.

For disc area assigns: required disc pack is not resourced or mounted.

For PDN assigns: requested device has not been resourced.

6 — User does not have required access bit to access this physical device.

7 — Batch/interactive work file cannot be generated and does not exist for this user.

8 — Spool pack full or not mounted. Cannot create spool area for spool assign. Cannot create spool area for output physical device assign.

9 — No execute access for program.

10 — Cannot exclusively assign; area in use by another program.

In addition, on disc area assigns, the E register is set:

Bits 23-2	contain disc area type as described in Section IX
Bit 1	is set if the user has read access to the area
Bit 0	is set if the user has write access to the area

Examples of $ASSIGN calls:

1. Assign LFN 8 to physical device 6:

```
                TLO     PARLIST
                BLU     $ASSIGN
                ...
PARLIST         DATA    2
                DATA    8
                DATA    6
```

2. Assign LFN 8 to disc area 1234ABCD*CAT:

```
                TLO     PARLIST
                BLU     $ASSIGN
                ...
PARLIST         DATA    B23
                DATA    8
                DATA    T"CAT␣␣␣␣"
                DATA    T"1234ABCD"
```

3. Assign LFN 8 to cassette two of data terminal PDN 23:

```
                TLO     PARLIST
                BLU     $ASSIGN
                ...
PARLIST         DATA    '2002
                DATA    8
                DATA    23
```

3.3 $PDN

The $PDN service returns information about a physical device. The calling sequence is:

```
                TOK     pdn
                BLU     $PDN
```

Upon return from this call, the A register is set:

A =	0	Specified physical device does not exist.
	1	Specified physical device was zero.
	>1	Physical device type:

Bit 22 set if device is a virtual terminal.
Bit 21 set if device is an interactive terminal.
Bit 20 set if device is spooled input only (e.g., card reader).
Bit 19 set if device is spooled output only (e.g., line printer).
Bit 18 set if device requires resource call to be used.
Bit 17 set if device is connected via CBC channel.
Bits 15-8 contain model information based on particular device type. (See Table 3-2.)
Bits 7-0 contain device type (see Table 3-2).

Table 3-2. Device Types

Type	Device Name	Information in Bits 15-8 of PDN or LFN Service Call.
2	TTY	
3	CRT	Model: 0=Model 8610 or 2310 1=Harris Standard Terminal 2=Model 3270 4=Harris 8680
4	Paper Tape Reader	
5	Paper Tape Punch	
6	Line Printer	Model: 1=Analex 2=Data Products 3=Data Printer 4=Potter 5=CDC 6=Tally 10=Versatec 11=ODEC 12=CDC Serial
7	Card Reader	
8	Card Punch	

Table 3-2. Device Types (Cont'd)

Type	Device Name	Information in Bits 15-8 of PDN or LFN Service Call.
9	Mag Tape Drive or Cassette	Bit: 8=9 Track 10=200BPI 11=556BPI 12=800BPI 13=1600BPI
10	Synchronous Interface for RJE-to "Host" Computer	
11	Remote RJE Line	
12	Remote RJE site	
13	Plotter	
14	Real-Time Peripheral Device	
15	Floppy Disc	

If A > 1 then the E register is set for TTYs, CRTs, and Line Printers as:

Bits 7-0: Number of characters per line
Bits 15-8: Number of lines per page

3.4 $LFN

The $LFN service returns the type of assignment made on a specific LFN. The calling sequence is:

```
TOK     lfn
BLU     $LFN
```

Upon return, the A, V, and C registers are set as follows.

Register	Value	Description
A	-1	LFN is assigned to a disc area.
	0	LFN is unassigned or is incompletely assigned. (See the V register.)
	>0	LFN is assigned to a physical device. In this case the A register is set to describe the physical device type exactly as returned by the $PDN service.
E		If the value of the A register is greater than zero, the contents of the E register are the same as those returned for the E register by the $PDN service.
V	=0	LFN is unassigned.
	≠0	LFN may be indirectly assigned, but the assignment series is incomplete.
C		Reflects the condition of the A register.

If the LFN is assigned to a spooled device, $LFN returns information about the physical device, rather than the spool file.

3.5 $LFINFO

The $LFINFO service obtains information about a particular logical file (LFN) assignment. This information may include a description of the object of an assignment or of the current status of a particular assignment. The calling sequence is:

```
                TLO     PARLIST
                BLU     $LFINFO

PARLIST         DATA    function-code
                DATA    logical file number to query
                DATA    status-word-1
                DATA    0
                DATA    word/byte count of buffer BA
                DAC     BA
BA              BLOK    LF/.LOC
```

The buffer for $LFINFO is defined through the LF/.LOC EQIV to be 12 words; the buffer must be at least 12 words long or the program will abort. The LF! EQIV is defined to be 6 and is the total number of words set aside for the parameter list.

Standard VULCAN EQIVs are listed in the parameter list and object descriptor explanations below.

The parameter list is defined as:

Base Location	Displacement	EQIVs	Description
PARLIST	+0	LF!FC	The function code. Only function codes 0, 4, and 9 are allowed for the $LFINFO service (EQIVs for each function code are given at the left under the column labeled EQIVs):
	+0	LF:LFN	0 – Returns an object descriptor in the buffer.
	+0	LF:STAT	4 – Returns the status of the LFN in status words 1 and 2; however, the buffer address area is not required.
	+0	LF:NEXT	9 – Determines the assignment whose LFN is next greater than the LFN in the LF!LFN word in the parameter list. The service then returns the LFN of that assignment in the parameter list and executes as if function code 0 had been specified. Repeated calls to $LFINFO are intended to sequentially return all assigned LFNs. If the specified word count is 0, the buffer address is ignored and only the LFN and link status are returned (i.e., function code 4 is called rather than function code 0).

Base Location	Displacement	EQIVs	Description
PARLIST	+1	LF!LFN	Contains the LFN for which information is desired. If function code 0 or 4 was specified, this parameter should be the LFN to query. If function code 9 was specified, this parameter should originally be set to -1 on the initial call to the function.
	+2	LF!STT1	Status word 1. Only bit 22 is applicable; all other bits are reserved for future use. If bit 22 is 0, the file is not open. If bit 22 is 1, the file is open. EQIVs for the PARLIST word are: LF!OPN PARLIST word displacement LF!OPN! The bit value (B22) LF!OPN@ The bit number (22)
	+3	LF!STT2	Status word 2. Reserved for future use.
	+4	LF!WC	Word/byte count of the buffer address. Bits 23-22 contain a byte count indicator. If bits 23-22 are 00, then bits 21-0 contain a word count. If bits 23-22 are 01, then bits 21-0 contain a byte count starting from the leftmost byte of the first word.
	+5	LF!BA	The buffer address. This area need not be present if function code 4 was specified.

An object descriptor is returned in the buffer for function code 0. The format of the object descriptor is:

Base Location	Displacement	EQIVs	Description
BA	+0	LF/TYPE	Bits 23-0 contain the device type (as defined for the $PDN service). Zero is returned for disc areas. (D/TYPE EQIVs should be used when interpreting values returned in this word.)
	+1	LF/TYP2	Bits 23-0 provide an additional device type (as defined for the $PDN service). Zero is returned for disc areas. (D/TYP2 EQIVs should be used when interpreting values returned in this word.)
	+2	LF/NTYP	Reserved for future use.
	+3	LF/PDN	Bits 23-0 contain the physical device number (if the device is a physical device or a spooled device).

Base Location	Displacement	EQIVs	Description
BA	+4	LF/AREA	Contains the eight-character truncated ASCII areaname. This field is 0 if the LFN is assigned to a non-spooled PDN.
	+6	LF/QUAL	Contains the eight-character truncated ASCII qualifier.
	+8	LF/REGN	Reserved for future use. Should be ignored.
	+10	LF/SITE	Reserved for future use. Should be ignored.

Note: LF/.LOC is the size required for this buffer.

$LFINFO returns the following information in the registers:

A:
- -1 Means the LFN is not assigned or is invalid (if function codes 0 or 4 were specified).
- -1 Means the end of the LFN list was reached (if function code 9 was specified).
- 0 The LFN was assigned to a disc area.
- 1 The LFN was assigned to a physical device or spooled device.

E: Contents are unspecified (effectively, destroyed).

K: Contains the File Control Block (FCB) pointer.

V: Same contents as Register K.

3.6 $LFNAME

The $LFNAME service is used to find the name of the disc area to which an LFN is assigned. It may also be used to find the physical device number if the LFN is assigned to a physical device. If the LFN is assigned to a spooled device, $LFNAME will return the name of the spool file rather than the PDN. $LFNAME is called by:

```
            TLO     PARLIST
            BLU     $LFNAME

PARLIST     DATA    lfn
            BLOK    3
```

Upon return the A register is:

A =	-1	LFN not assigned.
	0	LFN assigned to a disc area. The areaname is stored in the first and second words; the qualifier is stored in the third and fourth words of PARLIST.
	+1	LFN is assigned to the PDN stored PARLIST+0.

The C register reflects the condition of A.

For example, to find out what LFN 8 is currently assigned to and to assign LFN 9 to that same device or disc area independently of LFN 8 (to work in same file position, use the indirect assign feature):

```
            TOA     8
            TAM     PAR2
            TLO     PAR2
            BLU     $LFNAME
            TME     =B23
            CZA
            BON     not assigned
            BOZ     * + 2
            TOE     2
            TEM     PAR1
            TLO     PAR1
            BLU     $ASSIGN
            ...
PAR1        DAC     0
            DATA    9           LFN for $ASSIGN
PAR2        DAC     0
            BLOK    3
```

3.7 $ASGNM

The Assign Name service returns information similar to that returned by the $LFNAME service but may be used to gather information on many LFNs by repeated calls to the service. LFNs are queried in increasing numeric order. The calling sequence is:

```
                TLO         PARLIST
                BLU         $ASGNM
                ...
PARLIST         DATA        first-lfn
                BLOK        4
```

The next LFN with a number greater than first-lfn will be sought. Upon return, the LFN queried will be in PARLIST+0 and the A register will be set:

A =	-1	No LFNs assigned greater than PARLIST+0.
	0	LFN assigned to a disc area. Areaname stored in second and third words, qualifier stored in fourth and fifth words of PARLIST.
	+1	LFN assigned to PDN stored PARLIST+1.

The C register reflects the condition of A.

CHAPTER 4
BACKGROUND SERVICES

Background services are available only to interactive or control point programs. If called by real-time programs, the information returned is unspecified.

4.1 $NXTPRG

The Next Program service allows existing interactive or control point programs to "chain in" another specific program rather than having Job Control come in by default. The calling sequence is:

```
                TLO         PARLIST
                BLU         $NXTPRG
                ...
PARLIST         DATA        T "areaname-qualifier"
```

The parameter areaname-qualifier must be the 16 ASCII characters of the chained program disc area name. For example, if the disc area name is 1234ABCD*PROGNAME, then this field is T "PROGNAME1234ABCD". There is no return to the calling program from this service. No registers are passed to the called program if it exists. This function performs an exit followed by a program load. See also the $CHAIN service.

4.2 $OPTIONS

The Options service returns to the program the local option word associated with the loading request of the program. These local options are the option letters following the program name on the Job Control statement. The option letters A-X correspond to option bits 0-23 respectively, where bit 23 is the most significant bit in the option word (sign-bit).

The calling sequence for this service is:

```
                BLU         $OPTIONS
```

The local option word is returned in the E register.

4.3 $LINES

The Lines service returns the number of lines per page that has been set either by default or by a Job Control MODE command. This number may be used to control printer page spacing. The calling sequence is:

```
BLU             $LINES
```

The number of lines per page is returned in the E register.

4.4 $UNWORK

The $UNWORK service is used to eliminate all of the work disc areas (such as W1, LR, LO, etc.) for a particular control point or terminal, regardless of their access bits. This may be useful to ensure that no work areas were left over from a previous user. The calling sequence is:

```
BLU             $UNWORK
```

4.5 $DLINES

The Default Lines service returns in the E register the system default lines per page as specified in GENASYS. The calling sequence is:

```
BLU             $DLINES
```

4.6 $LISTDV

The List Device service returns in the E register the system default list output PDN. For jobs transmitted from a remote site, the remote site PDN is returned. The calling sequence is:

```
BLU             $LISTDV
```

where:

areaname — is a potential work file area name. The "areaname" may contain the attached terminal number or control point letter.

qualifier — is the qualifier of the area. Zero implies 0000SYST. A value of truncated ASCII blanks implies the sign-on qualifier.

On return the A, E, C, I, V, and H registers have the following values.

Register	Value	Meaning
A	0	The file is a valid work file for the current process.
	-1	The file has an invalid work file syntax.
	>0	The file name has a valid work file syntax. However, there is not a work file for the current process. In addition, bits are set in the A register to indicate why this work file does not belong to the current process:
		Bit 0: The qualifier is not the sign-on qualifier.
		Bit 1: The work file identification is for an interactive program other than the calling program.
		Bit 2: The work file identification is for a control point program other than the calling program.
		Bit 3: The calling program is a real-time program. Check other bits for work file information.
E		The work file identification for the current process (if the value of the A register is not negative).
C		Reflects the status of the A register.
I		The current temporary area pointer.
V		Saved.
H		Saved.

Note: The work file identification is the last four characters appended to a work file name in truncated ASCII.

4.7 $BKSTOR

The Back-Store service is used to supply a data record to a specified logical file such that a subsequent backspace record and symbolic read will input that record. Most VULCAN processors do a backspace record and symbolic read to obtain the command that initiated them, and this service provides a means of setting this record prior to calling a processor with a service such as $NXTPRG. The calling sequence is:

```
                TLO         PARLIST
                BLU         $BKSTOR
                ...
PARLIST         DATA        'xxxyy
                DATA        word-count
                DAC         buffer-address
```

where xxx is the specified LFN in octal and yy is any octal number. For example, to supply a record to LFN 42, this field should be '05200. The word-count and buffer-address define the record being stored. Upon return from this service, the A register is set to reflect the status of the call:

A	=	0	Record stored as requested
		1	A record has already been stored on this logical file (disc only).
		2	Status call not yet done for specified logical file (disc only).
		3	Specified logical file not assigned.
		4	Record larger than 255 words (disc only).
		5	Invalid device type.

If the specified LFN is not open, the program will abort. Any $I/O function code other than '24-dumpbuffer, '13-open, '00-status, '37-flushbuffer and '21-backspace record will eliminate the backstored record. For information on the $I/O function codes, see the VULCAN I/O Service manual, Harris publication no. 0860004.

4.8 $CHWORK

The Check Work service determines syntactically if a specified area name is that of a valid work file. The calling sequence is:

```
                TLO         PARLIST
                BLU         $CHWORK
                ...
PARLIST         DATA        T"areaname"
                DATA        T"qualifier"
```

CHAPTER 5
REAL TIME SERVICES

5.1 REAL TIME PROGRAM CONTROL

Real Time services are designed to provide the user with a flexible method for controlling real-time programs under VULCAN. The services are designed to affect the operation of real-time programs only. However, most services may be invoked by any program executing under VULCAN. Some, such as $SUSP, $PABORT, and $RSTRT, may only be called by real-time programs and require the user to have abort access, as defined at GENASYS. Additionally Operator Communications provides access to most of these services, allowing the operator to control the real-time system.

5.1.1 Sleep State

All real-time and monitor programs may "go to sleep" using the $SLEEP service. The sleep state is a self-invoked suspension. Once a program has gone to sleep, it remains in the program list in a suspended state until awakened by another program, the operator, a timer schedule activation, or an external interrupt. This provides a means for a program to rapidly go into execution following an external stimulus, rather than waiting for a complete load operation.

5.1.2 Timer Schedule

The Timer Schedule services provide means for activating tasks at specified intervals, or at a specific time. Timer scheduled events are based on the 120 Hertz clock (100 Hertz on 50 cycle power systems), and may be specified in terms of clock ticks. There are 120 clock ticks per second on a 60 cycle power system, 100 clock ticks per second on a 50 cycle power system.

Timer scheduling may be used for two functions: program initiation and program wakeup. The Wakeup and Sleep services are recommended for use with programs that must be activated more often than once every two seconds. These services ensure that a program will be ready when called upon. The Initiate service should not be used with these frequently activated programs because it is impossible to load and unload real-time programs more often than once every two seconds.

Two types of schedule entries are available: temporary and permanent. Temporary schedule requests are lost when the system is booted from disc. Permanent entries remain through system boots (they are kept on a disc area) and will be recalibrated to the current time each time the system is booted. Events scheduled to occur while the system is not operating are ignored.

5.2 $INIT

The Initiate service initiates real-time programs at a specified time and date by loading the program and placing it in execution. This service can also be used to place the initiated program on a timer schedule so that the program will be reinitiated at regular intervals. The specified interval must not be less than two seconds. The calling sequence is:

```
        TLO     PARLIST
        BLU     $INIT
```

where the parameter list is defined as:

PARLIST	+0 +1	8-character program areaname in truncated ASCII.
	+2 +3	8-character program qualifier in truncated ASCII.
	+4	Execution priority. If a -1 is entered, the Vulcanized priority for the program is used. Note, however, that the -1 parameter will cause three additional disc accesses to occur during the execution of the $INIT call. The priority may not exceed the user's priority limit.
	+5	Initiation parameter. Passed to program at initiation and loaded in its K register.
	+6	Days in future from current day to start program.
	+7	Hour of day to start program. *(0-23)
	+8	Minutes of hour to start program. *(0-59)
	+9	Seconds of minute to start program. *(0-59)
	+10	Ticks of clock after specified second to start program (0-119 or 0-99). If these words are negative, the current value in the time-of-day clock is used.
	+11	Period for re-initiation in days.
	+12	Period for re-initiation in hours.
	+13	Period for re-initiation in minutes.
	+14	Period for re-initiation in seconds.
	+15	Period for re-initiation in clock ticks.
	+16	Zero for temporary schedule entry, non-zero for permanent entry

Upon exit from this service, the A register will be set.

A	=	0	Operation performed as requested
		1	Invalid priority
		2	Initiation interval of less than 2 seconds requested
		3	Specified starting time has already passed

Four FORTRAN calls are available to perform the above functions. They are:

1. To initiate a program only once: CALL TOADS ('INITIATE', name, priority, param, istat)
2. To initiate a program only once at some time in the future: CALL TOADS ('FSTART', name, priority, param, dd, hh, mm, ss, tt, perm, istat)
3. To schedule a program for periodic initiation: CALL TOADS ('PSTART', name, priority, param, rdd, rhh, rmm, rss, rtt, perm istat)
4. To schedule a program for periodic initiation at some time in the future: CALL TOADS ('DSTART', name, priority, param, dd, hh, mm, ss, tt, rdd, rhh, rmm, rss, rtt, perm, istat)

The parameters are specified as follows:

name	A 17 character string giving the qualifier and areaname of the program to initiate in standard disc area format.
priority	Priority at which the program will execute; no greater than the user's maximum priority.
param	The initiation parameter which is passed to the program in the K register when it is initiated.
dd	Number of days in the future before the program is to be initiated.
istat	An integer variable set as defined by the A register above.
hh	Time in hours (24 hour clock) when the program is to be initiated.
mm	Time in minutes when the program is to be initiated.
ss	Time in seconds when the program is to be initiated.
tt	Time in ticks when the program is to be initiated.

rdd — Number of days between program re-execution.

rhh — Number of hours between program re-execution.

rmm — Number of minutes between program re-execution.

rss — Number of seconds between program re-execution.

rtt — Number of ticks between program re-execution.

perm — A flag to indicate whether or not the program is to be permanently scheduled. If a program is permanently scheduled, the scheduling information will be retained between re-boots of the system. To indicate that the program is to be permanently scheduled, perm should be set to a non-zero value.

Examples

1. Initiate program 1234ABCD*CAT immediately just once, with parameter -2 at priority 50:

```
            TLO     PARLIST
            BLU     $INIT
            ...
PARLIST     DATA    T"CAT␣␣␣␣␣"
            DATA    T"1234ABCD"
            DATA    50
            DATA    -2
            DATA    0
            DATA    -1, -1, -1, -1
            DATA    0, 0, 0, 0, 0, 0
```

or CALL TOADS ('INITIATE', 17H1234ABCD*CAT␣␣␣␣␣, 50, -2, ISTAT)

2. Initiate program 1234ABCD*CAT at exactly 11:30 A.M. every day after today, using parameter -2 at priority 50.

```
            TLO     PARLIST
            BLU     $INIT
            ...
PARLIST     DATA    T"CAT␣␣␣␣␣"
            DATA    T"1234ABCD"
            DATA    50
            DATA    -2
            DATA    1                   tomorrow
            DATA    11
            DATA    30                  11:30 A.M.
            DATA    0
            DATA    0
            DATA    1
            DATA    0, 0, 0, 0
            DATA    -1                  (permanent)
```

or CALL TOADS ('DSTART', 17H1234ABCD*CAT␣␣␣␣␣, 50, -2, 1, 11, 30,
C 0, 0, 1, 0, 0, 0, 0, -1, ISTAT)

5.3 $WAKEUP

The Wakeup service is used to wakeup or trigger a sleeping program at a specified time and date and with an optional repeating frequency. In all cases, the program must be sleeping for this operation to have any effect. The calling sequence is:

```
        TLO     PARLIST
        BLU     $WAKEUP
```

where the parameter list is the same as for $INIT.

Upon exit from this service, the A register will be set as:

A =	-1	Program not found
	-2	Program not asleep
	0	Operation performed as requested

Four FORTRAN calls are used to access the above functions. They are:

1. Wakeup a program just once: CALL TOADS ('WAKEUP', name, param, istat)
2. Wakeup a program at some time in the future: CALL TOADS ('FWAKEUP', name, param, dd, hh, mm, ss, tt, perm, istat)
3. Schedule a program for periodic wakeup: CALL TOADS ('PWAKEUP', name, param, rdd, rhh, rmm, rss, rtt, perm, istat)
4. Schedule a program for periodic wakeup beginning at some time in the future: CALL TOADS ('DWAKEUP', name, param, dd, hh, mm, ss, tt, rdd, rhh, rmm, rss, rtt, perm, istat)

The parameters are defined as for $INIT.

Example

Wakeup program 1234ABCD*CAT immediately and once every second thereafter:

```
                TLO     PARLIST
                BLU     $WAKEUP
                ...
PARLIST         DATA    T"CAT/////"
                DATA    T"1234ABCD"
                DATA    0
                DATA    -2              parameter
                DATA    0               first wakeup
                DATA    -1, -1, -1, -1  is now
                DATA    0, 0, 0
                DATA    1               1 second
                DATA    0
                DATA    0               (temporary)
```

or CALL TOADS ('PWAKEUP', 17H1234ABCD*CAT/////, -2, 0, 0, 0,
C 0, 1, 0, 0, ISTAT)

5.4 $TERMIN

The Terminate service is used to remove a program entry from the timer schedule. A user may terminate a program only if the user put the program on the timer schedule. Any scheduled program may be terminated from the operator terminal. The calling sequence is:

```
            TLO     PARLIST
            BLU     $TERMIN
            ...
PARLIST     DATA    T"areanamequalifier"
```

where the parameter list contains the program disc areaname and qualifier as 16 ASCII characters.

The entry for the specified program is removed regardless of whether the entry was temporary or permanent. The corresponding FORTRAN call is:

```
CALL TOADS ('TERMINATE', name)
```

Example

Remove program 1234ABCD*CAT from the permanent wakeup list:

```
            TLO     PARLIST
            BLU     $TERMIN
            ...
PARLIST     DATA    T"CAT////1234ABCD"
```

or CALL TOADS ('TERMINATE', 17H1234ABCD*CAT////)

5.5 $SLEEP

The Sleep service is the means by which a program places itself in the sleep state. It will remain in this state until aborted or triggered by a wakeup request by another program, timer schedule, operator, or external interrupt. The E, V, and H registers are preserved through the sleep call. The K register will return the Wakeup parameter passed by the program making the wakeup call. The calling sequence is:

```
BLU        $SLEEP
```

When awakened, execution continues at the instruction following the BLU $SLEEP. The FORTRAN call is:

```
CALL TOADS ('SLEEP')
```

5.6 $DEXIT

The Delay Exit service is a mechanism by which a program exits from the system for a specified time interval, and is then re-initiated. The calling sequence is:

```
TOK        n
BLU        $DEXIT
```

where n is the number of clock ticks to delay before initiating.

When the call is made, the program is entirely unloaded from memory and all allocated resources are removed. When the specified interval has elapsed, a new copy of the program is loaded from disc as in any other program initiation. This service may be used only by real-time or monitor programs.

Example

Exit for 5 seconds and then be reloaded (assume a 120 Hz clock).

```
TOK        600
BLU        $DEXIT
```

5.7 $SUSP

The Suspend service enables one program to suspend another program from execution for an indefinite period of time. A Release Program command from the operator or another program is required to continue execution. This service can only be called from monitor or real-time programs.

The calling sequence is:

```
TLO        PARLIST
BLU        $SUSP
```

where the parameter list is:

PARLIST	+0, +1	8-character program areaname in truncated ASCII.
	+2, +3	8-character program qualifier in truncated ASCII.
	+4	Physical device number used to distinguish multiple copies of the same program. This is the diagnostic PDN of the desired program. If zero, the highest priority program with the specified name is suspended.

Upon return from this service, the A register is set:

A	=	0	Operation performed as requested.
	<	0	Program not found.
	>	0	Program found, but already suspended.

The corresponding FORTRAN 66 call is:

CALL TOADS ('SUSPEND', name, pdn, istat)

The parameters are:

name	A 17-character string giving qualifier and program name in standard disc area format.
pdn	The diagnostic PDN or zero.
istat	An integer variable set as defined by the A register.

Example

Suspend program 1234ABCD*CAT executing from terminal 42:

```
                TLO     PARLIST
                BLU     $SUSP
PARLIST         DATA    T"CAT␣␣␣␣␣␣"
                DATA    T"1234ABCD"
                DATA    42
```

or CALL TOADS ('SUSPEND', 17H1234ABCD*CAT␣␣␣␣␣␣,42, ISTAT)

5.8 $RSTRT

The Restart service releases a suspended program. The program will continue execution from the point at which it was suspended. This service can only be called from monitor or real-time programs.

The calling sequence is:

```
        TLO     PARLIST
        BLU     $RSTRT
```

where the parameter list is:

PARLIST	+0, +1	8-character program areaname in truncated ASCII.
	+2, +3	8-character program qualifier in truncated ASCII.
PARLIST	+4	Physical device number used to distinguish multiple copies of the same program. This is the diagnostic PDN of the desired program. If zero, the highest priority program with the specified name is released.

Upon return from this service, the A register is set as:

A	=	0	Operation performed as requested
	<	0	Program not found
	>	0	Program found but not suspended

The FORTRAN 66 call is:

```
CALL TOADS ('RESTART', name, pdn, istat)
```

The parameters are:

name	A 17-character string giving qualifier and program name in standard disc area format.
pdn	The diagnostic PDN or zero.
istat	An integer variable set as defined by the A register

5.9 $QSTAT

The Query Status service is used to examine the status of another executing program. The calling sequence is:

```
        TLO     PARLIST
        BLU     $QSTAT
```

where the parameter list is:

PARLIST	+0 +1	8-character program areaname in truncated ASCII.
	+2 +3	8-character program qualifier in truncated ASCII.
	+4	Physical device number used to distinguish multiple copies of the same program. If zero, the highest priority program with the specified name is tested.

Upon return from this service, the A register is used to return the status of the examined program. A may be:

A	=	-1	Program not found
		+1	Program is suspended
		+2	Program is aborting
		+3	Program is loading
		+4	Not used.
		+5	Program is sleeping
		+6	Program is waiting for I/O transfer
		+7	Program is in execution

In the event of more than one of the above conditions being true, the smallest number will be returned.

The FORTRAN call is:

```
CALL TOADS ('STATUS', name, pdn, istat)
```

The parameters are:

name	A 17-character string giving qualifier and program name in standard disc area format.
pdn	The diagnostic PDN or zero.
istat	An integer variable set as defined by the A register.

Example

Examine the status of program 1234ABCD*XYZ.

```
                TLO     PARLIST
                BLU     $QSTAT
                ...
PARLIST         DATA    T"XYZ␣␣␣␣␣"
                DATA    T"1234ABCD"
                DATA    0
```

or CALL TOADS ('STATUS', 17H1234ABCD*CAT␣␣␣␣␣, 0, ISTAT)

5.10 $PABORT

The Program Abort service enables one program to remove another program from the system. This service can only be called from monitor or real-time programs.

The calling sequence is:

```
        TLO     PARLIST
        BLU     $PABORT
```

where the parameter list:

PARLIST	+0, +1	8-character program areaname in truncated ASCII.
	+2, +3	8-character program qualifier in truncated ASCII.
	+4	Physical device number used to distinguish multiple copies of the same program.

Upon return from this service, the A register is used to return the status of the call as:

A	=	0	Operation performed as specified
	<	0	Program not found
	>	0	Program found and already aborting

The C register reflects the condition of the A register.

The corresponding FORTRAN 66 call is:

```
CALL TOADS ('ABORT', name, pdn, istat)
```

The parameters are:

name	A 17-character string giving qualifier and program name in standard disc area format.
pdn	The diagnostic PDN or zero.
istat	An integer variable set as defined by the A register.

5.11 $PRIOR

The change Priority service allows a program to change its own priority or the priority of another program. The calling sequence is:

```
        TLO     PARLIST
        BLU     $PRIOR
```

where the parameter list is defined as:

PARLIST	+0 +1	8-character program areaname in truncated ASCII whose priority is to be changed. If both words are zero, the calling program's priority is changed.
	+2 +3	8-character program qualifier in truncated ASCII.
	+4	Physical device number used to distinguish multiple copies of the same program, or PCA address of the program. If B23 is set, the pdn is assumed to be a control point letter. If zero, the highest priority program of the specified name is changed.
	+5	New priority value.

Upon return from this service, the A register is set to reflect the status of the call:

A	=	-1	Failed to find program
		-2	Cannot change memory-resident real-time to non-resident
		-3	Invalid priority requested
	≥	0	Function performed as requested

The FORTRAN call is:

CALL TOADS ('PRIORITY', name, pdn, new-priority, istat)

The parameters are:

name	"A 17 character string containing the program qualifier and area name which is to be changed. A qualifier and area name of "@@@@@@@@*@@@@@@@@" will refer to the calling program".
pdn	The optional terminal number or zero as discussed above.
new-priority	The new priority value
istat	An integer variable which is set to reflect the status of the call as returned in the A register.

Example

Change the priority of the calling program to 40.

```
                TLO     PARLIST
                BLU     $PRIOR
                ...
PARLIST         DATA    0,0
                BLOK    3
                DATA    40
```

or, assuming calling program is 1234ABCD*CAT,

```
CALL TOADS ('PRIORITY', 17H1234ABCD*CATbbbbbb, 0, 40, ISTAT)
```

5.12 $CONNECT

The Connect service is used to connect a real-time program to an unused external interrupt, and optionally arm and enable that interrupt. When the interrupt occurs, the program will be awakened from the sleep state. The calling sequence is:

```
                TLO     PARLIST
                BLU     $CONNECT
```

where the parameter list is defined:

PARLIST	+0, +1	8-character program areaname in truncated ASCII.
	+2, +3	8-character program qualifier in truncated ASCII.
	+4	External interrupt number. Group 1 is numbered 0-23, and group 2 is numbered 24-47.
	+5	Parameter to be passed to program's K register when awakened.
	+6	-1: temporary, arm and enable interrupt immediately. 0: temporary, do not restore on reload of system, do not arm and enable interrupt. 1: permanent, restore on reboot. 2: permanent, restore on reboot and arm and enable interrupt.

Upon exit from the service, the A register will be set:

A	=	0	Operation performed as requested
		1	Interrupt level does not exist
		2	Interrupt level is used by standard system I/O device
		3	Interrupt level is already connected to a program

The FORTRAN calling sequence is:

CALL TOADS ('CONNECT', name, interrupt, parameter, perm, istat)

The parameters are:

name — A 17 character string containing the program name and qualifier.

interrupt — The interrupt level to be connected (0-47).

parameter — The value to be loaded in the program's K register when awakened.

perm — The flag set to indicate the type of operation: -1: temporary, do not reload on reboot, arm and enable interrupt immediately.

0: temporary, do not arm and enable interrupt.

1: permanent, restore entry on reload of VULCAN, but do not arm and enable interrupt.

2: permanent, and arm and enable interrupt immediately and on each system load.

istat — An integer variable which is set to the contents of the A register following the operation as discussed above.

Example

Connect program 1234ABCD*CAT up to interrupt level 5 on group 2, with parameter -2:

```
            TLO     PARLIST
            BLU     $CONNECT
            ...

PARLIST     DATA    T"CAT␣␣␣␣␣"
            DATA    T"1234ABCD"
            DATA    29
            DATA    -2
            DATA    0
```

or CALL TOADS ('CONNECT', 17H1234ABCD*CAT␣␣␣␣␣, 29, -2, 0, ISTAT)

5.13 $DISCONNECT

The Disconnect service is used to remove a program connected to the specified external interrupt. The interrupt is also disarmed by this call. The calling sequence is:

```
TOK     interrupt
BLU     $DISCONNECT
```

where "interrupt" is the external interrupt number from which the program is to be disconnected.

Upon return from this service, the A register is set:

A	=	0	Operation performed as requested
		1	Invalid interrupt designation
		2	Specified interrupt level does not have a program connected to it

The FORTRAN calling sequence is:

```
CALL TOADS ('DISCON', interrupt, istat)
```

The parameters are:

interrupt	The interrupt level to be disconnected as discussed above.
istat	An integer variable which is set to the contents of the A register as discussed above following the operation.

Example

Disconnect the program from interrupt level 5 on group 2:

```
TOK     29
BLU     $DISCONNECT
```

or CALL TOADS ('DISCON', 29, ISTAT)

5.14 $ENABLE

The Enable service is used to arm and enable a specified interrupt level. A program must be connected to the level. The calling sequence is:

```
TOK     interrupt
BLU     $ENABLE
```

where "interrupt" is the external interrupt number.

Upon return from the service the A register is set:

A	=	0	Operation performed as requested
		1	Interrupt invalid or does not exist
		2	No program is connected to specified interrupt level

The FORTRAN calling sequence is:

```
CALL TOADS ('ENABLE', interrupt, istat)
```

The parameters are:

interrupt	The interrupt level
istat	An integer variable which is set to the contents of the A register upon completion of the operation.

Example

Enable interrupt level 5 on group 2:

```
TOK     29
BLU     $ENABLE
```

or CALL TOADS ('ENABLE', 29, ISTAT)

5.15 $INHIBIT

The Inhibit service is used to disarm and inhibit a specific external interrupt level. A program must be connected to the interrupt level. The calling sequence is:

```
TOK     interrupt
BLU     $INHIBIT
```

where "interrupt" is the external interrupt level designation.

Upon return from this service, the A register is set to reflect the status of the operation as:

A	=	0	Operation performed as requested
		1	Interrupt invalid or does not exist
		2	No program is connected to the specified interrupt level

The FORTRAN calling sequence is:

CALL TOADS ('INHIB', interrupt, istat)

The parameters are:

interrupt	The interrupt level
istat	An integer variable which is set to the contents of the A register given above upon completion of the operation.

Example

Disable level 5 on group 2:

```
TOK     29
BLU     $INHIBIT
```

or CALL TOADS ('INHIB', 29, ISTAT)

CHAPTER 6
TIME/DATE SERVICES

6.1 $DATE

This service returns the current date and time in ASCII format. The calling sequence is:

```
                TLO             PARLIST
                BLU             $DATE
                                  . . .
PARLIST         BLOK            6
```

Upon return PARLIST will contain:

PARLIST	+0	day (dd⌀)
	+1	month (3-letter abbreviation)
	+2	year (⌀yy)
	+3	hours after midnight (⌀hh)
	+4	minutes (:mm)
	+5	seconds (:ss)

For example, if the current date and time were 11:24 PM on July 3, 1974, the service would return:

PARLIST	+0	⌀3⌀
	+1	JUL
	+2	⌀74
	+3	⌀23
	+4	:24
	+5	:00

The $DATE service also allows a special calling sequence which converts a date in $TIME format to that described above. The calling sequence is:

```
                TLO             PARLIST
                NSK
                BLU             $DATE

PARLIST         BLOK            6
```

where Bit 23 of the K register indicates that the date to convert is located in PARLIST+0 and PARLIST+1.

6.2 $TIME

This service returns the current date and time in binary format. It is called by:

```
BLU         $TIME
```

Upon return the A and E registers are set:

A	=	tenths of seconds since midnight
E	=	(bits 23-12) year (bits 11-0) day of year, 1-366
K	=	number of clock ticks since last tenth second

6.3 $EXTIME

The $EXTIME service returns the execution time of the current job or program. This is the CPU time that the program has used up to that point. It is called by:

```
BLU         $EXTIME
```

Upon return the D register contains the current execution time in T register increments. The K register is set to a constant which is the number of T increments per second for the particular CPU.

For example, the $EXTIME service can be used to return a value in seconds.

```
BLU         $EXTIME
DVK
```

$EXTIME can be used to return seconds and milliseconds with the sequence:

```
BLU         $EXTIME
DVK
TAM         seconds
TEA
MYO         1000
DVK
TAM         milliseconds
```

Note that this service will return unspecified values if the program making the call has been Vulcanized as 'non-accounting', or if the system has been generated as not having the timer option.

6.4 $STRTIM

The Start Time service returns the date and time at which this real-time program, terminal, or control point began execution. It is called by:

```
BLU         $STRTIM
```

Upon return, the E register contains the starting date in $TIME format. The A register is set to the starting time in tenths of seconds since midnight.

CHAPTER 7
TEMPORARY STORAGE SERVICES

For user or system subroutines to be reentrant, temporary storage is usually required. Therefore, all programs have cataloged into them a variable size temporary storage area. The default size is 125 words. The system services and control routines utilize this area, which is also available to user routines.

7.1 $PUSH

The $PUSH service allocates temporary storage by executing a "push down" on the storage stack. The calling sequence is:

```
TJK
BLU         $PUSH
DATA        n
```

where n is the number of temporary cells to be allocated.

The K register value to $PUSH is saved as the return address for the subsequent $POP call. Return from $PUSH is to the location following "DATA n" and the J register will be set to the value of the K register on input. The I register points to the first available temporary cell. The E register is saved across the call.

On each $PUSH call, the system increases the number of cells requested by 2 to allow storage of internal pointers. An abort will result if an attempt is made to push more temporary storage than has been allocated.

7.2 $POP

The $POP service releases temporary storage and returns to the address passed in K on the corresponding $PUSH call, and thus is the complement of the matching $PUSH call.

The A, E, and K registers remain intact through the $POP call. The I register returns the current pointer to the temporary area as set after the storage has been released. The calling sequence is:

```
BLU         $POP
```

7.3 $TEMP

The Temp service picks up the current value of the temporary storage pointer as determined by the preceding $PUSH or $POP calls. The calling sequence is:

```
BLU          $TEMP
```

The A, E, and K registers are not destroyed and the I register is returned as the current temporary storage value.

7.4 TEMPORARY STORAGE EXAMPLE

The following example is a reentrant user subroutine using temporary storage to achieve reentrancy. The subroutine saves the calling parameter, which is a floating point number, multiplys it by a constant, and adds in the value in temporary storage, leaving the result in the X register.

```
            XDEF    SUB,ENTRY
            ...
ENTRY       GAP     1
            TIE                 Save argument pointer
            TJK
            BLU     $PUSH
            DATA    2
            TEK                 K = Argument pointer
            TMX*    DAC0/K      Pick up argument
            TXM     0, I        Save in temp.
            MMX     CONS
            AMX     0, I        Add in original argument
            BLU     $POP
            ...
CONS        DATA    2.4D1
DAC0/K      LAC     0, K
            END
```

This routine would be called:

```
            BLL     $SUB
            DAC     argument
```

CHAPTER 8
MEMORY ALLOCATION SERVICES

Blocks of memory may be allocated using the Dynamic Core Manager (DCM). These blocks are allocated from the program's logical address space and thus do not remain when a program exits or chains to another program.

The logical address space from which the blocks are allocated is called dynamic memory. It is above all program code, data, and temporary storage, extending up to the maximum number of pages allocated when the program is Vulcanized.

Dynamic memory is divided into two parts: regular dynamic memory and special common dynamic memory. For "non-X" programs, the regular dynamic memory cannot exist above 64K words of memory. Regular dynamic memory is followed by special common blocks, if referenced in the program, and by special common dynamic memory. Special common blocks and special common dynamic memory may begin below 64K. "X" programs may not have any special common dynamic memory.

There are normally three 1024-word pages of regular dynamic memory Vulcanized with each interactive or control point program. The number of pages of regular dynamic memory may be modified with the Vulcanizer ALLOCATE statement. Special common dynamic memory pages, not normally included, may be Vulcanized with the program by using the Vulcanizer SPALLOCATE statement.

8.1 $DCM

The Dynamic Core Manager is used to allocate and deallocate buffer space within the logical address space of the calling program. The following calling sequence is used to allocate memory:

```
TOE     address
TOK     n
BLU     $DCM
DATA    function-code
```

where "n" is the number of words to allocate, "address" is the address at or above which the block is to be allocated (function 4 and 6 only), and "function-code" indicates the type of memory to be allocated. On return, the registers are set up as follows:

K = non-negative; address of allocated block.

-1; Insufficient space of the type requested exists in logical memory to satisfy the request.

-2; Insufficient Monitor Memory exists for the program to allocate DCM linkage blocks. Some of the program's DCM, JCL registers, or LFN assignments must be returned before the program can request more DCM.

C = condition of K

I = current temp pointer

The function codes are:

1. Allocate regular DCM memory.

2. Allocate regular DCM memory in semi-conductor memory which has been previously resourced by the $RSOURCE service.

3. Allocate regular DCM memory at or above the address specified in E.

4. Allocate special DCM memory. "X" mode program's cannot use this call since they have no special DCM memory. If an "X" program uses this call, it will return a message saying "THERE IS NO SPECIAL DCM".

5. Allocate special DCM memory at or above the address specified in E. "X" mode programs cannot use this call since they have no special DCM memory. If an "X" program uses this call, it will return a message saying "THERE IS NO SPECIAL DCM".

The following calling sequence is used to deallocate all types of DCM memory:

```
TMK     address
BLU     $DCM
DATA    2
```

where "address" contains the address of the memory block to be deallocated. There are no error returns; an invalid address causes the program to abort.

8.2 $LSPACE

The Largest Space available service returns to the user the size of the largest block of memory that could currently be allocated by $DCM. The calling sequence is:

```
BLU         $LSPACE
```

On return, the E register is set to contain the size of the largest memory block available in regular dynamic memory, and the K register is set to contain the size of the largest block of special dynamic memory.

8.3 $MSPACE

The Map Space service returns to the user the size of the largest block of regular dynamic memory, in a particular 32K memory map, at or above a specified address. The calling sequence is:

```
TOK         address
BLU         $MSPACE
```

where address is the address, in a regular dynamic memory map, at or above which the memory block is to be found. On return, the E register contains the size of the largest block and the K register contains the total available regular dynamic memory at or above the specified address in the same map.

8.4 SYSTEM USAGE

The VULCAN system uses program logical address space for each blocked disc area which is open. The number of words used is the number of words in the block plus three. For example, a blocking factor of 2 will cause an allocation of 2 * 112 + 3 = 227 words when opened. With a maximum blocking factor of 7, the system will use 787 words per blocked disc area open. If the disc area is double buffered, twice as much space will be used by the system.

When using the $LSPACE service care should be taken to leave enough space for subsequent blocked area opens. Each active $ADD file requires one additional blocking buffer area.

This example determines the largest available space, allocates it, and later returns the space.

```
BLU         $LSPACE
TEK
BLU         $DCM
DATA        1
CZK
BON         no space
TKM         BLOK
...
TMK         BLOK
BLU         $DCM
DATA        2
```

CHAPTER 9
DISC MANAGEMENT SERVICES

9.1 DISC AREA TYPES

Disc areas are classified according to type. The most basic classification is Program area or Data area. Program areas are always unblocked areas and are generated by the Vulcanizer, or by copying a program area to a non-existant area name. Reentrant library areas are created by the Vulcanizer and are classified as program areas. Areas created through Job Control for the storage of monitor common blocks are treated as program areas also.

Data areas may be generated using Job Control or user programs, and may be blocked, unblocked, or random access.

The type of the area is stored in the Qualifier Disc Directory (QDD) entry for each area. Table 9-1 shows the format of the QDD entry. Tables 9-2, 9-3, and 9-4 give further information about specific area types.

Table 9-1. Format of QDD Entry

Field Name	Word	Usage
M/NAME	0,1	Area name
M/MAI	2	Starting sector number of Master Area Index
M/GRANL	3	B23 Create pending flag B20-B0 Granule size in sectors
M/USER	4	High order 24 bits of user number
M/PACK	5	B23-B12 Low order 12 bits of user number B11-B0 Pack number
M/PROT	6	B18-B12 Access bits B3-B0 Access level
M/TYPE	7	B23 On for program areas B22 On for core directory B21 On for blocked disc areas (always off for programs) B20 On for random disc areas (always off for programs) B19-B0 Additional type information (see Table 9-2)
M/TYP1 M/TYP2 M/TYP3 M/TYP4	8 9 10 11	Additional type dependent information (see Tables 9-3 thru 9-4 for additional information)

Table 9-2. M/TYPE Information in QDD

Program and Data Areas	
B23:	On – Program area Off – Data area
B22:	On – Core Directory Off – Disc Directory
B21:	On if blocked disc area Always off for program areas
B20:	On if random disc area Always off for program areas
Data Areas	
B19:	On — Double buffered blocked area Always off for unblocked areas or single buffered blocked area
B11-B8:	Blocked area type 0 — Blocked standard 1 — Blocked output spool 2 — Blocked input spool 3 — Blocked data queue
B7 — B3	Data area subtype 0 — Normal data area 1 — Checkpoint data area 2 — Checkpoint word area 3 — Checkpoint spool file 4 — Swap area
B2-B0	Type number (0 through 7)
Program Areas	
B19:	On for multiple copy execution
B18:	On for reentrant programs
B17:	On for overlay programs
B16:	On for "X" programs
B15:	On for high core non-resident handler on monitor program
B14:	On for non-paged program (non-resident handler or monitor)
B13-10:	Reserved for future use
B9:	Program uses SAU code
B8:	Program uses 3800 code
B7:	On for reentrant library program
B6:	On for interactive/control-point program
B5:	On for real time program (RTor RRT)
B4:	On for monitor common block program
B3:	On for resident monitor common or real time program

Table 9-2. M/TYPE Information in QDD (CONT'D)

Program Areas		
B2-B0	Program area type	
	000	Non-Resident handler
	001	Monitor program
	010	Reentrant library area
	011	Interactive/control-point program
	100	Real time program
	101	Resident real time program
	110	Monitor common block area
	111	Resident monitor common block area

Table 9-3. M/TYP1, M/TYP2, M/TYP3, and M/TYP4 Data Area Information in QDD

Field Name	Bits	Usage
M/TYP1	B2-B0	Blocking factor (always zero for unblocked areas)
M/TYP2	B15-B8 B7-B0	PDN for spool devices (only for spool areas) Remote site spool device type (only for remote site spool area)
M/TYP3 M/TYP4		Reserved for future use

Table 9-4. M/TYP1, M/TYP2, M/TYP3, and M/TYP4 Program Area Information in QDD

Non-Resident Handler or Monitor Program area		
M/TYP1	Number of words to allocate for temporary stack (Monitor programs only)	
M/TYP2	Size of program in words	
M/TYP3	Number of words in high map relocation vector	
M/TYP4	Reserved for future use.	
Reentrant Library area		
M/TYP1	Number of words in program and data area.	
M/TYP2	B15-B8 B7-B0	Number of reentrant pages. Number of data pages.
M/TYP3 M/TYP4	Reserved for future use.	
Interactive, Control Point, or Real Time Program area		
M/TYP1	Address of TEMP module.	
M/TYP2	B23-B12 B11-B0	Number of reentrant pages. Number of paging registers required for program.
M/TYP3	B16-B12 B6 B5 B4-B0	Number of reentrant library pages. Program is checkpointed Program has subsystem access Number of data library pages
M/TYP4	B23-B12 B11-B0	Number of pages of monitor common. Number of non-reentrant program pages before dynamic memory.

Table 9-4. M/TYP1, M/TYP2, M/TYP3, and M/TYP4 Program Area Information in QDD (CONT'D.)

Monitor Common Area	
M/TYP1	Number of pages in monitor common block.
M/TYP2	Absolute page number for resident monitor common block.
M/TYP3 M/TYP4	Reserved for future use.

9.2 $GENERATE

Disc areas may be created by using the Generate service. The calling sequence for this service is:

```
TLO     PARLIST
BLU     $GENERATE
```

The parameter list is defined as:

PARLIST +0, +1 Areaname. Format is truncated ASCII (8 6-bit characters).

+2, +3 Qualifier. Format is truncated ASCII (8 6-bit characters). If other than blanks or the user's sign-on qualifier is entered, the user must have the appropriate access bit.

First four characters are account, second four are identifier. If both words are zero, then the system qualifier is used. If all 8 characters are blanks, then the user's default qualifier is used unless areaname is a workfile, in which case the user's sign-on qualifier is used.

+4 Granule size in sectors. If zero, default granule size for the specified disc pack is used.

If bit 23 is set and this is an interactive or control-point work area, the work area will be generated with its default granule size and the specified granule size will be ignored.

If bit 22 is set, multiple granules will be allocated. The number of granules to allocate is determined by the current size, in sectors, in PARLIST+18.

If bit 21 is set and this is an interactive or control point work area, the work area will be generated with its default disc area type and the specified disc area type will be ignored. If this is not a work area, bits 23 and 21 are ignored.

PARLIST +5 Maximum size in sectors to which this disc area may expand. If zero, system default maximum area size is used.

+6 Pack number. This pack must have been resourced if it is not a permanently mounted pack. If zero, user pack is used.

+7 Disc area type, as in Table 9–2.

+8 Purge date. Entered as number of days the disc area will be saved. If zero, the system default is used.

+9 Blocking factor. Sectors per block specification for blocked disc areas. Must be in range of 1-7. If zero, default for the disc pack is used.

+10 Spool PDN. Normally used only on system generated spool disc areas.

+11 Access bits:

Bit 6:	1 = public 0 = account
Bit 5 set:	public/account read
Bit 4 set:	public/account write
Bit 3 set:	public/account execute
Bit 2 set:	public/account delete
Bit 1 set:	owner write
Bit 0 set:	owner delete

+12 Access Level. (0-15)

+13 Forced starting sector. This must be input as an absolute sector on the specified pack. If zero, the disc area is generated wherever space exists.

+14, +15, +16, +17 Words M/TYP1 through M/TYP4 used by Vulcanizer for generating program areas, or area name of edit area used by Job Control for generating TP area during edit.

+18 Number of sectors to allocate on generation. Used if bit 22 is set in PARLIST +4. If zero, one granule is allocated.

On return from the $GENERATE call, the status of the operation is returned in the A register as:

A =	0	Function performed as specified
	1	Area name already used on master disc
	2	No space on disc pack
	3	No space on master disc to create qualifier directory
	4	No space on satellite pack to create qualifier directory
	5	Fatal disc I/O error occurred
	6	Area name already used on satellite disc
	7	Pack not resourced
	8	Invalid access level requested
	9	Spool PDN supplied on non-spool generate
	10	Invalid sectors per block specification
	11	Invalid area name syntax.
	12	Old name for TP create does not exist
	13	Old area for TP create does not have write access
	14	Maximum size is less than granule size
	15	Invalid granule size specified
	16	User does not have proper access for this qualifier
	17	User not allowed access to CD type
	18	User not allowed to Vulcanize this program type
	25	Granule size incorrect for area type

For example, with

Name	=	1234ABCD* XYZ
Pack	=	1
Granule Size	=	20 sectors
Maximum size	=	Default
Type	=	Blocked data, 2 sectors per block
Access	=	Owner only

The call would be:

```
            TLO     PARLIST
            BLU     $GENERATE
            BNZ     error
            ...
PARLIST     DATA    T"XYZbbbbbb"
            DATA    T"1234ABCD"
            DATA    20
            DATA    0
            DATA    1               pack = 1
            DATA    '10000000       type = blocked
            DATA    0
            DATA    2               blocking factor
            DATA    0
            DATA    3               access
            DATA    0
            DATA    0
            DATA    0
            DATA    0
            DATA    0
            DATA    0
            DATA    0
```

9.3 $ELIMINATE

Disc areas may be deleted by using the Eliminate service. The calling sequence is:

```
        TLO     PARLIST
        BLU     $ELIMINATE
```

The parameter list is defined as:

PARLIST	+0 +1	Areaname. Truncated ASCII (eight 6-bit characters).
	+2 +3	Qualifier. Truncated ASCII. First 4 characters are account, second four are identifier. If both words are zero, the system qualifier is used. If both are full of blanks, user's sign-on qualifier is used.

Upon return, the A register is set to reflect the status:

A	=	0	Function performed as specified
		5	Fatal disc I/O error
		7	Pack not resourced
		11	Invalid area name syntax
		20	Disc area is open to another program
		21	Disc area does not exist
		22	User does not have delete access

For example, to remove disc area 1234ABCD*XYZ:

```
                TLO     PARLIST
                BLU     $ELIMINATE
                BNZ     error
                ...
PARLIST         DATA    T"XYZ␣␣␣␣␣"
                DATA    T"1234ABCD"
```

9.4 $RNAME

The name of a disc area, along with the qualifier, may be changed with the Rename service. Changing the name of a disc area requires that the user have delete access to the disc area. The calling sequence is:

```
        TLO     PARLIST
        BLU     $RNAME
```

The parameter list is defined as:

PARLIST	+0, +1	Old areaname. Eight characters in truncated ASCII.
	+2, +3	Old qualifier. Eight characters in truncated ASCII. If both words are zero, the system qualifier is used. If all eight are blanks, the sign-on qualifier for the user is used.
	+4, +5	New areaname. Eight characters in truncated ASCII. The first character must be alphabetic.
	+6, +7	New qualifier. Eight characters in truncated ASCII. First four are account, second four are identifier. If both words are zero, the system qualifier is used. If both are blanks, the sign-on qualifier for the user is used.

Upon return from this service, the A register is set to reflect the status of the operation, as:

A	=	0	Function performed as requested.
		1	New area name and qualifier combination already exists on master disc.
		3	No room on master disc to create qualifier directory.
		4	No room on satellite pack to create qualifier directory.
		5	Fatal disc I/O error.
		6	New areaname and qualifier combination already exists on satellite disc.
		7	Disc pack not resourced.
		11	Invalid area name syntax.
		20	Old disc area is in use by another program.
		21	Old disc area does not exist.
		22	User does not have delete access.

For example, to change disc area 1234ABCD*XYZ to 1234FGHI*XYZ:

```
                TLO         PARLIST
                BLU         $RNAME
                BNZ         error
                ...
PARLIST         DATA        T"XYZ ⌀⌀⌀⌀⌀"
                DATA        T"1234ABCD"
                DATA        T"XYZ⌀⌀⌀⌀⌀"
                DATA        T"1234FGHI"
```

9.5 $RTYPE

The disc area type, access, and other parameters may be altered for an existing disc area by using the $RTYPE system service. $RTYPE can only be used by the file's owner.

The calling sequence is:

```
                TLO         PARLIST
                BLU         $RTYPE
```

The parameter list is defined as:

EQ IV	PARLIST		
RT!NAME	+0 +1		Areaname. Eight characters in truncated ASCII.
RT!QUAL	+2 +3		Qualifier. Eight characters in truncated ASCII. If both words are zero, the system qualifier is used. If all blanks, the user's sign-on qualifier is used.
RT!TYPE (B23:RT!CTY!)	+4	B23:	If B23 is set, bits 2-0 contain new type information, otherwise bits 2-0 are ignored.
RT!CCD!		B22:	If B22 is set, bit 20 contains the new core dictionary information, otherwise the core will not be changed.
RT!NTY!		B21:	If B21 is set, PARLIST+9 through PARLIST+12 contains M/TYP1 through M/TYP4 words to retype (for program retypes), otherwise the type words will not be changed.
RT!CD!		B20:	If 0, The DD disc dictionary is requested.
RT!CST!		B19:	If 1, indicates that there is a new Data Area. Subtype in bits 7-3.
		B18-8:	Reserved for future definition.
		B7-3:	Data area subtype.

	B2-0:	For data areas, Bits 2-0 contain the new disc area type. For monitor common blocks, Bits 2-0 contain: 6 — non-resident monitor common blocks 7 — resident monitor common blocks
RT!PROT	+5	If non-zero, Bits 6-0 are new area access bits to be ORed with existing access bits. If Bit 23 if set, then old bits are not used, and Bits 6-0 become new access bits. See $GENERATE for the definition of bits. If PARLIST+5 is zero, the access is left unchanged.
RT!MAX	+6	If non-zero, new area maximum size in sectors. If zero, maximum remains unchanged.
RT!PURG	+7	If non-zero, new purge date in future from current day. If zero, purge date remains unchanged.
RT!ACC	+8	New access level. If zero, old access level remains. To change the access level to zero, set bit 23 only.
RT!TYP1	+9	New M/TYP1 word for program areas (not used for monitor common retypes).
RT!TYP2	+10	New M/TYP2 word for program areas.
RT!TYP3	+11	New M/TYP3 word for program areas (not used for monitor common retypes).
RT!TYP4	+12	New M/TYP4 word for program areas (not used for monitor common retypes).

Upon return from this service, the A register is set to reflect the status of the operation:

A	=	0	Operation performed as requested
		5	Fatal disc I/O error
		7	Pack not resourced
		11	Invalid areaname syntax
		20	Disc area in use by another program
		21	Disc area does not exist
		22	User is not the owner of the disc area.

For example, to change the access of disc area 1234ABCD*XYZ to public read, owner write, and owner delete:

```
                TLO         PARLIST
                BLU         $RTYPE
                BNZ         error
                ...
PARLIST         DATA        T"XYZ ØØØØØ"
                DATA        T"1234ABCD"
                DATA        0
                DATA        B23B6B5B1B0
                DATA        0
                DATA        0
                DATA        0
```

9.6 $SQUEEZE

A disc area may be compressed to a smaller size with the Squeeze service. This routine is used to remove granules from the end of the disc area. Blocked areas are automatically compressed to contain only valid data by this service. The user specifies the resultant size desired for unblocked areas. The Squeeze service requires that the user have delete access to the disc area. The calling sequence is:

```
                TLO         PARLIST
                BLU         $SQUEEZE
```

The parameters are defined as:

PARLIST	+0, +1	Areaname. Eight characters in truncated ASCII format.
	+2, +3	Qualifier. Eight characters in truncated ASCII. If both are zero, the system qualifier is used. If blanks, the user's sign-on qualifier is accessed.
	+4	For unblocked areas, this parameter is required to be the number of sectors to compress to. The disc area will be compressed to the next larger granule boundary above this size. This parameter is not required for blocked areas.

Upon return from this service, the A register is set to reflect the status of the operation:

A	=	0	Operation performed as required
		5	Fatal disc I/O error
		7	Disc pack not resourced
		11	Invalid areaname syntax
		20	Disc area in use by another program
		21	Disc area does not exist
		22	User does not have delete access to disc area.

For example, assume disc area 1234ABCD*XYZ is unblocked and currently has three granules of 20 sectors each. Then, after the following call is made, it will have only two granules of 20 sectors each:

```
            TLO     PARLIST
            BLU     $SQUEEZE
            BNZ     error
            ...
PARLIST     DATA    T"XYZ ØØØØØ"
            DATA    T"1234ABCD"
            DATA    30
```

9.7 $DASAVE

The $DASAVE service obtains disc area information on a single disc area or a group of areas. The calling sequence is:

```
            TLO     PARLIST
            BLU     $DASAVE
            DATA    n
```

where the contents of the parameter list is determined by the specific function invoked, and n is the function code.

Each of these services return the QDD (Qualifier Disc Directory) entry for one or more disc areas. A QDD entry is twelve words long. A description of the QDD entry is provided in Table 9-1.

9.7.1 Single Disc Area Information

This function obtains the area information of the specified disc area. This information will be returned only if the user has valid access to the disc area. The calling sequence is:

```
TLO     PARLIST
BLU     $DASAVE
DATA    6
```

On entry, the parameter list is defined as:

PARLIST	+0 +1	Areaname. Eight characters in truncated ASCII.
	+2 +3	Qualifier. Eight characters in truncated ASCII.

If the disc area is located, the following information is returned.

PARLIST	+4 . . . +15	Disc Area QDD entry. The user-number field will be set to zero.
	+16 +17	Qualifier.
	+18	Current disc area size in sectors.
	+19	Maximum disc area size in sectors.
	+20 +21	Purge date/time.
	+22 +23	Generated date/time.
	+24 +25	Last referenced date/time.
	+26 +27	Last written date/time.

If the disc pack was not resourced, PARLIST+18 will be set to -1, and PARLIST+19 to PARLIST+27 will not be returned. All date/time fields are in $TIME format. The first word has the year in bits 23-12, the day of the year in bits 11-0. The second word is the time in tenths of seconds since midnight. Upon return from this service, the A register is set to reflect the status of the call:

A	=	-2	Disc area not there
		0	Function performed as requested
		3	Disc read error in MDD
		4	Disc read error in QDD
		5	Disc read error in specified disc area

For example, to find out information about disc area 1234ABCD*XYZ:

```
                TLO         PARLIST
                BLU         $DASAVE
                DATA        6
                BNZ         error
                ...
PARLIST         DATA        T"XYZbbbbbb"
                DATA        T"1234ABCD"
                BLOK        24
```

9.7.2 Privileged Disc Area Information

This service is identical to the single disc area service except that it returns the correct user number field. Only disc areas generated by the user may be accessed, unless the requesting program is Job Control, and the user has System Save access. The calling sequence is:

```
TLO         PARLIST
BLU         $DASAVE
DATA        7
```

9.7.3 Multiple Disc Area Information

This service obtains the area names and corresponding parameters of all disc areas that match a specified parameter or group of parameters.

Information for up to 9 disc areas is returned on each call. Subsequent calls may be made to obtain further disc areas that match the previously specified parameters.

The calling sequence is:

```
TLO         PARLIST
BLU         $DASAVE
DATA        n
```

where the matching function is determined by n as listed in the following table.

n	Function
< 0	Invalid.
0	Used for subsequent calls to continue returning information on previously specified parameters.
1	Find all disc areas on requested pack number supplied in PARLIST+0.
2	Find all disc areas with requested qualifier supplied in PARLIST+2 and PARLIST+3.
3	Find all disc areas on requested pack in PARLIST+0 that have requested qualifier in PARLIST+2 and PARLIST+3.
4	Find all disc areas with requested account number supplied in PARLIST+2.
5	Find all disc areas on specified pack in PARLIST+0 having requested account number in PARLIST+2.
6	Single disc area information.
7	Privileged disc area information.
8	Find all disc areas generated by current user.
9	Find all disc areas on specified pack in PARLIST+0 generated by current user.
10	Find all disc areas with specified qualifier in PARLIST+2 and PARLIST+3 generated by current user.
11	Find all disc areas on specified pack in PARLIST+0 with specified qualifier in PARLIST+2 and PARLIST+3 generated by current user.
12	Find all disc areas with specified account number in PARLIST+2 generated by current user.
13	Find all disc areas on specified pack in PARLIST+0 with specified account number in PARLIST+2 generated by current user.
>13	Invalid

Access to disc areas is restricted to those generated by the current user unless the request comes from the Job Control program and the user has System Save access. Thus for regular user program requests, functions 1-5 are treated as if they were 9-13.

Upon return from this service, one or more 24-word Disc Area Information Blocks (DAIB) are returned starting at PARLIST+4, PARLIST+28, PARLIST+52, etc. Because of the disc optimization used by this service, 1-9 blocks are returned on each call. The word count returned (24 times the number of information blocks returned) will be returned in register E on each call. Subsequent calls may be made until the word count in E goes to zero (all disc areas located). Each call after the first should be with a function code of n=0, to continue scanning based on previously entered parameters.

The contents of the Disc Area Information Blocks will be:

Word	Contents
0-11	Disc area QDD entry
12-13	Disc area qualifier
14	Current file size in sectors
15	Maximum file size in sectors
16-17	Purge date/time
18-19	Generated date/time
20-21	Last referenced date/time
22-23	Last written date/time

Each date/time is in $TIME service format.

If the disc pack containing the located disc areas has not been resourced, word 14 is set to -1 and words 15-23 are not returned.

Upon return from this service, in addition to the word count in the E register, the A register is set:

A	=	-1	One or more disc areas are on unresourced packs
		0	Function performed as requested
		1	Invalid function requested
		2	No initial call made though n passed as 0
		3	Disc read error in MDD
		4	Disc read error in QDD
		5	Disc read error in user disc area

For example, the following routine will locate all disc areas generated by the current user located on disc pack 2:

```
START   TLO     BUF
        BLU     $DASAVE
        DATA    9
        CZA
        BNZ     error
LOOP    ...     process block
        TLO     BUF
        BLU     $DASAVE
        DATA    0
        CZA
        BNZ     error
        CZE
        BNZ     LOOP
        end     all disc areas processed
BUF     DATA    2           Pack
        BLOK    3           not used
BLOCK   BLOK    216         9 blocks times 24
                            words per block
```

9.8 $DAASGN

The Disc Area Assign service is used to access a disc area which is to be saved. It is used by Job Control to assign to the disc area being accessed with the $KEEP command. However, user programs may access the service to provide special purpose assignments.

This service will assign LFN 200 to the specified disc area, and will allow the disc area to be accessed in an unblocked mode regardless of the mode of the disc area. The calling sequence is:

```
TLO     PARLIST
BLU     $DAASGN
```

where PARLIST is the label of a Disc Area Information Block output from the $DASAVE service.

The status of the call is reflected in the contents of the A register returned from the service as:

A	=	0	Function performed as requested
		1	User does not have valid access to disc area
		2	Specified disc area does not exist
		3	User specified QDD entry does not match actual entry

9.9 $DAREST

The Disc Area Restore service is used to generate a disc area prior to fetching it. The service is used by Job Control for the $FETCH command. User programs should not need to use it.

The service generates a disc area matching the specified parameters. It then assigns LFN 200 to the disc area and provides unblocked access to the area regardless of the status of the area. The calling sequence is:

```
TLO     PARLIST
BLU     $DAREST
```

where PARLIST contains a Disc Area Information Block in the format output by the $DASAVE service.

The status of the call is returned in the A register as:

A	=	0	Function performed as requested
		1	User does not have valid access to disc area
		2	Specified disc area already exists
		3	Disc area cannot be generated

CHAPTER 10
TAPE MANAGEMENT

10.1 $TAPEOP

The Tape Option service is used to obtain the default system tape option word. This word is set in GENASYS and is used on resource commands by Job Control when no specific tape option information is included. The calling sequence is:

```
BLU        $TAPEOP
```

Upon return, the A register is set to the default tape type:

Bit 0	=	1	9 track
		0	7 track
Bit 1	=	1	High Speed (greater than or equal to 75 IPS)
		0	Low Speed (less than 75 IPS)

The E register is set to the default tape option word:

Bit 16	=	0	No conversion
		1	BCD/EBCDIC conversion
Bit 14	=	0	Odd parity
		1	Even parity

Bits 17, 13-12

Definition:				
17	13	12		
0	0	0	=	200 BPI
0	0	1	=	556 BPI
0	1	0	=	800 BPI
0	1	1	=	1600 BPI
1	0	0	=	6250 BPI

Bits 11-10

Characters per word:

11	10		
0	0	= 1	CPW
0	1	= 2	CPW
1	0	= 3	CPW
1	1	= 4	CPW

All other bits are zero.

For a 9 track drive, 3CPW implies a complete 24-bit memory word transfer. For a 7 track drive, 4 CPW implies a complete memory word transfer. With 9 track each character is 8 bits; with 7 track, 6 bits. A smaller CPW specification than a full 24-bit memory word transfers correspondingly fewer bits. Characters are taken from or transferred to the lower bits when less than a full memory word is transferred.

10.2 TAPE LABELLING

The VULCAN tape labelling system is composed of the non-resident handlers V:TLSS:V, V:TLH1:V, and V:TLH2:V. Since the handlers are loaded only when tape labelling is requested, the impact on installations not using tape labelling is negligible.

For installations using the VULCAN tape labelling system, the chances of operator or user error is minimized and tape data integrity and security are enhanced. Tape labelling also provides for easy transfer of data tapes between a VULCAN system and other computer installations using ANSI standard tape labels.

The tape labelling system has been organized so that tape labelling on individual tape drives may be declared as default, optional, or mandatory. If default, a user must explicitly declare if unlabelled processing is desired. If optional, a user must declare if labelled processing is desired. If mandatory, only users with the correct access mask (set at GENASYS) may process labelled tapes as unlabelled.

10.2.1 Standard Tape Labels

Labelled tapes processed by the VULCAN tape labelling system must conform to ANSI Standard X3.27-1969 and revisions. The standard is described in *Magnetic Tape Labels for Information Interchange* published by the American National Standards Institute. A summary of the standard as it applies to VULCAN tape labelling is provided in the following paragraphs. Definitions of the key terms used are:

volume	A reel of magnetic tape.
file	A collection of user data records which does not include imbedded user-end-of-file-records. When writing data records to a tape, the user may separate "files" with an end-of-file.
file section	A part of a file recorded on any one volume. A large file continued on two or more volumes consists of two or more "file sections".
multi-volume file	A file which is continued on two or more volumes and therefore consists of two or more file sections.
file set	A collection of related files recorded consecutively on a set of volumes.
void file	An empty data file indicated by a double tape mark which precedes subsequent useable data on the tape volume.
label	A record at the beginning or end of a volume, file, or file section. The label identifies and/or provides information about the file. A label record is not considered part of a file.
label group	A collection of one or more contiguous label records at the beginning or end of a volume, file, or file section.

system labels — Label records read, written, and processed by the system with minimal user control. The content of these labels is defined, maintained, and verified by the system. The actual content of some fields of the system labels may be supplied by the user, but the system will supply a default content if none is specified by the user. All required labels are system labels, although all system labels are not required. Only those labels which are required are written by the VULCAN tape labelling system. Optional system labels will be processed, checked for validity, and ignored by the system on reading and will not be written by the system.

user labels — Optional user label records, the contents of which are defined by the user. The length and content of certain fields is defined by the ANSI standard. User labels will be processed, checked for validity, and ignored by the system on reading and will not be written by the system.

tape mark — A special hardware end-of-file record written on the magnetic tape by the system to delimit the boundary between files and/or label groups. There is no direct one-to-one correspondence between user write-end-of-file functions and tape marks on the tape. A user write-end-of-file results in one or more tape marks and label groups being written to the tape.

double tape mark — Two tape marks, one immediately following the other. The termination of useable data on a tape volume is indicated by a double tape mark.

a — Any alphabetic character.

n — Any numeric character.

The following table defines the types of labels, names, identifiers, and numbers of labels as described in the ANSI specification.

Type	Name			Identifier and Number
Beginning of Volume	Operating System	Volume Header Label	Required	VOL1
			Optional	Prohibited
	User	User Volume Header Label	Optional	UVL1 to UVL9
End of Volume	Operating System	End of Volume Label	Requried	EOV1
			Optional	EOV2 to EOV9
	User	User End of Volume Label	Optional	UTLa
Beginning of File	Operating System	File Header Label	Required	HDR1
			Optional	HDR2 to HDR9
	User	User File Header Label	Optional	UHLa

Type	Name			Identifier and Number
End of File	Operating System	End of File Label	Required	EOF1
			Optional	EOF2 to EOF9
	User	User End of File Label	Optional	UTLa

The required system labels and their use are:

Volume-Header Label (VOL1) The first label record on each labelled volume. This label record must appear at the beginning of every volume and may not appear at any other place in the volume.

End-of-Volume Label (EOV1) If a volume ends within a file, the last user data record of the file in that volume must be followed by an end-of-volume label (EOV1). A single tape mark must immediately precede and a double tape mark must immediately follow each end-of-volume label group. File sets must not be terminated by an end-of-volume label group.

File-Header Label (HDR1) Every file must be preceded by a file header label (HDR1). If a volume ends within a file, the continuation of that file in the next volume must also be preceded by a file header label. Every file header label group must be immediately followed by a tape mark. Actual user data records follow this tape mark.

End-of-File Label (EOF1) The last record of user data of every file must be followed immediately by a tape mark. An end-of-file label must immediately follow this tape mark and another tape mark must immediately follow every end-of-file label group. The end-of-file label group appearing at the end of the last (or only) file in a set must be followed by two tape marks rather than one. This double tape mark indicates the end of useable data on the volume.

The following examples illustrate how these labels are used with various tape formats. An asterisk (*) indicates a tape mark.

Single file, single volume
VOL1 HDR1* – file A – *EOF1**

Single file, multi-volume
VOL1 HDR1* – first part of file A – *EOV1**
VOL1 HDR1* – last part of file A – *EOF1**

Multi-file, single volume
VOL1 HDR1* – File A – *EOF1*HDR1* – File B – *EOV1**

Multi-file, multi-volume
VOL1 HDR1* – file A – *EOF1*HDR1* – first part of file B – *EOV1**
VOL1 HDR1* – continuation of file B – *EOV1**
VOL1 HDR1* – last part of file B – *EOF1*HDR1* – file C – *EOF1**

10.2.2 Standard Label Record Contents

VULCAN tape labels are 80 character records written in ASCII. Labels are not padded, although the system can process foreign tapes which have padded labels. The contents of each field of the required labels are defined:

VOL 1 – VOLUME-HEADER LABEL FORMAT

Field	Name	Length in Characters	Description
1	Label Identifier	3	Must be "VOL"
2	Volume Label Number	1	Must be "1"
3	Volume Serial Number	6	Any six "a" characters to identify this physical volume. Corresponds to external reel identification (if any).
4	Accessibility	1	Any "a" character which indicates restrictions on who may have access to the information in the volume. Access determination is discussed in paragraph 10.2.4.
5	Reserved for future standardization	20	Must be "spaces"
6	Reserved for future standardization	6	Must be "spaces"
7	Owner Identification	14	Any 14 characters supplied by user
8	Reserved for future standardization	28	Must be "spaces"
9	Label Standard Level	1	1 means the labels and data formats on this volume conform to the requirements of the ANSI standard.

HDR 1 – FILE-HEADER LABEL FORMAT

Field	Name	Length in Characters	Description
1	Label Identifier	3	Must be "HDR"
2	Label Number	1	Must be "1"
3	File Identifier	17	Any "a" characters
4	Set Identification	6	Any "a" characters
5	File Section Number	4	The file section number of the first header label of each file is "0001". This applies to the first or only file on a volume and to subsequent files on a multi-file volume. This field is incremented by one on each subsequent volume of the file.
6	File Sequence Number	4	Four "n" characters denoting the sequence (that is, 0001, 0002, etc.) of files within the volume or set of volumes. In all the labels for a given file, this field will contain the same number.
7	Generation Number	4	Four "spaces". Not checked on input.
8	Generation Version Number	2	"Spaces" on output. Not checked on input.
9	Creation Date	6	A "space" followed by two "n" characters for the year, followed by three "n" characters for the day ("001" to "366") within the year.
10	Expiration Date	6	Same format as field 9. The file is regarded as "expired" when today's date is equal to, or later than, the date given in this field. When this condition is satisfied, the remainder of the volume may be overwritten. To be effective on multi-file volumes therefore, the expiration date of a file must be less than, or equal to, any previous files on the volume.
11	Accessibility	1	Any "a" character which indicates any restrictions on who may have access to the information on this file. Access determination is discussed in paragraph 10.2.4.

HDR 1 – FILE-HEADER LABEL FORMAT (CONT'D.)

Field	Name	Length in Characters	Description
12	Block Count	6	Must be six zeros.
13	System Identifier	13	VULCAN operating system release level.
14	Reserved for future standardization	7	Must be "spaces".

EOF 1 END-OF-FILE LABEL FORMAT

Field	Name	Length in Characters	Description
1	Label Indentifier	3	Must be "EOF".
2	Label Number	1	Must be "1".
3-11	Same as corresponding fields in the File-Header label	50 (Total)	Identical to the corresponding fields in the File-Header (HDR 1).
12	Block Count	6	Six "n" characters denoting the number of data blocks (exclusive of labels and tape marks) since the preceding HDR label group.
13,14	Same as corresponding fields in File-Header label	20 (Total)	Identical to the corresponding fields in the File-Header Label (HDR1).

EOV 1 END-OF-VOLUME LABEL FORMAT

Field	Name	Length in Characters	Description
1	Label Identifier	3	Must be "EOV".
2	Label Number	1	Must be "1".
3-14	Same as corresponding fields in File-Header label	76 (Total)	Identical to the corresponding fields in the File-Header label.

10.2.3 Skeleton Tapes

The tape labelling system will only write data files to labelled tapes. Consequently any new tapes or tapes that are to be "scratched" must be skeletonized by a user with the proper access mask. A skeleton tape contains a VOL1 label, a HDR1 label and three hardware tape marks. The formats of the skeleton labels are:

VOL1 – SKELETON LABEL

Field	Name	Length in Characters	Description
1	Label Identifier	3	"VOL"
2	Label Number	1	"1"
3	Volume Serial Number	6	6 "a" character to identify this physical volume. Corresponds to external reel identification (if any).
4	Accessibility	1	See paragraph 10.2.4.
5-6	Reserved for future standardization	26	All "spaces"
7	Owner Identification	14	The user name specified by the person writing the skeleton label.
8	Reserved for future standardization	28	"spaces"
9	Label Standard Level	1	"1"

HDR1 – SKELETON LABEL

Field	Name	Length in Characters	Description
1	Label Identifier	3	"HDR"
2	Label Number	1	"1"
3	File Identifier	17	All "0"s
4	Set Identification	6	"spaces"
5	File Section Number	4	"0001"
6	File Sequence Number	4	"0001"

HDR1 – SKELETON LABEL (CONT'D.)

Field	Name	Length in Characters	Description
7	Generation Number	4	"spaces"
8	Generation Version Number	2	"spaces"
9	Creation Date	6	Year and day
10	Expiration Date	6	Year and day
11	Accessibility	1	See paragraph 10.2.4
12	Block Count	6	All "0"s
13	System Identifier	13	VULCAN operating system level
14	Reserved for further standardization	7	"spaces"

10.2.4 Access Determination

Since a primary objective of the labelling system is file security, access determination is an integral part and is organized to allow maximum flexibility. In order to write to a tape, a user must have write access both to the volume and to any files he intends to rewrite; in order to read a file, the user must have both volume and file read access.

Volume access characters are specified when a tape is skeletonized and are not altered thereafter. Note that volume access in no way alters file access restrictions. Access characters supported and the access levels allowed are:

Volume Access Character	Access	
	Owner	Non-owner
Space	Read,Write	Read,Write
R	Read,Write	Read
Other	Read,Write	None

File access characters are written in the HDR1 label whenever the first record of a file is written. Thus, the character is set when a new file is created or when an entire file is rewritten; it is not altered when only a portion of an existing file is rewritten. All file access is, of course, subject to volume access restrictions. File access characters and the access levels are:

File Access Character	Access	
	Owner	Non-owner
Space	Read,Write	Read,Write
R	Read,Write	Read,Write*
0	Read,Write	Write*
Other	Read,Write*	Write*

WRITE* indicates that write access is allowed only if the file has expired or if the user is writing a new file. Expired means that the current date is equal to or greater than the expiration date in the file header label.

Sample Access Structures

a. Volume access character "R", file access character "R"; the volume owner can read or write to the file, but a non-owner can only read the file.

b. Volume access character "space", file access character "*"; the owner can read the file and can write to it only after it has expired; a non-owner cannot read the file and can write to it only after it has expired.

c. Volume access character "*", file access character "space"; the volume owner can read or write the file, but a non-owner has no access.

Note that the structure of magnetic tape is such that all information beyond the point of a write must be regarded as being destroyed. Thus, the elimination dates for a file set must be non-increasing in order to ensure the file access restrictions have the intended effect.

10.2.5 Tape Label System Service

In order to access tapes with the tape labelling system, the system service $TLABEL must first be invoked to initialize the system. Although the service may be called directly from any program, initialization will normally be effected through use of the Job Control $TLABEL command. In initializing the system, the calling program submits both a list of volume identifiers and file names. Taken together, these are regarded as constituting a file set with no presumed correlation between specific volumes and specific files. Thus, the order of the volume identifiers and file names is significant. In submitting the list of file names, the calling program is in effect declaring its universe of reference to the tapes. Once initialized, the system will operate as though all of the volumes constituted a single tape containing only files specified by the calling program. If writing to the tape, files will be written with file names taken sequentially from the list; if reading, the system will make the tape appear to contain only the files specified in the file name list. The files will appear to be separated from each other by a single tape mark.

If during a forward motion of the tape the list of file names becomes exhausted, the last name in the list is repeated as often as needed. A count of the number of times this name has been repeated is maintained so that the correct correspondence of files and file names can be maintained during a backwards motion of the tape. Thus, arbitrarily many files can be written to the tape without the need to specify a corresponding number of file names.

10.3 $TLABEL

The Tape Label service is used to initialize (or reinitialize) the tape labelling system. This section details the calling sequence for the system services. The calling sequence for the service is:

```
TLO     PARLIST
BLU     $TLABEL
```

where PARLIST is defined as:

PARLIST+0 — Bits 13-6 contain the LFN to be used in label processing; Bits 5-0 contain a function code to specify what processing is to be performed.

PARLIST+1 — The number of volumes, or reels of magnetic tape, in the file set. For each volume, there is a seven word block describing the volume:

WORD0, WORD1 — Six ASCII characters to specify the volume name (Volume Serial Number, or VSN).

WORD2 . . WORD6 — Fourteen ASCII characters for the volume owner's user name. This is used only when creating skeletons. The fifteenth character is the volume access character.

If *v* is the content of PARLIST+1 (the number of volumes in the file set), then the volume identification block for the *i*th volume ($1 < i < v$) will be located at PARLIST+2+7*i-1,...,PARLIST+8+7*i-1.

After the volume identification blocks are the file identification blocks, together with a count of the number of files in the files set:

PARLIST+2+7*v The number of files in the file set. For each file in the set, there is a nine word information block describing the file:

WORD0 – WORD5: Seventeen ASCII characters for the file name followed by the file access character.

WORD6 – WORD7: Six ASCII characters for the file set to which this file belongs.

WORD8: The expiration date of the file, given as the year in bits 23-12 and the day of the year in bits 11-0.

If v, as above, is the number of volumes in the file set, then the file identification block for the jth file is at: PARLIST+3+7*v+9*j-1).

The LFN specified at PARLIST+0 is presumed to be assigned to a tape drive which has been resourced. If this is not the case, the calling program will be aborted.

Function codes (fc) processed by the service are:

1. Query. Each volume specified is scanned to interpret the volume structure. The results are sent to LFN 6, which will be dynamically assigned to work file LO if not presently assigned. The results consist of an exact copy of each label record found on the tape, a double asterisk ("**") to indicate each hardware tape mark detected, and an indication of each data region on the tape. When a function code of 1 is used, the number of file identification blocks specified should be zero.

2. No label processing. This function directs the tape labelling system not to process any I/O transfers on the LFN specified. Upon successful execution of this command, the tape drive will operate as though the labelling system did not exist. This function code is used to switch from labelled processing to unlabelled processing. In order to preserve the security provided by the system, this command may not be executed on drives having labelled processing mandatory unless the user has the correct access mask. For this function code and function code 5, a short call may be used:

```
TNK     100*lfn + fc
BLU     $TLABEL
```

3. Normal initialization for labelled read/write functions.

4. Create skeleton tapes. In order to execute this function, the user must have the correct access mask to skeletonize. For this function, the user names in the volume identification blocks are used in creating volume header labels. The system sequences through each volume creating skeleton labels and then deallocates the drive upon completion. Note that the number of file identification blocks is expected to be zero for a function code of 4.

5. Unlabelled processing. The subsystem examines the current volume to determine if it is a labelled tape. If so, the current process is aborted. If unlabelled, the subsystem allows the tape to be processed as an unlabelled tape. Note that the short-form call described for function code 2 can be used for a function code 5.

File names in the file identification blocks may be any names the user chooses and need have no correlation with any external names. If it is desired to write a new file on the volume, bit 23 of the first word in the file identification block should be set. If it is desired to read or rewrite a file known to exist, bit 23 should not be set.

The procedure followed while searching for a file is to search for an actual name match if this bit is not set, or to position the tape to the first position encountered during a forward scan at which a file could be written if the bit is set.

CHAPTER 11
MESSAGE COMMUNICATION SERVICES

Message Communication services are used to pass information between any program, user, terminal, or combination thereof. Messages can be either symbolic or binary, up to a maximum of 1024 words. Messages are maintained on system disc areas and are thus saved across system reboots. On each VULCAN reboot, the areas are squeezed automatically to remove areas no longer used. Each message has associated with it a purge date, after which the message is automatically removed if not already received by a program.

11.1 $SEND

The Send service is used to transmit a message to a program, terminal, and/or user. The calling sequence is:

```
        TLO     PARLIST
        BLU     $SEND
```

where the parameter list is defined as:

PARLIST	+0, +1	Areaname of program to which the message is destined. The format is 8 characters in truncated ASCII. All zeroes should be entered if no specific program is required.
	+2, +3	Qualifier of program. The format is 8 characters in truncated ASCII. If no specific program, zeroes should be used.
	+4, +5, +6, +7	Username of user to which the message is destined. This should be 12 ASCII characters left justified with blank fill, and must appear exactly as the user name stored for the user when added to the system. All zeroes should be entered if no specific user is required.
	+8	The PDN of the terminal to which the message is to be sent. If zero, any terminal may receive the message.
	+9	Purge Flag. Bit 23 should be set to indicate a permanent message, which is not to be removed until the purge date is reached. If Bit 23 is off, the message is removed when received. Bits 22-0 of this word specify the number of days to keep the message before automatic removal. If zero, the default of 7 days is used.

+10	Word Count. This is the word count of the message to be sent. If Bit 23 is set, a binary message is assumed. If not set, symbolic I/O is used.
+11	Buffer Address. This word contains the address of the message to be sent.

Upon return from this service, the A register is set as follows:

A	=	-1	Message areas in use by another program. The message could not be sent.
		0	Message sent as requested.

Note that if any of the three fields (programname, username, terminal) are zero, no match is required on them in order to receive the message. Thus, if all three fields are entered as zeroes, any program, terminal, or user may receive the message.

Note also that as a matter of good programming, should the return code in the A register indicate "Message areas in use by another program", simply using a branch back to attempt to resend (such as via "BNZ *-2") may cause the program to loop without ever allowing the message files to be freed. A call to $DELAY is recommended before attempting to resend the message.

Example

Send a message to user "SMITH":

```
            TLO     PARLIST
            BLU     $SEND
            ...
PARLIST     DATA    0,0,0,0
            DATA    "SMITHbbbbbbb"
            DATA    0
            DATA    0
            DATA    3
            DAC     BUF
BUF         DATA    "HI SMITH"
```

11.2 $RECEIVE

The Receive service is used to receive a message destined for the calling program, user, and/or terminal. The calling sequence is:

```
            TLO     PARLIST
            BLU     $RECEIVE
            ...
PARLIST     DATA    Word count of buffer
            DAC     Buffer address to receive message
```

Upon return from this service, the A register is set:

A	=	-1	Message areas in use by another program and no messages could be received.
		0	There are no messages currently registered for this program, user, and/or terminal.
	>	0	Word count of message transferred.

If the word count of the message is greater than the length of the buffer, additional characters are truncated at the right of the message string. If the word count of the message is less than the buffer's length and if the message was sent in symbolic mode, blanks will be appended to the right of the message string to fill the buffer; no padding occurs if the message was sent in binary mode.

Note that the first word of the buffer contains the purge date of this message in $TIME format. Thus, to receive a message which is known to be 27 words long, a word count of 28 with a 28-word buffer is required.

11.3 $LOOK

The Look service is used to determine if there are any outstanding messages for this program, user, and/or terminal. The calling sequence is:

```
BLU     $LOOK
```

Upon return from this service, the A register is set:

A	=	-1	Message areas in use and no query could be performed.
		0	No messages found.
		1	At least one message found.

11.4 $CMESAG

The Count Message service is used to determine the number of messages outstanding to the calling program, user, and/or terminal. The calling sequence is:

```
BLU     $CMESAG
```

Upon return from this service, the A register is set:

A	=	-1	Message areas in use and no count could be made.
		0	No messages found.
	>	0	Number of messages found.

It should be noted that if merely a determination as to whether any messages are outstanding is required, the $LOOK service is potentially much faster as an entire search of the message area may not be required.

CHAPTER 12

INTERRUPT SERVICES

12.1 Introduction

The Special Interrupt services are designed to provide the user with program interrupt capability. These interrupts, called "SPINTs," can be generated from interactive terminals resourced by the interrupted program or from other programs running concurrently on the same system. A SPINT causes the interrupted program to execute a service routine defined by the user for that SPINT. The methods for sending, receiving, and controlling these interrupts are discussed in the remainder of this chapter.

12.2 $SPINT: Special Interrupt Enabling/Disabling Service

A SPINT is defined as being within a particular group and at a specific level in that group. The $SPINT service allows the program to enable or disable itself for SPINTs on any group/level.

The calling sequence for this service is:

```
        TLO     SPLIST
        BLU     $SPINT
```

Where the parameter list referenced by "SPLIST" contains:

```
SPLIST  DATA    group/level-specification
        DAC     special-interrupt-data-block-address
        DAC     interrupt-routine-address
```

The group/level specification word in the parameter list will contain the group and level to be enabled or disabled in the following format:

Bit 23:	1 - Disable levels specified
	0 - Enable levels specified
Bits 22-18:	Group Specification
	(1 or 2; binary value)
	Refer to subsection 12.2.2 for restrictions on the use of group 1.
Bits 17-00:	Level Specifications
	(0 through 17; bit map)
	Multiple levels may be specified

The use of the interrupt data block and interrupt service routine are discussed in subsection 12.2.1.

Upon return from the $SPINT service, the contents of the registers will be:

E : Group 0 SPINT request bits
A : Group n SPINT request bits
(n is the group specified in the group/level specification word)
I : Current TEMP pointer
J : Saved

K : Undefined
V : Undefined
H : Undefined
C : Set to reflect condition of A register

Multiple calls can be made to the $SPINT service. Each enable call can specify a different interrupt data block and service routine for the enabled SPINTs, or additional SPINTs can be enabled for a previously specified interrupt data block and service routine. The service routine and data block for a SPINT can be redefined by calling the service with an enable request for the group and level of the SPINT, specifying the new routine and data block. No prior disable is required. Previously enabled levels that are not specified are not affected.

Note: A call to $SPINT should not be made from within an interrupt service routine as this will force an exit from the routine.

12.2.1 Special Interrupt Handling

When a process or device "SPINTs" a program, the SPINT is queued with prior unserviced SPINTs for that program. When a program is selected for execution by the operating system, a check is made for pending SPINTs. The first SPINT on the queue will be selected for service if any SPINT exists. Spint priorities are therefore chronologically determined.

The system first determines the interrupt data block and service routine addresses specified for the group/level of the SPINT. The basic SPINT information (group/level of the SPINT and information word) is then passed to the user program in the space designated by the interrupt data block address. The program's registers are saved in the data block and control is given to the interrupt service routine.

The special interrupt data block must contain nine words for the basic SPINT information and the register save area. The information is stored in this block as follows:

Word 0: Group/Level Interrupting

Bit 23:	1 - Additional information block available 0 - No additional information
Bits 22-18:	Group Specification (0 through 31; binary value)
Bits 17-00:	Level Specification (0 through 17; bit map)

Word 1: Information Word

This word will contain information in a format defined by the interrupting program or device. If the interrupt is from a device, the word will be formatted as:

Bits 23-16: LFN assigned to device at time of BLU $I/O, function code '25
Bits 15-08: SPINT subtype
Bits 07-00: SPINT type

Word 2: P Register (Program Counter)
Word 3: C Register (Condition Codes)
Word 4: I Register
Word 5: J Register

Word 6: K Register
Word 7: E Register
Word 8: A Register

Upon entry to the user program SPINT service routine, the A register will be set to indicate the size of the additional information block (if it exists). This information can be retrieved via a call to $SPINFO.

Exit from the service routine is made by a call to the $IRETRN service.

12.2.2 Restrictions on SPINT Usage

Some Group 1 SPINT levels have special meanings in certain situations. These levels are documented in the system equivalence file "*V.EQIV". For these group 1 levels to be used by the program, it must be known that these special situations are not occurring. Group 2 is reserved for user applications, and it is recommended that this group be used.

12.3 $TRIGER: Program-Generated SPINT

A call to the $TRIGER service interrupts a program which has been enabled for interrupts via $SPINT. The calling sequence to $TRIGER is:

```
        TLO     PLIST
        BLU     $TRIGER

PLIST   DATA    "idxxxx"
        DATA    information-word
        DATA    group-level-specification
        DATA    word-count of additional information block (0 if none)
        DAC     buffer-address of additional information block
```

The identifier "idxxxx" must have been previously defined as the identifier of the program to be interrupted (see DEFID). The information word and contents of the additional information block must have been previously defined for use between the two programs. The group-level-specification is in the standard group/level format (Bit 23 is a zero).

Upon return from the service, the A Register will be set to reflect the status of the SPINT as follows:

-2: Invalid additional information buffer
-1: Specified identifier not defined
0: TRIGER performed as specified
1: Invalid group/level specification
2: Invalid PCA associated with identifier
3: Desired group-level not defined/enabled by program

To call $TRIGER, either the user or the program must have subsystem access.

12.4 $SPINFO: Additional Information Retrieval

Often it is necessary for a SPINT to convey more information than simply group/level and a single information word. Up to 64 words of additional information may be passed to the interrupted program by means of an additional information block. This block is copied into a buffer within the interrupted program by calling the service $SPINFO and specifying the parameters of the buffer to which the additional information block is to be copied. For example:

```
        TLO     PLIST
        BLU     $SPINFO

PLIST   DATA    word-count of user buffer
        DAC     logical-address of user buffer
```

If the buffer specified by the user is shorter than the actual information block passed by the SPINT, only that portion requested by the user will be copied to his buffer. If, however, the buffer specified is longer than the actual information block passed by the SPINT, the entire actual information block will be copied to the user's buffer with the excess user's buffer remaining unchanged. The status of the additional information block copy will be returned in the A register (with the condition codes set) as follows:

0: Information block returned in specified buffer
-1: Specified buffer not provided in logical space of user program
-2: No additional information block associated with SPINT
-3: User program is not in SPINT interrupt routine

12.5 $DEFID: Define Identifier for $TRIGER

For calls to $TRIGER to be performed, all programs expecting to receive a SPINT from another program must define the identifier under which they expect to receive the SPINT (as defined by a $TRIGER). This is done by a call to $DEFID as follows:

```
        TLO     PLIST
        BLU     $DEFID

PLIST   DATA    "idxxxx"
```

Where "idxxxx" is a string of six ASCII characters which will serve as the program-id. Upon return from this call, the A register (and the corresponding condition codes) will be set to reflect the outcome of the call as follows:

0 : Identifier accepted as specified
-1 : Program already has a specified identifier
-2 : Some other program has already specified the same identifier

After this call has been accepted, all $TRIGER calls referencing this identifier will attempt to SPINT the program making this call.

To call $DEFID, either the user or the program must have subsystem access.

12.6 $INITSS: Initiate Sub-system Program

The $INITSS service is used to initiate a sub-system program. Its use is similar to the $INIT service in that it is used to initiate a real-time program. However, there are several SPINT-related features provided by $INITSS, which allow it to be used to initiate and control a sub-system.

First, a SPINT identifier which is specified in the $INITSS parameter list, is automatically defined for the initiated program. This ensures that the initiated program will have an identifier which cannot be changed and which is known by the initiating program.

Secondly, the calling program will be notified via a SPINT, when the initiated process exits or aborts. This is called a contingency interrupt spint, and will be received by the calling

program on the group/level specified in the $INITSS parameter list. (This group/level must have been previously defined and enabled using the $SPINT service.) When the contingency spint is received, information about the exiting program will be placed in a user-defined information block known as a contingency interrupt block or CIB. This block must be at least 4 words in length and will contain the following:

word 0 & 1	2-word SPINT identifier of the exiting process
word 2	Exit/abort code of the exiting process
word 3	Exit/abort address of the exiting process

It is recommended that the SPINT group/level used for contingency interrupts NOT be used for any other purpose. This is not required, but may help avoid confusion as to whether a received SPINT is a contingency SPINT or not.

The calling sequence for $INITSS is as follows:

```
        TLO     PARLIST
        BLU     $INITSS

        . . .
```

PARLIST	+0	Reserved for future use
	+1	2-word truncated-ascii areaname
	+2	of program to be initiated
	+3	2-word truncated-ascii qualifier
	+4	of program to be initiated
	+5	Priority
	+6	Initiation parameter
	+7	2-word SPINT id to be defined for
	+8	the program to be initiated
	+9	SPINT selection word
	+10	Address of CIB

Upon return from this service, the A-register will be set as follows:

A = 0	Program initiated successfully.
1	Invalid priority specified.
5	No sub-system access.
6	Invalid CIB address.
7	Group 0 is invalid for contingency interrupts.
8	Contingency interrupt group is too large.
9	Multiple SPINT levels specified.
10	Invalid id specified for process being initiated.

12.7 $HPINT: Hold Program Interrupts

To delay reception of program interrupts, a program may call $HPINT. This disabled state will be re-enabled by a call to $RPINT, $IWAIT, or $IDELAY. The call to $HPINT is of the form:

```
BLU         $HPINT
```

12.8 $RPINT: Release Program Interrupts

If a program has held program interrupts through a call to $HPINT, interrupts may be re-enabled by calling $RPINT. The call to $RPINT is of the form:

```
BLU         $RPINT
```

12.9 $IWAIT: Wait for Program Interrupts

If the program desires to wait until a program interrupt occurs, a call to $IWAIT will place the program in a wait state until the receipt of a program interrupt. When a program interrupt occurs, the program will be vectored to the SPINT interrupt routine (as specified in the call to $SPINT) as is normally done. When the program exits the program interrupt, however, the program will continue after the call to $IWAIT. When the program continues the returned registers will contain:

E : Saved
A : Always 0
I : Undefined
J : Saved
K : Undefined
V : Saved
H : Saved
C : Undefined

The call to $IWAIT is of the form:

```
BLU         $IWAIT
```

12.10 $IDELAY: Delay for Program Interrupts

If the program is to wait a specified period of time for a program interrupt, a call to $IDELAY will place it in a delay state for the number of clock ticks specified in the K register. If a program interrupt occurs within the specified period of time, the program will be vectored to the SPINT interrupt routine. On returning it will continue after the call to $IDELAY (any clock ticks remaining at the time of the interrupt will be ignored). If, however, the period of time specified for the delay passes without a program interrupt, the program will resume after the call to $IDELAY. In either case, when the program continues, the return register will contain:

E : Saved
A : 0 Delay terminated by interrupt
-1 Delay terminated by time out
I : Undefined
J : Saved
K : Number of ticks remaining in delay if delay terminated by interrupt
V : Saved
H : Saved
C : Undefined

An example calling sequence is:

```
TOK         n
BLU         $IDELAY
```

where n is the number of clock ticks the program is to wait for an interrupt.

12.11 $IRETRN: Return from Program Interrupts

When a SPINT has occurred and the program has been vectored to the SPINT interrupt service, the program may only exit the service and return to the interrupted processing by a call to $IRETRN. After this call has been made, any basic or additional information associated with the SPINT will no longer be available to the program. A call to $IRETRN is of the form:

```
BLU         $IRETRN
```

12.12 Device-Generated SPINTs

The striking of certain keys on the keyboard of an interactive device can generate SPINTs for the program that has resourced that device. The SPINT generated by an interactive device can be in group 1 or a higher group (group n device SPINT), or it can be a special system SPINT on group 0, level 0 (group 0 device SPINT). In all cases, the program to be interrupted must be enabled for SPINTs via a call to $SPINT.

12.12.1 Group N Device Generated SPINTs

For an interactive device resourced by a program to generate group n SPINTs, it is necessary for the program to alert the device handler as to what SPINT parameters are expected to be used for the interrupt. This information is passed to the respective handler through calls to $I/O with appropriate function codes to enable or disable the type of SPINT desired.

12.12.1.1 Function Code '25: Set SPINT Linkage

If the program expects to receive a SPINT from the device to which the I/O is being performed, it must first set up the SPINT linkage through a call to $I/O using function code '25. The calling sequence is:

```
        TLO     PLIST
        BLU     $I/O

PLIST   DATA    'LFN25
        DATA    SPINT-type
        DATA    group/level-specification
```

The group/level-specification is in the same format as for the call to $SPINT except that bit 23 is always a zero. Only one level may be specified by this function; if more than one is specified, an error condition will be returned. The SPINT-type is defined for each device in the appropriate section of the VULCAN I/O Services Reference Manual, Pub. No. 0860004.

No further calls by the user are required during the execution of the program unless the group/level linkage must be changed or the SPINT is to be reset (function code '26). Upon receipt of the SPINT, the information word of the interrupt data block will contain the number defining the SPINT-type.

12.12.1.2. Function Code '26: Reset SPINT Linkage

If the program expects to receive no more special interrupts from the device to which the I/O is being performed, it may reset the SPINT linkage through a call to $I/O using function code '26. The calling sequence is:

```
        TLO     PLIST
        BLU     $I/O

PLIST   DATA    'LFN26
        DATA    SPINT-type
        BLOK    1
```

No group/level-specification is necessary for this command and is ignored if provided. The reset is performed only according to SPINT-type as defined for each device in the appropriate section of the VULCAN I/O Services Reference Manual.

12.12.1.3 Function Code '14: Close

If the close function is for the same LFN specified on any previous set SPINT function(s) (function code '25), a reset SPINT function will automatically be executed for the associated SPINT(s).

The calling sequence for a close function is:

```
TNK     'LFN14
BLU     $I/O
```

12.12.1.4 Function Code '24: Dump Buffer

If the dump buffer function is for the same LFN specified on any previous set SPINT function(s) (function code '25), a reset SPINT function will automatically be executed for the associated SPINT(s).

The calling sequence for a close function is:

```
TNK     'LFN24
BLU     $I/O
```

12.12.2 Group 0 Device-Generated SPINT

The special group 0, level 0 device-generated SPINT utilizes special system SPINT services. This SPINT will be serviced prior to all other SPINTs pending for the program. This SPINT is enabled by a call to $SPINT with a group/level specification of 1. It is disabled via a disable call to $SPINT. The keyboard sequences required for generation of this SPINT are documented in the section of the VULCAN I/O Services Reference Manual that deals with the specific device.

Note: A call to $SPINT to enable the group 0, level 0 SPINT will redefine the service routine for all levels in group 0. These levels are used by the system in certain situations, and an interrupt service for group 0 may have been previously defined by the system. The user is cautioned to ensure that no system use of group 0 is occuring prior to enabling the group 0, level 0 SPINT.

CHAPTER 13

INTER-PROCESS COMMUNICATION SERVICE

13.1 $PLINK Services

The following is a presentation of a system service that will permit general inter-process communication on VULCAN. The service allows concurrent programs to communicate arbitrarily large blocks of data back and forth. This communication is done via a "link" initiated by one of the communicating programs. A description of the functions provided by the $PLINK service follows. Note that the $SPINT, $SPINFO, and $DEFID services must also be employed by the user of the $PLINK service.

13.2 Request Link

The request link function allows a process to request a link with another process. The request is made via the other process's program-id. This id is established via a call to $DEFID. A SPINT group/level is specified for the link. Processes that wish to communicate over the link must first enable themselves on this group/level via the $SPINT service.

The calling sequence for the request link service is:

```
        TLO     PLIST
        BLU     $PLINK

PLIST   DATA    1
        DATA    48-bit program-id of the program to
        DATA    ... which the link is being offered
        DATA    interrupt selection word
        DATA    link capability bytes
        DATA    -1 (to be used in future development)
        DATA    -1 (to be used in future development)
```

Word 0 of the parameter list is always a 1, indicating a link request. Words 1 and 2 are the 48-bit program-id that identifies the program to connect with over the link. Word 3 is the interrupt selection word to be used for all SPINTs related to the program link. Its format is the same as the interrupt selection word used in a BLU $SPINT.

Word 4 of the parameter list contains the link capability bytes. These bytes define the read and write capability that will exist over the link. Bits 15-8 define the capability that is being offered to the process the caller is attempting to connect to. Bits 7-0 define the capability that the process is defining for itself. The capability bytes are defined as follows:

```
B0      - send message capability
B1      - receive message capability
B2      - to be used in future development
B3      - drop link capability
B4-B6   - to be used in future development
B7      - alter capability access
```

Setting any of the above bits gives the appropriate access to the referenced process. Bit 0 set indicates that the referenced process will be capable of using the send message

service. Bit 1 set indicates that the referenced process will be capable of using the receive message service. These two are equivalent to write and read access, respectively. Bit 3 set indicates that the referenced process will be capable of using the drop link service. Bit 7 set indicates that the referenced process will be capable of altering both processes' capability bytes using the alter link capability service.

If both bytes are 0, then both processes will have full capabilities. This is equivalent to setting bits 0, 1, 3, and 7 for both processes' capability bytes.

Words 5 and 6 should both be -1.

The possible returns from this service are:

A >0 A positive integer in register A indicates that the link has been offered to the process specified by the program-id in the parameter list. The integer in register A is the link-id for the communication link. The process specified by the program-id in the calling parameter list is SPINTed on the specified group/level with the following word passed as information:

B23-20 = 1
B19-00 = link-id being offered

A = -1 Program-id in the calling parameter list is not defined for any process in execution on the system.

A = -2 Process specified by the program-id in the calling parameter list does not have the specified SPINT group/level enabled. No link may be established with a process that does not have the specified group/level enabled.

A = -4 Process specified by the program-id in the calling parameter list has a link request pending to the caller. No link is offered. The calling process must first accept or reject the link offered by the other process before a second link may be formed.

A = -5 Invalid parameter list pointer.

A = -6 Invalid capability bytes. Neither process has alter capability access and both processes have send message capability only.

A = -7 Invalid capability bytes. Neither process has alter capability access and both processes have receive message capability only.

A = -8 Invalid capability bytes. Neither process has alter capability access and neither process has drop link capability.

A = -10 Invalid group/level specified. The specified group/level is not a valid SPINT level for program link communications.

13.3 Link Information

The link information service allows a process to request information about a particular link via the link-id. The calling sequence is:

```
        TLO     PLIST
        BLU     $PLINK

PLIST   DATA    2
        DATA    link-id
        DATA    word count of info block
        DAC     ... address of info block
```

Word 0 of the parameter list is always 2, indicating a link information request. Word 1 of the parameter list is the link-id to obtain information on. Word 2 is the number of words of information desired (the currently defined information block is 13 words long). Word 3 is the address of a block into which information about the link will be placed. The format of the link information block is as follows:

Words 0-1 = Program name of the process that the calling process is connected to.

Words 2-3 = Program qualifier of the process that the calling process is connected to.

Words 4-5 = Program-id of the process that the calling process is connected to.

Word 6 = Status of the link as follows:

B7-0 : 0 = link established
1 = link offered

Word 7 = Current link capability bytes.

Word 8 = Number of messages pending reception by caller.

Word 9 = Message-id of the next message available for reception.

Word 10 = Length of the next message available for reception.

Word 11 = Message-id of the last message received by the calling process via this link.

Word 12 = Message-id of the last message received by the connected process.

The possible returns from this service are:

A = 0 The link information is placed in the block provided.

A = -1 The link-id is not defined for the calling program.

A = -3 The information block is not contained in the caller's logical address space.

A = -5 Invalid parameter list pointer.

13.4 Accept Link

The accept link service allows a process to accept a link request from another process. The calling sequence is:

```
                TLO     PLIST
                BLU     $PLINK

PLIST           DATA    3
                DATA    link-id to accept
                DATA    -1 (to be used in future development)
                DATA    -1 (to be used in future development)
```

Word 0 of the parameter list is always a 3, indicating an accept call. Word 1 of the parameter list is the link-id to accept. Words 2 and 3 should both be -1.

The possible returns from this service are:

A = 0 The communication link between the two processes is established. The process which offered the link is SPINTed on the specified group/level for the program link, indicating that the link has been established. The SPINT information word passed has the following format:

B23-20 = 3
B19-00 = link-id that was accepted.

A = -1 The link-id specified is not offered to the calling process.

A = -2 The offering process has exited. The link is no longer available.

A = -5 Invalid parameter list pointer.

13.5 Reject Link

The reject link service allows a process to reject a communication link with the process that has requested one. The calling sequence is:

```
                TLO     PLIST
                BLU     $PLINK

PLIST           DATA    4
                DATA    link-id to reject
```

Word 0 of the parameter list is always a 4, indicating a reject call. Word 1 of the parameter list is the link-id to reject communications on.

The possible returns from this service are:

A = 0 The process that requested the link is SPINTed on the specified group/level for the program link indicating that the link has been rejected. The format of the information word is:

B23-20 = 4
B19-00 = link-id that was rejected.

A = -1 The link-id specified is not offered to the calling process.

A = -2 The offering process has exited.

A = -5 Invalid parameter list pointer.

13.6 Alter Link

The alter link capability service allows a process to alter the link capability bytes of both the connected and calling processes. This service is available to the calling process only if it has alter capability access. The calling sequence is:

```
        TLO     PLIST
        BLU     $PLINK

PLIST   DATA    12
        DATA    calling process's new capability
        DATA    connected process's new capability
```

Word 0 of the parameter list is always a 12, indicating an alter link capability call. Word 1 (bits 7-0) is the new capability byte for the calling process. Word 2 (bits 7-0) is the new capability byte for the connected process. If the current capability byte of one of the processes is to remain unchanged, then the appropriate word should be set to -1.

The possible returns from this service are:

A = 0 The capability bytes are altered as requested. A SPINT is generated to the connected process, indicating a change in capability. The format of the information word passed on the SPINT is:

B23-20 = 12
B19-00 = link-id on which capability altered.

The connected process may do a link information call to determine the new capabilities.

A = -6 Invalid capability bytes. Neither process has alter capability access and both processes have send regular message capability only.

A = -7 Invalid capability bytes. Neither process has alter capability access and both processes have receive regular message capability only.

A = -8 Invalid capability bytes. Neither process has alter capability access and neither process has drop link capability.

13.7 Send Message

The send message service allows a process to send a regular message to the process connected via a particular link-id. The calling sequence is:

```
        TLO     PLIST
        BLU     $PLINK
```

```
PLIST    DATA    8
         DATA    link-id to send on
         DATA    send buffer
         DAC     ... descriptor
```

Word 0 of the parameter list is always an 8, indicating that this is a send message request. Word 1 is the link-id of the communication link to send the message on. The send buffer descriptor describes the message to be sent. The buffer described must not be modified until the "message received" SPINT (described under receive message service) is received.

A buffer descriptor is a sequence of data chains. Data chains are defined in pairs of words which must consecutively follow each other in the parameter list. The first word of the data chain pair is the byte count or word count:

BITS 23-22 = starting byte specification:
- 00 = bits 21-0 are a word count
- 01 = leftmost byte
- 10 = middle byte
- 11 = rightmost byte

BITS 21-00 = byte count or word count. This is determined by bits 23-22.

The second word defines the starting word of the data chain and whether or not another data chain follows:

BITS 23-22 = restart bits:
- 00 = no restart, end of data chain
- 01 = no restart, end of data chain
- 10 = another data chain follows
- 11 = invalid

BITS 21-00 = address of first word of data chain

The data chain list is interpreted as follows. Bits 23 and 22 of the first word are examined. If they are both zero, then bits 21-0 define the length of the data chain in words. Bits 21-0 of the second word point to the starting word for the data chain. If bits 23 and 22 of the first word are not both zero, then they are logically or'ed with bits 21-0 of the second word to form a byte pointer. This byte pointer points to the first byte of the data chain. Bits 21-0 of the first word define the length of the buffer in bytes.

The possible returns from this service are:

A > 0 The message specified has been queued to the connected process for reception. The positive integer in A represents the message-id assigned to the message by the operating system. The connected process will receive a SPINT on the specified group/level for the program link, indicating that a message is pending on the link. The format of the information word will be:

B23-20 = 8
B19-00 = link-id that message is pending on

The process may determine the message-id of the next message on the link via the link information service.

A = -1 The link-id specified is not established for the calling process.

A = -3 The message buffer to be sent is not contained in the calling program's logical address space.

A = -4 No send message capability for calling process.

A = -5 Invalid parameter list pointer.

A = -6 Invalid capability bytes. Neither process has alter capability access and both processes have send regular message capability only.

A = -7 Invalid capability bytes. Neither process has alter capability access and both processes have receive regular message capability only.

A = -8 Invalid capability bytes. Neither process has alter capability access and neither process has drop link capability.

A = -9 Invalid capability bytes. Send-receive hotline capability is not available to the caller but a hotline buffer was defined.

A = -10 Invalid group/level specified. The specified group/level is not a valid SPINT level for program link communications.

13.8 Receive Message

This service allows a process to receive a message from a connected process. The calling sequence is:

```
        TLO     PLIST
        BLU     $PLINK

PLIST   DATA    9
        DATA    link-id to receive on
        DATA    receive buffer
        DAC     ... descriptor
```

Word 0 of the parameter list is always a 9, indicating that this is a receive call. Word 1 is the link-id to receive the message on. The receive buffer descriptor describes a buffer into which the message header and the message will be placed.

The possible returns from this service are:

A = 0 The message is received into the specified buffer. The sending process is SPINTed on the specified group/level for the program link, indicating that the message is received. The format of the information word is:

B23-20= 9
B19-00= link-id that connected process received a message on.

The sending process may determine which message was received by issuing a $SPINFO call. One additional word of information--the message-id that was received by the connected process--is passed on the SPINT. This allows the sender to free up the buffer associated with the message-id.

A = -1 The link-id specified by the calling program is not established.

A = -2 No message is pending on the communication link.

A = -3 The buffer provided to receive the message is not contained in the caller's logical address space.

A = -4 No receive regular message capability for calling process.

A = -5 Invalid parameter list pointer.

Each message is preceded by an n-word header defined as follows:

Word 0 = The value of n. This is the number of words (including this one) that are contained in the header message. The value is currently 4. This will allow expansion of the header at a future time if necessary. All programs should take this into consideration when processing messages.

Word 1 = Message-id of the message received.

Word 2 = Number of messages left to receive.

Word 3 = Status and word count transferred:

B22 : 0 = Byte count complete.
1 = Byte count not complete. The buffer provided was not large enough to contain the message header and message.
B15-0 = Byte count transferred.

13.9 Flush Message

This service allows a process to remove all of the pending messages from its message queue. The messages pending will be discarded and a "message received" SPINT will be generated to the connected process for each message that is flushed. The calling sequence is:

```
        TLO     PLIST
        BLU     $PLINK

PLIST   DATA    13
        DATA    -1 (to be used in future development)
```

Word 0 of the parameter list is always a 13, indicating a flush message call. Word 1 must be -1.

13.10 Drop Link

The drop link service allows a process that has established a communication link with another process to terminate the communication link. The calling sequence is:

```
        TLO     PLIST
        BLU     $PLINK

PLIST   DATA    5
        DATA    link-id to terminate
```

The possible returns from this service are:

A = 0 The communication link specified by the link-id in the calling parameter list is terminated. The process connected via this link is SPINTed on the specified group/level for the program link, indicating that the communication link is dropped. The format of the information word passed is:

B23-20 = 5
B19-00 = link-id dropped

Two additional words of information are available through a $SPINFO call. The first word is the number of messages sent by the calling process to the connected process that were pending reception when the link was dropped. The second word is the number of messages sent by the connected process to the calling process that were pending reception when the link was dropped. All messages pending on the link in both directions are discarded.

A = -1 The link-id specified is not defined for the calling process.

A = -2 The link-id exists but was never accepted.

A = -4 No drop link capability for calling process. The link is still available.

A = -5 Invalid parameter list pointer.

13.11 $PLINK - SPINT Summary

The following is a summary of all SPINTs which are defined for the $PLINK service and their meaning. They all pass an information word of the form:

B23-20 = N
B19-00 = link-id

The value of N may be:

N = 1 LINK REQUESTED. A process has issued a link request. The identity of the process attempting to connect may be determined by a link information call.

N = 2 PROCESS HAS EXITED. The process to which a link offer is outstanding has exited. No link may be established.

N = 3 LINK ACCEPTED. The process to which a link was offered has accepted the link request. The communication link is established.

N = 4 LINK REJECTED. The process to which a link was offered has rejected the link request.

N = 5 LINK DROPPED. The process connected by the link-id has dropped the communication link. Messages pending in both directions are lost.

N = 8 MESSAGE AVAILABLE. The process connected by the link-id has sent a message. Information about the next message on the link may be obtained by a call to the link information service.

N = 9 MESSAGE RECEIVED. The process connected by the link-id has received a message. One additional word of information is available via a $SPINFO call. This word is the message-id of the message received by the connected process. The sender must not modify the message being sent between the time that the send message call is issued and the time that this SPINT is received.

N = 12 LINK CAPABILITY ALTERED. The process connected by the link-id has altered the link capability bytes. A link information call may be used to obtain the new capabilities.

N = 15 LINK EXITED. The process connected by the link-id has exited without explicitly dropping this link. One additional word of information is available via the $SPINFO service. This word is a zero if the process exited normally. Otherwise, it is the abort code indicating the cause of the abort. Conditions under aborts are described below.

13.12 Exceptional Conditions

The following are some exceptional conditions which may occur and the action to be taken:

1. Process A has offered a link to process B. Process A aborts. In this case, process B will get an error return when it attempts to accept or reject the communication link.

2. Process A has offered a link to process B. Process B aborts. In this case, process A is SPINTed, indicating that process B has aborted. The link offer is no longer outstanding.

3. Process A has established a communication link with process B. Process A aborts. In this case, process B receives a SPINT indicating that process A has aborted and receives the abort code via a $SPINFO call. Messages pending on the link are lost.

CHAPTER 14
MISCELLANEOUS SERVICES

14.1 $PTYPE

The Program Type service allows the calling program to determine if it is a control point, interactive, or real-time program. The calling sequence is:

```
BLU     $PTYPE
```

Upon return, the condition code and E register are set as follows:

E	<	0	Real-time program
	=	0	Interactive program
	>	0	Control-point program

14.1A $SYSLEV

The System Level service allows the calling program to determine the current system level. The calling sequence is:

```
BLU     $SYSLEV
```

Upon return, the E register is set as follows:

E = system level in 3 digit ASCII (e.g., "10A")

14.2 $SPOOL

The SPOOL service places a blocked disc area on the spool-out queue for a particular device. The calling sequence is:

```
TLO     PARLIST
BLU     $SPOOL
```

where the contents of the parameter list are:

PARLIST	+0, +1	8-character disc areaname in truncated ASCII
	+2, +3	8-character qualifier in truncated ASCII
	+4	Bits 7-0: PDN of device to spool to. Bits 23-8: Number of copies to spool (default is one copy).

Upon return from this service, the A register is set as follows:

A	=	0	Operation performed as requested
		1	Requested physical device does not exist
		2	Spool area is not a blocked disc area

3	Cannot ASSIGN to spool area
4	User does not have read access to spool area.
7	Spool area being written to by another program.

Example

Print two copies of disc area 1234ABCD*XYZ on printer 6:

```
            TLO     PARLIST
            BLU     $SPOOL
            ...
PARLIST     DATA    T"XYZbbbbb"
            DATA    T"1234ABCD"
            FORM    16,8
            DATA    /2,6/
```

14.3 $IJOB

The Insert Job service places a disc area containing a control-point job on the jobs-to-be-run queue. The calling sequence is:

```
            TLO     PARLIST
            BLU     $IJOB
```

where the contents of the parameter list are:

PARLIST	+0, +1	8-character disc areaname in truncated ASCII
	+2, +3	8-character qualifier in truncated ASCII
	+4	Zero if the job is to be run as soon as possible; non-zero if a 20 second delay is to be placed on the job entry to delay execution. This is used by Job Control to hold the execution of jobs requiring resources which are not yet available.
	+5, +6	Used only if IJOBPASSWORD GENASYS parameter used in system generation, for password of user on $JOB card.

If the system was generated without the IJOBPASSWORD parameter, PARLIST+5 and PARLIST+6 are not used and ignored if specified. If the IJOBPASSWORD parameter was used, PARLIST+5 and PARLIST+6 may contain:

Binary zero, denoting the user named on the $JOB card has no password or is the same user as the one calling $IJOB.

ASCII password of user named on $JOB card, left justified.

If the user named on the $JOB card has no password, PARLIST+5 and PARLIST+6 must be zero. Any password specified must accurately match the password of the user on the $JOB card.

Upon return from this service the A register is set:

A=	0	Operation performed as requested.
	1	First record of area not $JOB card.
	2	Disc area is not blocked.
	3	Cannot assign to area.
	4	User does not have read access to area.
	5	Invalid $JOB card format.
	6	Invalid password on $IJOB call.
	7	Area being written to by another program.

14.4 $USERNO

The User Number service is used to access and validate user numbers. Job Control uses it to handle sign-on, and any user program may use it to return the name of the current user.

The $USERNO service has two calling formats.

Format 1:

```
        TLO     PARLIST
        BLU     $USERNO
```

Format 2:

```
        TLO     PARLIST
        NSK
        BLU     $USERNO
```

The PARLIST for Format 1 is:

Base Location	Displacement	Description
PARLIST	+0	Unused
	+1	Unused
	+2	Unused
	+3	Unused
	+4	The 12-character user name is returned in
	+5	PARLIST+4 through PARLIST+7. The user
	+6	name is returned as ASCII characters, and,
	+7	if necessary, the buffer is blank-filled.

Upon return the A register is set:

A	=	0	Service completed.
		1	User number invalid.
		2	User number valid, qualifier invalid.
		3	Call type invalid or do not have access.
		4	User does not have access to terminal.
		5	User has no CPU time left.
		6	Password mismatch.

Normally the A register value is used only by Job Control.

Example

Return the current user name to BUF:

```
        TLO     BUF-4
        BLU     $USERNO
        ...
BUF     BLOK    4
```

The second format is available only to Job Control, OPCOM, and all high access programs. PARLIST is a buffer in which the following information is supplied or returned.

The PARLIST for Format 2 is:

Base Location	Displacement	Description
PARLIST	+0 +1	PARLIST+0 and PARLIST+1 contain a user number in ASCII, left-justified and blank-filled.
	+2 +3	PARLIST+2 through PARLIST+3 contain the qualifier in ASCII, left-justified and blank-filled.
	+4 +5 +6 +7	The 12-character user name is returned in PARLIST+4 through PARLIST+7. The user name is returned as ASCII characters, and, if necessary, the buffer is blank filled.
	+8 +9	PARLIST+8 and PARLIST+9 contain the password for use by Job Control or high access user programs.

14.5 $OPCOM

The OPCOM service is used to issue an OPCOM command. The valid commands accepted from user programs, and their format, are discussed in the VULCAN Operator Communications (OPCOM) manual.

Returns from the command will consist of A = 0 for valid commands, and A set to an OPCOM error number for OPCOM errors. Commands producing output information will write to LFN 3.

The calling sequence is:

```
        TLO     PARLIST
        BLU     $OPCOM
```

where PARLIST is a 24-word buffer containing the requested command in ASCII, 3-characters per word.

Example

Query the status of terminal 42 and return the output to LFN 3:

```
        TLO     BUF
        BLU     $OPCOM
        ...
BUF     DATA    "QP⍽42⍽"
        RDAT    22 ("⍽⍽⍽")
```

14.6 $PACK

The Pack service returns the system spool pack and work pack numbers. The calling sequence is:

```
        BLU     $PACK
```

The system spool pack number is returned in the E register and the work pack number in the K register.

14.7 $BINASC

The binary to ASCII conversion service is used to convert a binary integer into ASCII characters suitable for output. This routine will convert any 24-bit signed integer. Output will have leading zeroes replaced with blanks, and the negative sign, if present, will be positioned adjacent to the leftmost non-zero digit. The calling sequence is:

```
TMA     number
BLU     $BINASC
```

where "number" is the binary number to convert. Upon return, the ASCII characters are in the K and D registers where the K register content should be concatenated to the left of the D register content. The V register is saved.

14.8 $OCTASC

The Octal to ASCII conversion service converts an integer value into ASCII characters suitable for output. The calling sequence is:

```
TMA     number
BLU     $OCTASC
```

where "number" is the number to convert. The ASCII octal result is returned in the K, E, and A registers:

```
K       "space" followed by characters one and two
E       characters three, four, and five
A       characters six, seven, and eight
```

14.9 $C/RTN

The Contingency Return service provides a means whereby a program may regain control after an SAU trap or system abort. Following an operator (either OPCOM or terminal user) abort, any abort (including a second operator abort) is unconditionally fatal. After the user program is given control of an abort, the next abort is fatal unless the user again makes a call $C/RTN. The user program is allowed control of an unlimited number of non-aborting SAU traps. The calling sequence for this service is:

```
        TLO     USRRTN      K = address of user routine to
        BLU     $C/RTN      which control is to be given
        DATA    parameter
        ...
USRRTN  DATA 0
        DATA 0
```

The user may specify in "parameter" whether the system is to return control to the user only in the case of an abort, or in both cases of aborts and non-aborting SAU traps. The user may additionally specify whether system abort messages are to be output by the system. Valid parameter values and their meaning are:

Parameter	Definition
0	Remove previous C/RTN call entry.
1	Do not inhibit abort messages, branch to USRRTN+2 only in case of an abort.
2	Do not inhibit abort messages, branch to USRRTN+2 for both aborts and non-aborting SAU traps.
3	Inhibit system abort messages, branch to USRRTN+2 only in case of an abort.
4	Inhibit system abort messages, branch to USRRTN+2 for both aborts and non-aborting SAU traps.

In case of an abort, the system will branch to USRRTN+2 with USRRTN+0 containing the system abort code and USRRTN+1 containing the abort location in bits 15-0. In case of a non-aborting SAU trap, the system will branch to USRRTN+2 with USRRTN+0 containing the SAU error code with bit 23 set. USRRTN+1 will contain the trap location in bits 15-0 and the C Register in bits 19-16. The user may return to the non-aborting SAU trap location by restoring all registers to the values contained upon entry to USRRTN and executing a BRL* USRRTN+1 instruction.

$C/RTN is valid and useful for monitor programs as well as for real-time and interactive programs.

14.10 $MUNLD

The Monitor Common Unload service is used to force the updating of Monitor Common disc areas. Normally "MCOM" disc areas are only updated when the last program using them exits or a demand page forces them to be swapped out. The calling sequence for this service is:

```
TLO     address
BLU     $MUNLD
```

where "address" is any address in the Monitor Common Block that is to be updated. If it is desired to update all "MCOM" blocks belonging to the program, the following call may be made:

```
TNK     1
BLU     $MUNLD
```

14.11 $RESORC

The Resource service is used for resource allocation. It allocates four types of resources which are:

1. Disc Packs
2. Physical Devices
3. Magnetic Tapes
4. Floppy Disc Drives

The calling sequence is:

```
TLO     PARLIST
BLU     $RESORC
DATA    n
```

where "n" is the function:

n = 1 Allocate resource. This code is used to pass an initial resource request list. This is the only function needed for non-interactive programs. Interactive programs must follow this with function code 3.

2 Test allocation. Used only by interactive programs to test the status of the specified resource allocation; also used by real-time programs to test allocations made after using function code 5.

3 Wait for allocation. Used only by interactive programs to wait for completion of specified allocations. (Wait is implicit in calls made by real-time programs after using Function Code 1, and control-point programs never wait.) This code is also used by real-time programs to wait for allocations made after using function code 5.

5 Allocate resource. n=5 in real-time programs operates identically to n=1 in interactive programs.

The format of PARLIST is a series of resource entries, terminated by a zero word. The size and content of the resource entry is dependent on the device type, as follows:

Disc Pack	One word entry (followed by a zero word): bits 7-0 set to pack number; other bits are zero.
Physical Device	One word entry (followed by a zero word): bits 7-0 set to PDN; bits 15-8 set to LFN which is to be assigned to the device in bits 15-8, and bits 23-16 set to a 2 (bit 17 on).
Magnetic Tape Drive	Five word entry (followed by a zero word): Word 0: bits 15-8 set to LFN to be assigned to tape drive; bits 23-16 set to a 1 (i.e., bit 16 on). Words 1-2: six-character tape name in ASCII; if both zero, scratch is assumed. Word 3: tape option word (see $TAPEOP; bit 23 set if write access required (write ring to be inserted).

Word 4: has either the PDN's of tape drives to resource or a tape specification. Bit 23 of word 4 must be set if the resource is to a PDN. If a specific drive is to be resourced, its PDN is set in bits 7-0. If an alternate drive may be used, its PDN is set in bits 15-8. If any drive meeting tape specifications may be used, the tape type is set in bits 1-0 as:

Bit 1: 1 = high speed
0 = low speed
Bit 0: 1 = 9 track
0 = 7 track

Floppy Disc Drive

Five word entry (followed by a zero word): Word 0: bits 15-8 set to LFN to be assigned to the drive; bits 23-16 set to 1 (i.e., bit 16 on)

Words 1-2: six-character diskette name in ASCII; if both zero, scratch is assumed.

Word 3: bit 22 set; bit 23 set if write access is required.

Word 4: set to zero if any drive may be used. If a specific drive is required, bit 23 is set and the PDN of the required drive is set in bits 7-0. If an alternate drive may be used, it is specified in bits 15-8

Upon return from the call, the A register is set:

A =	0	Resources allocated as specified.
	1	For control points: requested resources not available. For interactive terminals: one or more resources not yet available. Not returned for real-time programs that have used function code 1; is returned for real-time programs that use function code 5.
	2	Non-existent physical device requested.
	3	Non-allocatable physical device requested.
	4	Disc I/O error on *V:PACK area.
	5	Disc pack does not exist.
	6	Resource request for this device already entered.
	7	No functional disc drives available for specified pack.

8	No functional tape drives of requested type are available.
9	Non-existent high speed memory requested.
10	More high speed memory requested than is available.
11	No functional floppy disc drives available.
12	LFN 0 or 3 requested from a control point or terminal
13	No access to requested tape drive(s).
14	No access to requested floppy disc drive(s).
15	No access to requested disc pack(s).
16	No access to requested physical device(s).
17	Physical device is marked down.

Interactive requests are queued while program execution continues. To access a requested device, a "wait" request (n=3) must be made at the same time following the initial (n=1) call. Real-time programs need make only an initial (n=1) call, and they will be suspended until the requests can be satisfied. Real-time programs can now allocate resources like interactive programs do. The function code n=5 operates the same as n=1 operates for interactive programs. n=2 and n=3 are also available to real-time programs. Control point programs will never wait; if one or more resources cannot be immediately allocated, all resources will be returned, and status returned to the calling program.

Non-interactive program allocations for disc drives and tape drives will return after the drive is allocated, but before the tape or disc is mounted. If the device is accessed before the medium is mounted, the program will wait in the I/O handler. Interactive resourcing (n=3) will wait until the medium is mounted.

Note: a real-time program that has used n=5 to allocate resources should not use n=1 until all resource requests are satisfied or cancelled (with a close).

Example

Assign LFN 8 to a 9-track low speed drive to read tape "ABCD" which is 800BPI, 3 characters per word ASCII, and wait for the tape:

```
            TLO     PARLIST
            BLU     $RESORC
            DATA    1
            BOZ     *+3
            COB     1
            BNZ     error
            TLO     PARLIST
            BLU     $RESORC
            DATA    3
            TNK     '1013       open 8
            BLU     $I/O
            ...
PARLIST     DATA    B16B11      LFN 8 (bit 11) and bit 16
            DATA    "ABCDØØ"
            DATA    B13B11      tape option
            DATA    1           low speed, 9 track
            DATA    0           list termination
```

14.12 $PASSW

The Password service is used to change the user's password. It is available to any program or user, though its primary use is by the Job Control command CP. The calling sequence is:

```
            TLO     PARLIST
            BLU     $PASSW
            ...
PARLIST     DATA    new-password
            DATA    old-password
```

Each password is two words in ASCII format (six 8-bit characters). A new-password of zero implies that the user is no longer using passwords. The old-password field should be set to zero if the user has no previous password. If old-password does not match the previous password, then new-password is not accepted and the password remains unchanged. Upon return the A register is set:

A	=	0	Password changed
		-1	Old password mismatch

14.13 $TPREAD

The $TPREAD service reads a word from the temporary storage area Vulcanized into a program. The temporary storage area contains information about the internal structure of the program and the program area. For example, it contains the number of reentrant and non-reentrant pages in each memory map, and the starting sector number and size of VBUG tables stored in the program area.

The calling sequence of the service is:

```
        TLO     PARLIST
        BLU     $TPREAD
```

where PARLIST is defined as:

PARLIST	+0	Displacement in temp area of the word to read.
	+1	LFN to be assigned to the program area.
	+2, +3	Eight character program areaname in truncated ASCII. If this field is all zeros, the calling program's name and qualifier are used.
	+4, +5	Eight character program area qualifier in truncated ASCII.

On return the A register is set:

A=	<0	Error condition; error code in E register
	0	Word read as requested; word is in K register
	>0	Unsuccessful assignment; A register contains $ASSIGN service error code.

If an error condition occurred and the A register is negative, the E register is set:

E=	1	Displacement in PARLIST+0 not in temp area.
	2	Area name from PARLIST not a program area.
	3	Area is non-resident handler or monitor program (no temp area).
	4	User does not have area read access.
	5	QDD entry contains errors.
	6	Area cannot be opened for read.
	7	Disc I/O error during area read.

If successfully assigned, the supplied LFN remains assigned to the program area.

Example

To get the size in sectors and the starting sector number of the VBUG tables in the current program, two calls are made to $TPREAD:

```
            TLO     PARLIST
            BLU     $TPREAD
            BNZ     error
            TKM     SIZE        VBUG table size
            TOA     57
            TAM     PARLIST
            TLO     PARLIST
            BLU     $TPREAD
            BNZ     error
            TKM     STRTSEC     VBUG tables starting sector
            ...
SIZE        DATA    -1
STRTSEC     DATA    -1
PARLIST     DATA    56
            DATA    100         LFN to assign to program area
            DATA    0,0         Use calling program's area name
```

14.14 $MTYPE

The Machine type service allows the calling program to determine the type of machine on which the program is running. $MTYPE also indicates the addressing mode for the operating system as well as the calling program.

The calling sequence is:

```
            BLU         $MTYPE
```

The values returned in the A and E registers indicate the machine type and addressing modes. See Table 14-1 for the A register and E register values and their meanings.

Table 14-1 $MTYPE A and E Register Values and Their Meanings

Machine	A Register	OS Addressing Mode	E Register	User Addressing Mode
H80-1, H100-1	-4	X16		X16
H80-1, H100-1	-3	X18	0	X16
			1	X18
Model 4 (S100)	-2	X16		X16
Model 7 (S200)	-1	X16		X16
H80, H100	0	X16		X16
H300, H500	1	X16		X16
H300, H500	2	X20	0	X16
			1	X20
H800	3	X16		X16
H800	4	X20	0	X16
			1	X20

Note: The X16 addressing mode is also known as "non-x" or "compatibility mode".

The X20 addressing mode is also known as "X mode".

The C register reflects the A register's condition.

14.15 $DEFLTS

The Defaults service is used to change or determine certain defaults associated with a user. Any defaults that are changed apply only to the current interactive session, control-point job, or real time/monitor program. This service is available to any program or user, though its primary use is by the Job Control $STATUS and $MODE commands. The calling sequence is:

```
            TLO     PARLIST
            BLU     $DEFLTS
            ...
PARLIST     DATA    function-code
            DATA    data-word
            DATA    data-word
```

Function codes are defined as follows:

Bits 23-9: Reserved for future definition.

Bit 8: Query the default or change the default bit.

- 0 Change default
- 1 Query default

Bits 7-0: Indicates which default to change/determine.

- 0 Qualifier.
  PARLIST+1,2: default qualifier (6 bit truncated ASCII).
  Binary zero implies 0000SYST.
  TASCII blanks imply the sign-on qualifier.
- 1-2 Reserved for future definition.
- 3 Program.
  PARLIST+1,2: default program (6 bit truncated ASCII.
  Binary zero implies *JOBCNTRL. The qualifier 0000SYST is always implied. The next exit or abort from the calling program will invoke the program named. In addition, exits and aborts from any future program will invoke the named program. The quantity of such returns to the named program is limited to 255 unless the program is a high access default program (i.e., *JOBCNTRL).
- 4 Reserved for future definition.
- 5 Checkpoint Data Area Name.
  PARLIST+1,2: Checkpoint Data Area Name.
  PARLIST+3,4: Checkpoint Data Area Qualifier.
  Binary zero implies 0000SYST.
  TASCII blanks imply the sign-on qualifier.
- 6-255 Reserved for future definition.

On return from the service the A and C registers are set as follows:

A:
- 0 Function performed as requested
- -1 Invalid data
- -3 Area not found (function 3 only)
- -4 Area not an interactive program (function 3 only)
- -6 Calling program must be interactive (functions 3 and 5 only)
- -7 Data not available (only if bit 8 is set)

C: Reflects the condition of the A register.

14.16 $CHAIN

The $CHAIN service allows a program to "chain" to or call another program and to resume execution upon return. When a program chains to another program, an additional set of Virtual Address Registers is assigned and a new link is said to be added to the program chain. A program in a program chain may be chained to another program, etc., which results in the addition of new links in the program chain. Note that a program may only chain to a program of the same type (interactive/control-point, real-time, or monitor program). The calling sequence for the $CHAIN service is:

```
TLO     PARLIST
BLU     $CHAIN
```

where PARLIST is defined as follows:

PARLIST
- +0, +1 Eight-character program areaname in truncated ASCII.
- +2, +3 Eight-character program area qualifier in truncated ASCII.
- +4 Program size in pages. If PARLIST+4 is negative, the Vulcanized program size is used. If PARLIST+4 is zero, the current program size set using the Job Control command is used ($MO PS=nnnn). If PARLIST+4 is positive, the new program size is used. If this size is less than the Vulcanized program size, then the Vulcanized program size is used.

PARLIST +5 Parameter to be passed to the program in the program chain in its K register on initialization.

When control is returned to the calling program, execution continues at the instruction following the BLU instruction. The A register is set to reflect the status of the exiting program as follows:

A: = 0 Normal exit.
> 0 Abort code if the exiting program was aborted.

The C Register is set to reflect the status of the A register.

When a program exits the system using the $EXIT service or if the program aborts and no Contingency Return ($C/RTN) is taken, the link in the chain for the program is removed from the program chain. If the program chain becomes empty for interactive and control-point programs, then Job Control or a default program, if supplied, is reloaded for execution. In the case of real-time and monitor programs, control is returned to the system. If the called program in a chain aborts, the result depends upon the setting of the EA (Exit-on-Abort, Jobcontrol MO command) mode. If Exit-on-Abort mode is selected, the terminal session or control point will be terminated when the called program aborts. If it is not selected, the system will return to the calling program with the error code in the A register.

A program in a program chain may also transfer control to another program using the $NXTPRG service. In such a case the next program is executed at the same link of the program chain as the exiting program, i.e., no new link is added to the program chain. When a program calls the $SEXIT service, the entire program chain is emptied, and control returns to the system. In the case of interactive or control-point programs, if the program making the $SEXIT call is not Job Control, then the call is converted to a regular $EXIT call.

A program may chain to itself (i.e., a program chain may contain more than one copy of the same program). In such cases, each copy of a program is different from the others. The maximum length of a program chain is limited by the value of the CHAINMAX parameter defined during system generation. The default value for the CHAINMAX parameter is 2; and a value of 0 or 1 disables chaining. This parameter may be changed using the Modify System (MS) command at the operator's console. If the maximum length of a program chain is exceeded, the calling program is aborted.

When a program chains to another program, the following actions are taken:

1. The entire logical address space, including any regular DCM or special common DCM allocated by the program, is saved.

2. The Contingency Return ($C/RTN) and the Special Interrupt ($SPINT) linkages, if any, defined by the calling program are saved. Note that the program in the chain must, if needed, define its own Contingency Return ($C/RTN) and Special Interrupt ($SPINT) linkages.

3. A Dump Buffer function (I/O function code '24) is performed on all the assigned LFN's. In case of blocked disc area assignments, this results in the deallocation of the blocking buffers. Hence, upon return from the program in the chain, an Open function (I/O function code '13) should be performed on the LFN's assigned to blocked disc areas before any other I/O operations are performed on them.

INDEX