PERKIN-ELMER

OS/32 SUPERVISOR CALL (SVC)

Reference Manual

48-038 F00 R02

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

FIGURES

TABLES

PREFACE

This manual describes the OS/32 supervisor calls (SVCs) that provide the task interface to OS/32 system services. The information in this manual is intended for assembly language programmers who design application level programs for operation in an OS/32 processing environment.

Chapter 1 presents an overview of all OS/32 SVCs, their functions and the data structure of the SVC parameter block. Chapter 2 describes the Input/Output (I/O) Request SVC1, which is used to request specific I/O services from the OS/32 I/O supervisor. This chapter also presents the SVC1 interface to the Perkin-Elmer Series 3200 I/O Bus Switch Driver. Chapter 3 details 22 general service functions provided by the General Service Functions SVC2. Chapter 4 presents the format for the End of Task SVC3, which is used to terminate task execution. Chapter 5 provides information on user-controlled loading of Link-generated overlays through the Fetch Overlay SVC5. Chapter 6 describes the Intertask Communications SVC6. Chapter 7 details the File Handling Services SVC7, which provides file and device handling functions supported by the file manager and the data communications subsystem. Chapter 8 describes how the Load Task Status Word (TSW) SVC9 is used to replace the current TSW located in the task control block (TCB) with a new user-specified TSW. Chapter 9 provides information on the Overlay Loading SVC10, which handles the automatic loading of overlays generated by Link. Brief descriptions of the Auxiliary Processing Unit (APU) Control SVC13, User SVC14 and Data Communications Device-Dependent I/O SVC15 are given in Chapters 10, 11 and 12, respectively.

Revision 02 includes additions to SVC1 functions for the screen editor along with new status codes for SVC1 device-dependent and device-independent status fields for Mirror Disk. This manual also introduces new SVC1 extended functions to enable 8-bit data transfer, along with documentation of SVC1 functions for the I/O Bus Switch Driver. Changes have been made to SVC7 dealing with 3270 Emulator support and the Perkin-Elmer Series 7000 File Transfer Utility. Also, there are changes made to SVC7 access privileges. In addition, all SVCs previously documented in the System Level Programmer Reference Manual have been added to this manual. These additions include SVC0, SVC2 codes 0, 14, 26 and 27, SVC6 System Task Release, various SVC7 functions and all of SVC13.

This manual is intended for use with the OS/32 R07.2 software release or higher.

For information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1
SUPERVISOR CALLS (SVCs)

1.1 INTRODUCTION

OS/32 provides each task with the support it needs to perform its designated function. In addition to programs that allow a user to design, implement, test and execute tasks, OS/32 provides a number of system services that can be accessed by a task during execution. Included among these services are task timing, interrupt handling, input and output to devices or files, resource allocations and intertask communication and control.

A task accesses a system resource by calling an OS/32 executor routine. An assembly program calls an executor routine by issuing an SVC. Table 1-1 lists the SVCs that access OS/32 system services for assembly tasks. These SVCs are divided into two groups:

- SVCs for general use in both application and system level programs, and
- SVCs for use in system level programs only.

TABLE 1-1 OS/32 SVCs

SVC	FUNCTION
SVC0	User-written SVC
SVC1	Input/output (I/O) request
SVC2 code 0	Make journal entries
SVC2 code 1	Pause
SVC2 code 2	Get storage
SVC2 code 3	Release storage
SVC2 code 4	Set status
SVC2 code 5	Fetch pointer
SVC2 code 6	Convert binary to ASCII hexadecimal or ASCII decimal
SVC2 code 7	Log message
SVC2 code 8	Interrogate clock
SVC2 code 9	Fetch date
SVC2 code 10	Time of day wait

TABLE 1-1 OS/32 SVCs (Continued)

SVC	FUNCTION
SVC2 code 11	Interval wait
SVC2 code 14	Internal reader
SVC2 code 15	Convert ASCII hexadecimal or ASCII decimal to binary
SVC2 code 16	Pack file descriptor (fd)
SVC2 code 17	Scan mnemonic table
SVC2 code 18	Move ASCII characters
SVC2 code 19	Peek
SVC2 code 20	Reserved for sequential tasking machines
SVC2 code 21	Reserved for sequential tasking machines
SVC2 code 23	Timer management
SVC2 code 24	Set accounting information
SVC2 code 25	Fetch accounting information
SVC2 code 26	Fetch device name
SVC2 code 27	Memory management
SVC2 code 29	Unpack fd
SVC3	End of task
SVC5	Fetch overlay
SVC6	Intertask communication and control
SVC7	File handling services
SVC9	Load task status word (TSW)
SVC10	Overlay loading
SVC13	Auxiliary processing unit (APU) control
SVC14	Function determined by user
SVC15	Communications device-dependent I/O

Perkin-Elmer also provides run-time library (RTL) routines that allow a program written in FORTRAN or Pascal to access system services. These routines issue general user SVCs for the task. A system macro library is also available that allows an assembly program to issue an SVC through a system macro call. See the OS/32 Application Level Programmer Reference Manual for an overview of the methods used by the application programmer to access system services.

1.1.1 Supervisor Call (SVC) Parameter Block

Associated with each SVC (except SVC3) is an operating system data structure called a parameter block. The parameter block contains the data required by the OS/32 executor. Each parameter block has a specific length and format. The full length of a parameter block must be reserved even if certain parameters are not required by the particular SVC executor routine.

To issue an SVC, a task must specify the identifying number of the SVC and the address of the SVC parameter block as operands to the call.

Format:

```
SVC n,parblk
```

Operands:

n	is a decimal number specifying the SVC.
parblk	is the label or address of the parameter block that contains the information necessary to execute the call. All parameter blocks must be fullword boundary-aligned.

Execution of an SVC causes an interrupt that is processed by the Internal Interrupt Subsystem. See the OS/32 System Level Programmer Reference Manual for a description of SVC processing by the Internal Interrupt Subsystem.

When building a parameter block structure, it is possible to use the standard symbolic names that have been assigned to the fields and functional values for the parameter block. To obtain these standard names and their definitions, expand the appropriate data structure macro. These macros are contained in the OS/32 System Macro Library Utility, SYSSTRUC.MLB. See the Common Assembly Language Macro/32 (CAL MACRO/32) Processor and OS/32 System Macro Library Utility Reference Manuals.

Use the following Macro Library Utility commands to display the SYSSTRUC.MLB directory:

```
*L MLU32
*ST
PERKIN-ELMER OS/32 MACRO LIBRARY UTILITY 03-340 R00-01
MLU >G MTM:SYSSTRUC.MLB/S
MLU >DIR
12/04/83
$DCB$     $TCB$     $FCB$     $REGS$    $UDL$     $LIB$     $IOB$
$SVC1$    $ERRC$    $SVC13$   $APB$     $SOPT     $RREGS    $EREGS
$UREGS    $PSW      $SPT      $SPTE     $$SPT     $TABL$    $IVT
$SVT      $STE      $PDCB     $DDCB     $PSDCB    $DDE      $MAGDCB
$VFDCB    $SDCB     $EVN      $SCV7SPL  $SD       $SPLMSG   $TMQ
$SDE      $CTX      $RCTX     $TCB      $OCB      $PSTCB    $TSW
$TOPT     $TSTT     $TWT      $TLFL     $TFL      $TPRC     $LTCB
$LIB      $LOPT     $LSG      $RLST     $RSARCPY  $VD       $DIR
$ACB      $FD       $FDE      $PFCB     $FCB      $FFLG     $CCB
$DATB     $DFLG     $DXFL     $SVC1     $S1X0     $SVC1ERR  $SVC4
$SVC5     $SVC6     $SVC7     $SVC7EXT  $SVC13    $APST     $UDL
$IOB      $IOBF     $IOH      $SPOL     $ATF      $GERC     $EFMG
$ESYS     $EMIL     $MERC     $ORT      $ODT      $SPR      $TQE
$TQH      $TG27     INTCPARM  $QH       $IPCB     $IRCB     $ICB
$VFCHARS  $HB       $WAP      $TKQ      $APB      $APRC     $APS
$AOPT     $TTB      $LPMT     $SYP      $QPB$     $QPB      $QPSTAT
$LLE      $ETHSTCM  $ETHDCBS  $ETHSTBF
116 MACROS IN LIBRARY MTM:SYSSTRUC.MLB/G
MLU >
```

Use the following Macro Library Utility command to list the desired structure.

Format:

```
LIST fd,macro
```

Example:

```
 MLU > LI M300:MAR,$SVC1
          MACRO
          $SVC1
          GBLB   %SVC1
          AIF    (%SVC1)&SVC1X
%SVC1     SETB   1
          SPACE  2
SVC1.     STRUC                    STRUCTURE OF SVC-1 PARAMETER BLOCK
          SPACE  1
SVC1.FC   DS     0                 FUNCTION CODE
SVC1.FUN  DS     1                 (ALTERNATE MNEMONIC)
SVC1.LU   DS     1                 LOGICAL UNIT
SVC1.STA  DS     1                 STATUS FIELD
SVC1.DN   DS     1                 DEVICE NUMBER
          SPACE  1
SVC1.SAD  DS     ADC               BUFFER START ADDRESS
SVC1.EAD  DS     ADC               BUFFER END ADDRESS
SVC1.RAD  DS     ADC               RANDOM ADDRESS
SVC1.LXF  DS     4                 LENGTH OF LAST TRANSFER
          ENDS
SVC1X     STRUC
          DS     SVC1.
SVC1.XIT  DS     4                 EXTENDED ITAM OPTION BITS
          ENDS
          SPACE  2
* * * *            THE SVC-1 FUNCTION CODES
          SPACE  1
SV1.CMDF  EQU    X'80'             COMMAND
SV1.READ  EQU    X'40'             READ
SV1.WRIT  EQU    X'20'             WRITE
SV1.BIN   EQU    X'10'             BINARY
SV1.WAIT  EQU    X'08'             WAIT
SV1.RAND  EQU    X'04'             RANDOM
SV1.UPRO  EQU    X'02'             UNCONDITIONAL PROCEED
SV1.IMG   EQU    X'01'             IMAGE MODE
SV1.XIT   EQU    X'01'             ITAM EXTENDED OPT
          SPACE  1
SV1.REW   EQU    X'C0'             REWIND
SV1.BSR   EQU    X'A0'             BACKSPACE RECORD
SV1.FSR   EQU    X'90'             FORWARD-SPACE RECORD
SV1.WFM   EQU    X'88'             WRITE FILE-MARK
SV1.FFM   EQU    X'84'             FORWARD-SPACE FILE-MARK
SV1.BFM   EQU    X'82'             BACKSPACE FILE-MARK
SV1.DDF   EQU    X'81'             DEVICE-DEPENDENT FUNCTION
          SPACE  1
SV1.HLT   EQU    X'80'             HALT I/O
SV1.SET   EQU    X'60'             TEST & SET
SV1.WO    EQU    X'08'             WAIT ONLY
SV1.TEST  EQU    X'02'             TEST I/O COMPLETION
          SPACE  1
* ***************************************************************
&SVC1X    ANOP
          MEND
1 MACRO LISTED TO M300:MAR
MLU >
```

1.2 SUPERVISOR CALL (SVC) ERRORS

The operating system informs the task of any error conditions encountered during SVC processing. Depending on the kind of error encountered, the operating system:

- pauses execution of the task and displays a message on the system console, or
- stores an error code in the error status field of the SVC parameter block and/or sets the condition code.

The first method is used when an error condition occurs as a result of a programming error in the task code (e.g., alignment or illegal instruction fault). If the user wishes the task to handle these errors, the task can take a trap that causes execution to branch to the task trap-handling routine. See the OS/32 Application Level Programmer Reference Manual for more information on trap-handling.

The second method informs the user of the execution status of the SVC executor.

1.2.1 Supervisor Call (SVC) Error Messages

When the user chooses not to take a trap when an illegal instruction fault occurs, the illegal instruction trap bit is set to 0 in the current TSW. On encountering an SVC error, the operating system pauses the task and outputs a message to the system console.

If the SVC error results from attempting to execute an undefined or illegal SVC or from specifying an invalid code for an SVC2, the following message is displayed:

ILLEGAL SVC - INSTRUCTION AT xxxxxx(yyyyyy)

Where:

xxxxxx	is the relative address of the SVC instruction that caused the error.
yyyyyy	is the physical address of the SVC instruction that caused the error.

If an address or alignment error occurs, the following message is displayed:

Format:

```
SVC ADDRESS ERROR - INSTRUCTION AT xxxxxx(yyyyyy)
SVC PARAMETER BLOCK AT xxxxxx(yyyyyy)
```

Where:

xxxxxx	is the relative address of the SVC or parameter block that caused the error.
yyyyyy	is the physical address of the SVC or parameter block that caused the error.

NOTE

Systems equipped with a memory address translator (MAT) display the following message when an address or alignment error occurs:

```
SVC ADDRESS ERROR-INSTRUCTION AT xxxxxx(yyyyyy)
SVC PARAMETER BLOCK AT xxxxxx(yyyyyy)
MEMORY FAULT ADDRESS = xxxxxx(yyyyyy)
```

An address or alignment error can result from any one of the following conditions:

- The address specified for the SVC parameter block lies outside task boundaries.
- The address specified for the SVC parameter block is not aligned on a fullword boundary.
- The address specified for the SVC parameter block is not within a writable segment, which is required for that particular SVC.

1.2.2 Supervisor Call (SVC) Status Codes

When an SVC execution error occurs, the operating system:

- returns an error code to the status field of the SVC parameter block, and/or
- sets bits in the condition code (CVGL) to reflect the results of SVC execution.

The status code returned depends on the particular SVC. Each SVC described in this manual has a defined set of status codes. The condition code (CC), if set for the SVC, depends on the particular SVC. Generally, a CC of 0 indicates successful execution and termination.

A nonzero error code may be returned to the status field of the SVC parameter block as a result of one of the following conditions:

- The buffer to which the SVC parameter block is pointing is not aligned on the proper boundary.
- An SVC parameter block that must point to a task-writable segment is pointing to a buffer outside a writable segment.

To test the CC, use a branch mnemonic that tests for a true condition.

Example:

In the following example, the CC of the program status word (PSW) is tested for the conditions specified by the mask field PSW.CC. PSW.CC is equated to X'F'. If any conditions tested are found to be true, a branch is taken to the location ERROR. For more information on branch instructions, see the Instruction Set Reference Manual or the Processor User's Manual for the appropriate processor model.

```
BTC PSW.CC,ERROR
```

1.3 SVC0: USER-WRITTEN SUPERVISOR CALL (SVC)

SVC0 is reserved for user-written OS/32 executor routines. Before writing an executor routine that can be called by SVC0, the operating system must be modified. This modification can be done dynamically at run-time by an executive task (e-task). However, the SVC executor table contains only halfword entries; the first instruction of the executor routine called by SVC0 must lie within the first 64kb of physical memory.

CHAPTER 2
INPUT/OUTPUT (I/O) REQUEST SUPERVISOR CALL 1 (SVC1)

2.1 INTRODUCTION

SVC1 executes all general I/O data transfer requests and specific command function requests. General I/O data transfer requests refer to either a read or write operation. Before any data can be transferred, the user must specify whether it is a read or write, the address and length of the I/O buffer that will receive or send the data, and the logical unit (lu) assigned to the device or file to which the I/O is directed. These specifications are indicated through certain fields of the SVC1 parameter block.

When requesting a read or write operation, the user must describe the data being transferred and the environment during the transfer in the SVC1 parameter block. For proper execution of a simple data transfer request, specify the:

- structure of the file to or from which a record is being transferred (sequential or random),
- form that the data is in when transferred (ASCII or binary, formatted or image mode), and
- state that the calling task will be in during I/O (I/O proceed, I/O wait or unconditional proceed).

If the device is busy when the data request is made, the user must decide if task execution is to wait, whether to queue the request and proceed or whether to proceed and retry the I/O request later. Link specifies the maximum number of I/O requests that are to be queued at one time. The user also has the option to start I/O and continue task execution, then stop task execution until the I/O is completed. If the device is free and the user wants exclusive access to a record or file (any file type), the user should execute a test and set operation to inform other tasks that the record or file is being used.

Once the read or write operation is completed:

- test for I/O completion (check the condition code (CC), status fields and task queue, or execute a test I/O complete) and, if the status fields indicate that no error has occurred,
- check to verify that all of the specified data was actually transferred (check the length of the data transfer field in the SVC1 parameter block).

All testing and checking for I/O completion can be accomplished through the SVC1 parameter block.

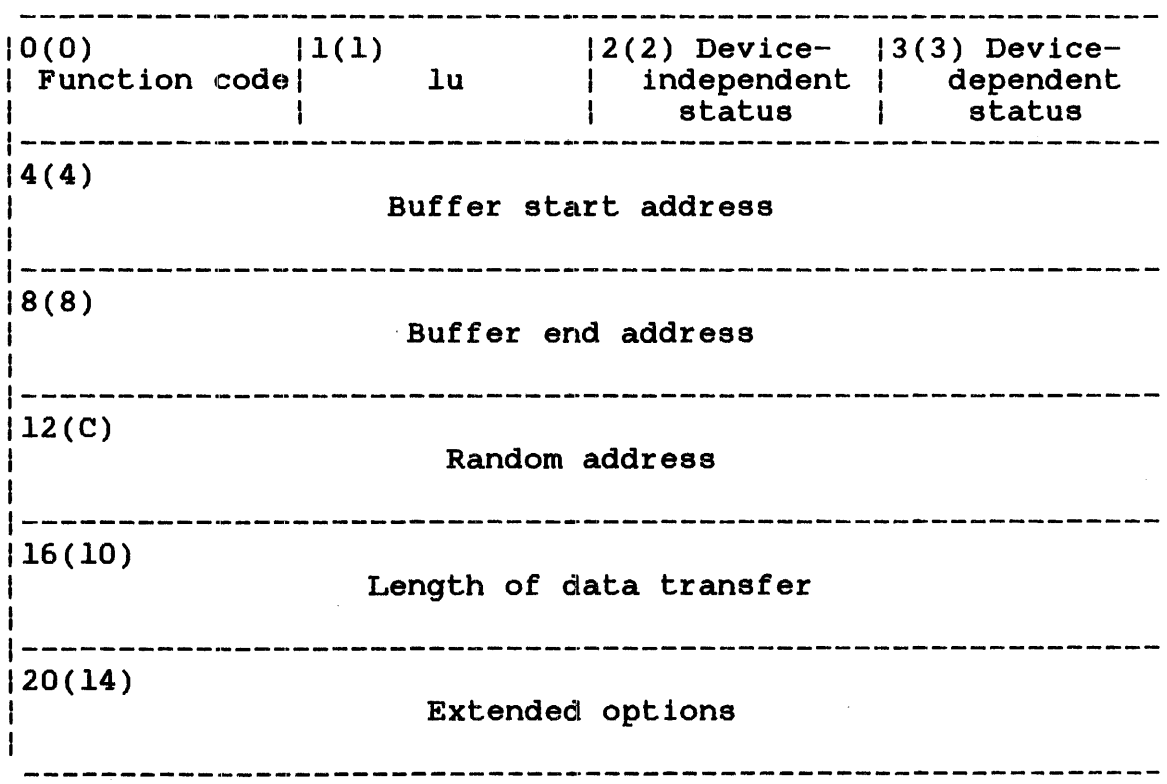
Specific I/O command function requests that can be made through SVC1 include:

- Rewind
- Backspace or forward space record
- Write filemark
- Backspace or forward space filemark
- User-specified, driver-dependent functions (reserved)
- Halt I/O

Before a command function request is issued, the desired command must be specified and the lu must be assigned to the device to which the command is directed. These specifications are indicated in the SVC1 parameter block shown in Figure 2-1.

2.2 SUPERVISOR CALL 1 (SVC1)

The SVC1 parameter block must be 24 bytes long, fullword boundary-aligned and located in a task-writable segment. Location within a writable segment is necessary so the status of an I/O request can be returned to the status fields of the parameter block. All fields in the parameter block are not required for every I/O request but must be reserved (see Figure 2-1).




```
          SVC   1,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'function code'
          DB    X'lu'
          DS    2 bytes for status
          DC    A(buffer start)
          DC    A(buffer end)
          DC    4 bytes for random address
          DS    4 bytes for length of data transfer
          DC    Y'extended options'
```

Figure 2-1 SVC1 Parameter Block Format and Coding

Fields:

Function code	is a 1-byte field indicating whether a request is a data transfer or a command function, and the specific operation to be performed. Bit settings for data transfer requests are described in Table 2-1. Hexadecimal function codes for command function requests are defined in Table 2-2.
lu	is a 1-byte field containing the logical unit currently assigned to the device to which an I/O request is directed.
Device-independent status	is a 1-byte field receiving the execution status of an I/O request after completion. The status received is not directly related to the type of device used.
Device-dependent status	is a 1-byte field receiving the execution status of an I/O request after completion. The status received contains information unique to the type of device used.
Buffer start address	is a 4-byte field used only for data transfer requests and must contain the starting address of the I/O buffer that receives or sends the data being transferred.
Buffer end address	is a 4-byte field used only for data transfer requests and must contain the ending address of the I/O buffer that receives or sends the data being transferred.
Random address	is a 4-byte field containing the address of the logical record to be accessed for a data transfer request; a legal hexadecimal number must be specified in this field if bit 5 of the function code is set to 1.
Length of data transfer	is a 4-byte field used only for data transfer requests. It receives the number of bytes actually transferred as a result of a data transfer request. If an error occurs during data transfer, this field is modified with indeterminate data.
Extended options	is a 4-byte field specifying device-dependent and device-independent extended functions that must be executed by the device when it is servicing a data transfer request.

2.2.1 Data Transfer Requests

Figure 2-2 shows the function code format for data transfer requests, and Table 2-1 defines each function code bit position.

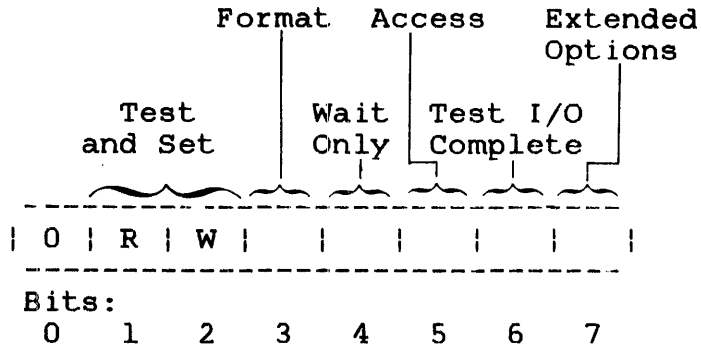


Figure 2-2 Function Code Format for Data Transfer Requests

TABLE 2-1 FUNCTION CODE BIT POSITIONS FOR DATA TRANSFER REQUESTS

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
0	Function code type	0 = data transfer request.
1	Read	1 = read operation. (Bit 2 must be set to 0.)
2	Write	1 = write operation. (Bit 1 must be set to 0.)
1-2	Test and set	1 = test if a specific record in a file is being used by another task.
3	ASCII	0 = the internal data is in the 7-bit ASCII character set and is translated to an equivalent character set appropriate for the external device.

TABLE 2-1 FUNCTION CODE BIT POSITIONS FOR DATA TRANSFER REQUESTS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
3	Binary	1 = the internal data is 8-bit binary and will not be translated. If bit 3 is set and an image I/O extended option is specified, the internal data byte (eight bits) is transferred without translation.
4	I/O proceed	0 = if the device is not busy, return control to the calling task after initiation of data transfer to the device. However, if the device is busy, the request is queued and task execution continues.
	Wait I/O	1 = stop task execution, initiate data transfer to the device, and wait until the completion of I/O.
	Wait only	1 = task execution stops and waits until the completion of all queued I/O proceed requests to the specified lu. When a wait only request is issued, bit 4 is the only bit set in the function code.
5	Sequential random	0 = access the next logical record. 1 = access the logical record specified by the hexadecimal value in the random address field of the parameter block. The association of the hexadecimal values with the logical record must be established before the data transfer occurs.

TABLE 2-1. FUNCTION CODE BIT POSITIONS FOR DATA TRANSFER REQUESTS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
6	Conditional proceed	0 = after the I/O request is issued, put the task into a wait state if the requested device is busy and the total number of queued requests exceeds the maximum. Once the I/O request is completed, the task resumes execution. If the maximum number of queued requests is 1, a pending request causes the task to be placed in a wait state.
	Unconditional proceed	1 = any I/O request made to a device that is busy is rejected if the total number of queued requests exceeds the maximum, and task execution continues.
	Test I/O complete	1 = test to check for the completion of I/O to a specified lu. If a previous I/O proceed request or queued I/O proceed request does exist, the CC is set to X'F'. However, if there is no outstanding I/O proceed request, the CC is set to X'0'. When a test I/O complete request is issued, bit 6 is the only bit in the function code set. If bit 4 is set, it is ignored.
7	Format	0 = the data being transferred is formatted as indicated by the bit 3 setting of the function code and according to the device type specified.

TABLE 2-1 FUNCTION CODE BIT POSITIONS FOR DATA TRANSFER REQUESTS (Continued)

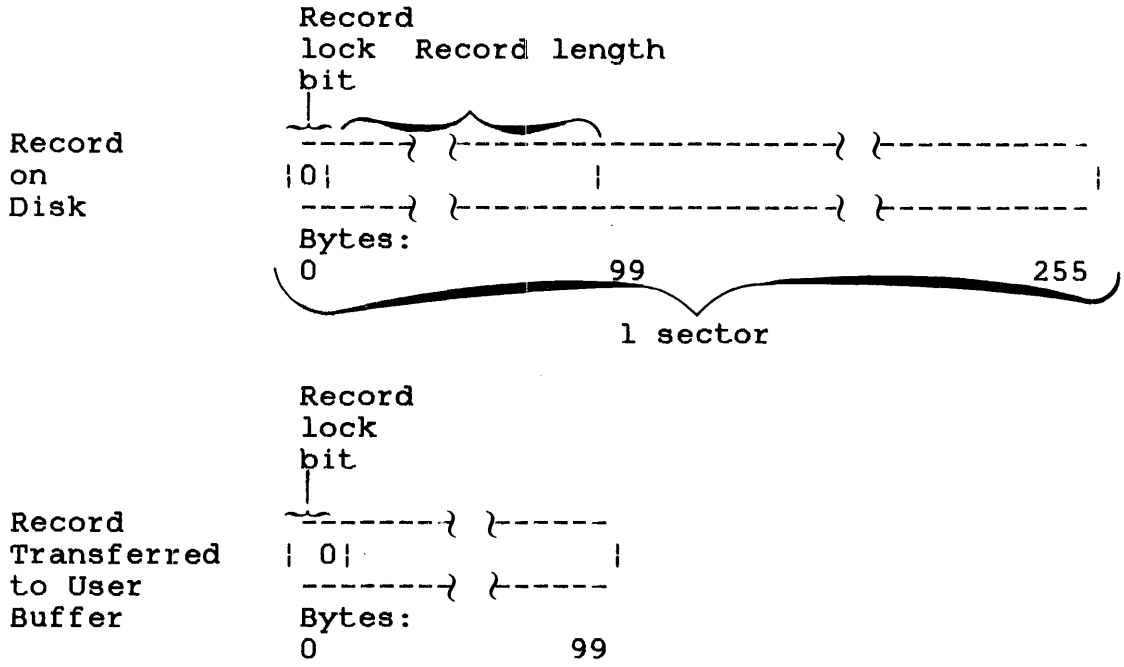
BIT POSITION	BIT NAME	BIT SETTING AND MEANING
7	Extended/ image options	1 = tests the setting of the XSVC1 task option. If XSVC1 is off, an image I/O transfer is performed. If the option is on, the extended options fullword in the parameter block is checked for specified options. When an image I/O is performed, the data being transferred is in image mode and is not formatted. In effect, the user must explicitly specify any control characters such as carriage returns (CRs) or line feeds (LFs) on writes and will receive exactly what is input on reads.

2.2.1.1 Test and Set

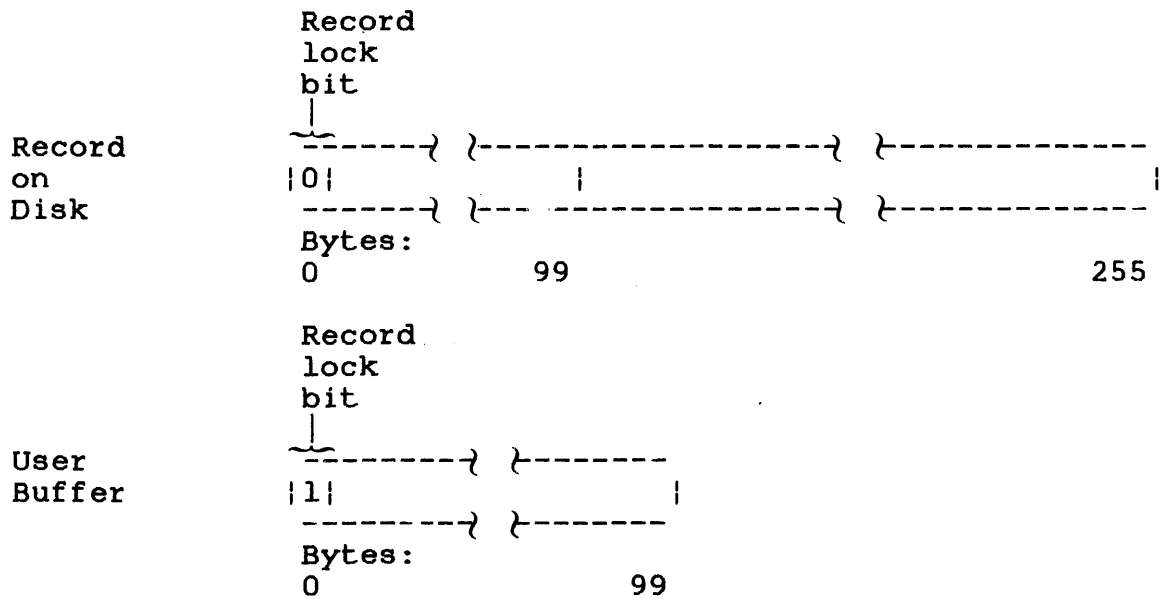
The test and set function can be used to write a program that prevents multiple tasks from modifying a record simultaneously. A task that issues an SVC1 with the test and set bit enabled notifies other tasks that it is using a record by setting the first bit of that record to 1. This bit setting is called a 1-bit record lock. Any task subsequently performing a test and set on the record is informed that the record is being accessed by another task.

To use the test and set function, set both bits 1 and 2 of the function code field to 1. If the test and set operation is used to lock out a record written in binary image mode, make certain that the first bit in the record is initially set to 0. In addition, the size of the user buffer should match the size of the file record. The following diagrams demonstrate how a test and set operation is performed.

In the first diagram, the calling task issued an SVC1 with test and set enabled to read a record into its user buffer specified by the SVC1 parameter block. Notice that the bit setting for the record lock bit is 0, indicating that the record is not being used by another task.

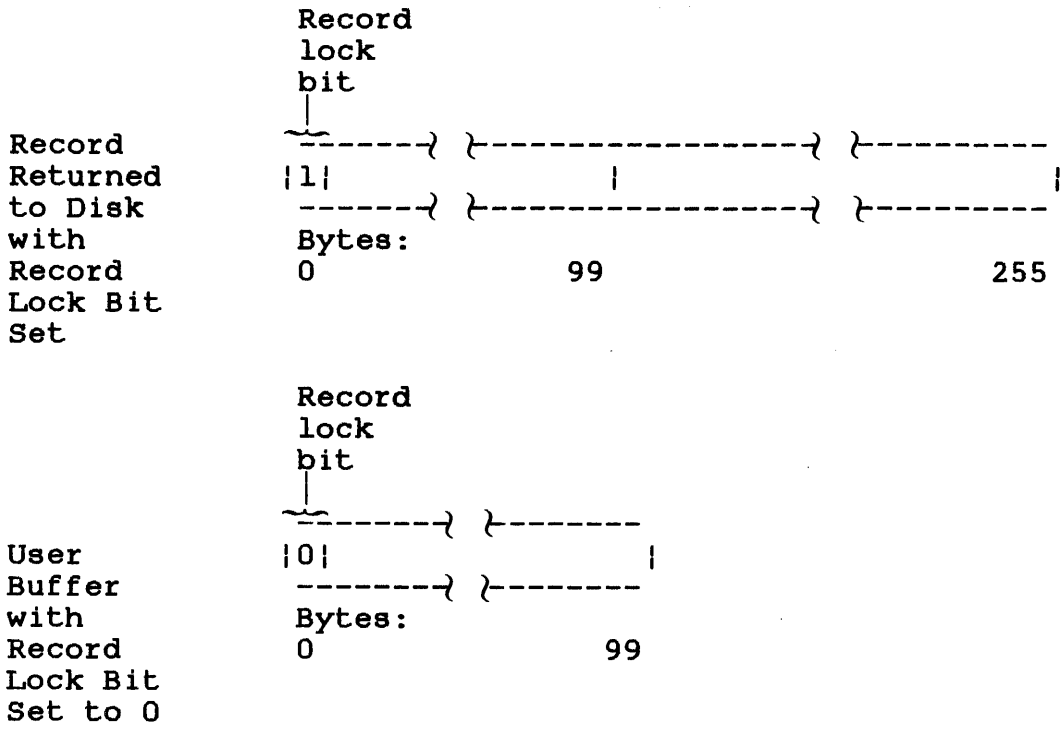


After the record is read into the user buffer, the test and set operation tests the record lock bit. If the bit is 0, the record lock bit in the user buffer is set to 1. The following diagram shows the record lock bit settings after the test operation is performed.



After the record lock bit is tested, the SVC1 test and set function sets the record lock bit on disk to 1 so that other tasks attempting to modify the record are notified that the record is in use. SVC1 sets the record lock bit on disk by copying the contents of the user buffer to the record's original location on disk. In addition, SVC1 sets the CC to X'0' and resets the record lock bit in the user buffer to 0.

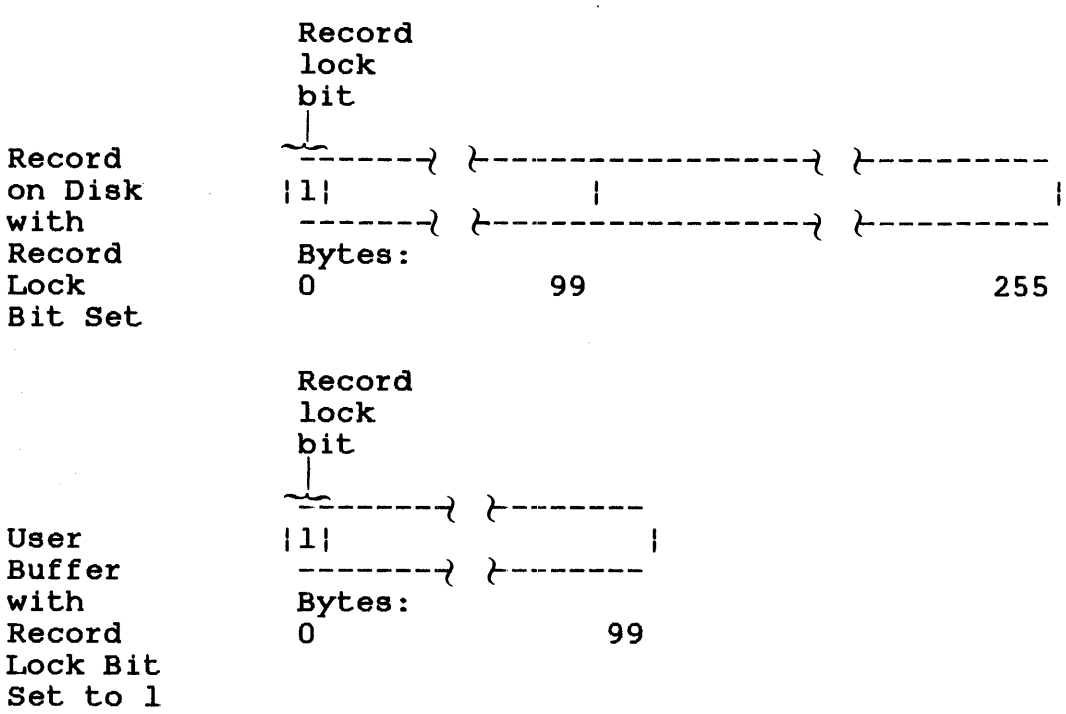
The following diagram shows the results of the completed test and set operation.



Condition Code:

C	V	G	L
0	0	0	0

If the calling task had performed a test and set operation on a record that had a record lock bit setting of 1, the CC would be set to X'F'. The following diagram shows the record lock bit settings and CC resulting from this test and set operation.



Condition Code:

C	V	G	L
1	1	1	1

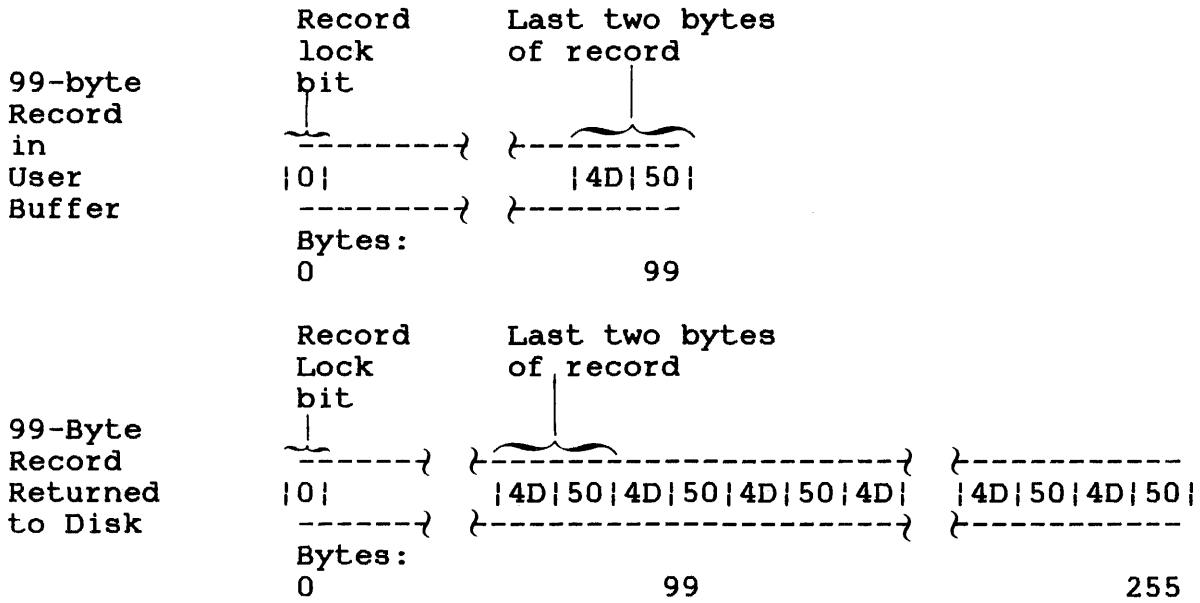
After a test and set operation, a record protection program checks the CC. If the CC is 0, the task can then proceed to modify the record. If the CC is X'F', the task should retry the test and set operation before attempting to modify the record.

To unlock the record on disk, the task that set the record lock bit should write the record in its user buffer back into its original location on disk, whether or not the task modified the record.

If the size of the user buffer is less than the size of the record, the following occurs when the record in the user buffer is written back to disk:

- If the record in the user buffer is written back to an indexed file, the remaining bytes of the record are filled with zeros.
- If the record in the user buffer is written back to a contiguous nonbuffered indexed or extendable contiguous file, the last two bytes of the record are propagated to the right until the remaining bytes of the record are filled.

The following diagram illustrates how a user buffer smaller than a 256-byte contiguous file record is returned to disk. Notice that the last two bytes of the record in the user buffer are propagated to the right to fill a 256-byte sector on disk.



The test and set operation can be executed as a wait I/O or I/O proceed request. Setting the CC during a test and set operation occurs only when wait I/O is specified. (Bit 4 of the function code is set to 1.) However, if an I/O proceed is requested (bit 4 of the function code is set to 0), setting the CC is not useful because it could be changed at any time during task execution when data transfer and task execution take place concurrently. Therefore, check the record lock bit in the buffer to determine whether the record is currently being used. See the OS/32 System Level Programmer Reference Manual for the devices supporting test and set.

The following sample program demonstrates how the test and set function can be used to write a program that provides record protection.

Sample Program:

```
*
*         TEST AND SET EXAMPLE
*
*         PRIOR TO PROGRAM EXECUTION ASSIGN LU 1 AND LU 2
*         TO A TEXT FILE CREATED BY EDIT32.
*
*
*         LU 1 AND LU 2 SIMULATE ACCESS OF THE FILE BY TWO
*         TASKS.
*
TESTSET   PROG  TEST AND SET EXAMPLE
          ENTRY TESTSET
TESTSET   EQU   *
*
          LIS   1,0                    SET UP FIRST RECORD NO.
LOOP      EQU   *
          ST    1,PB1+IO.RECNU         FOR LU 1
          ST    1,PB2+IO.RECNU         FOR LU 2
*
          SVC   1,PB1                  READ & TEST RECORD ON LU 1
          BZ    CONT05                 OK; RECORD IS NOT LOCKED
          PAUS  ERROR; RECORD IS LOCKED; SHOULD BE FREE
CONT05    EQU   *
          LA    14,PB1                 PB ADR FOR EOF TEST
          BAL   15,@IOERR              CHECK FOR END-OF-FILE
*
*
CONT10    EQU   *
          SVC   1,PB2                  READ AND TEST RECORD ON LU 2
          BM    CONT20                 OK;RECORD IS LOCKED
          PAUS  ERROR; RECORD IS FREE; SHOULD BE LOCKED
*
*
CONT20    EQU   *
          L     2,B1S                  MANIPULATE RECORD DATA
          ST    2,B1S
*         WRITE RECORD BACK TO FLE AND UNLOCK IT
          SVC   1,PB3
*
*
          SVC   1,PB2                  READ AND TEST RECORD ON LU 2
          BZ   CONT30                  OK; RECORD IS FREE
          PAUS  ERROR; RECORD IS LOCKED; SHOULD BE FREE
*
*
CONT30    EQU   *
          SVC   1,PB1                  READ AND TEST RECORD ON LU 1
          BM    CONT40                 OK; RECORD IS LOCKED
          PAUS  ERROR; RECORD IS FREE; SHOULD BE LOCKED
```

```
*
*
CONT40   EQU   *
         L     3,B2S                MANIPULATE RECORD DATE
         ST    3,B2S
*        WRITE RECORD BACK TO FILE AND UNLOCK IT
         WRITE LU=2,RECNUMB=(1),ADDR=B2S,ENDADDR=B2E
*
*
         AIS   1,1                  INCREMENT RECORD COUNTER
         B     LOOP                 DO NEXT RECORD UNTIL EOF
*
PB1      IOPCB FUN=X'76',LU=1,ADDR=B1S,ENDADDR=B1E,RESTART=CONT05
*
*
PB2      IOPCB FUN=X'76',LU=2,ADDR=B2S,ENDADDR=B2E
*
*
PB3      IOPCB FUN=X'36',LU=1,ADDR=B1S,ENDADDR=B1E,RECNUMB=(1)
*
*
         ALIGN ADC
B1S      DS    80                   BUFFER FOR LU 1
B1E      EQU   *-1
*
*
B2S      DS    80                   BUFFER FOR LU 2
B2E      EQU   *-1
*
*
         END
```

2.2.1.2 Input/Output (I/O) Proceed

An I/O proceed request is initiated when bit 4 of the function code is set to 0 and a read or write operation is specified.

If the device is free when a data transfer request is made with I/O proceed specified, task execution and data transfer take place concurrently. When the I/O is completed, the status of the data transfer is returned to the status fields in the parameter block. An illegal function code or illegal lu causes the status to be returned to the status fields before data transfer starts, resulting in rejection of the I/O proceed request. Since task execution and data transfer take place concurrently, the task must check for the completion of I/O. There are five ways to check for I/O completion:

- Execute a test I/O complete operation.
- Monitor the status fields in the SVC1 parameter block issuing the request.

- Take a trap when I/O is completed and branch to a service routine.

- Issue a wait I/O request to the device specified by the SVC1 making the request. This function will stop task execution until I/O is completed.

- Queue I/O requests by specifying the IOBLOCK parameter of the Link OPTION command and issuing the wait only function. This will stop task execution until all queued requests to a specified device are completed.

An SVC1 I/O proceed request to an indexed file executes in a different manner than an I/O proceed to other file types or devices. See the OS/32 Application Level Programmer Reference Manual for more information on I/O operations to indexed files.

2.2.1.3 Queuing Input/Output (I/O) Requests

When SVC1 issues an I/O proceed request to a device that is busy, the request is placed on the calling task's I/O control block, and task execution continues. The request is serviced when the device is free. Normally, each task has only one I/O control block on which to queue an I/O request. To queue more than one request, use the IOBLOCK parameter of the Link OPTION command to assign more blocks to the task.

Format:

OPTION IOBLOCK={b / 1}

Parameter:

b — is a decimal number from 1 to 65,535 indicating the maximum number of I/O control blocks assigned to a task. Each I/O control block can contain one queued I/O request. If this option is not specified by the user, Link automatically assigns one I/O control block to the task.

2.2.1.4 Conditional Proceed

If the number of queued requests exceeds the maximum number of I/O blocks assigned to the task and bit 6 of the function code is set to 0, SVC1 places the task in a wait state until one of the queued requests is serviced. Task execution resumes when the number of queued requests equals the maximum number set by Link.

The number of I/O requests a task can issue before going into the wait state is determined by the formula:

b + 1 + number of logical units assigned to task

Parameter b is the number of I/O control blocks assigned to the task.

2.2.1.5 Unconditional Proceed

To prevent the task from going into the wait state when the maximum number of requests specified by Link are queued, set bit 6 of the function code to 1. This code allows the task to reject all I/O requests made to a busy device after the maximum number of requests are queued. When a request is rejected, a status of 0 is sent to the device-independent status field, and the CC is set to X'F'. The user can retry the rejected I/O request during task execution.

2.2.1.6 Wait Input/Output (I/O)

To stop task execution during a read or write operation, use the wait I/O function. A wait I/O request is initiated when bit 4 of the function code is set to 1 and a read or write operation is specified.

If the device is free when a data transfer request is made with wait I/O specified, task execution stops, I/O is initiated, and the task waits to resume until I/O is completed. Status of the data transfer is returned to the status fields when the I/O is completed. If the device is busy when a data transfer request is made with wait I/O specified, the request is queued and task execution is suspended until the queued request is serviced and I/O is completed. Task execution then resumes.

2.2.1.7 Wait Only

A wait only request stops task execution until all I/O proceed requests to the specified lu (including queued requests) are completed. When the last queued I/O proceed request is completed, task execution continues. The status of the last completed I/O proceed request is returned to the status field of its respective SVC1 parameter block.

To issue the wait only request, set the SVC1 function code field to X'08' and the lu field to the appropriate device. A nonzero status code will be returned to the status field of the SVC1 wait only parameter block if any of the following conditions occur:

- The lu is illegal (code X'81').
- The lu is unassigned (code X'81').
- The wait only request is issued for a pseudo device without SVC interception (code X'C0').

2.2.2 Command Function Requests

All command function requests and task execution take place concurrently. Queued requests are handled the same way as conditional proceed data transfer requests. When the I/O is completed, the status of the command function is returned to the status fields in the parameter block. An illegal function code or illegal lu causes the status to be returned to the status fields before the command function starts. This results in rejection of the command function request.

Since task execution and command function requests take place concurrently, the task must check for I/O completion. These three methods are used to check for I/O completion:

- Execute a test I/O complete operation.
- Monitor the status fields in the parameter block for the command function status to be returned.
- Issue a wait only request to the device specified by the SVC1 making the request. This function stops task execution until I/O is completed.

Table 2-2 defines the function codes for command function requests.

TABLE 2-2 FUNCTION CODES FOR COMMAND FUNCTION REQUESTS

FUNCTION CODE	MEANING
X'C0'	Rewind - A rewind operation is to occur on the specified lu.
X'A0'	Backspace record - The device assigned to the lu is to backspace one record length.
X'90'	Forward space record - The device assigned to the lu is to move forward one record length.
X'88'	Write filemark - A filemark is to be written at the current pointer position on the device assigned to the lu.
X'84'	Forward space filemark - The device assigned to the lu is to move forward past the next filemark to the beginning of the next file.
X'82'	Backspace filemark - The device assigned to the lu is to backspace to the previous filemark. For disk files, this positions the pointer to the beginning of the previous file. For magnetic tape files, the tape is positioned at the end of the previous file.
X'81'	No echoplex - The device chooses no echoplex for an image I/O and selects 8-bit no parity as an option for SVC1 I/O. By preceding an I/O with an additional SVC1 with a function code Y'10000 0000', no echoplex is set in the data control block. This applies to device 156 and 157 drivers.
X'80'	Halt I/O - Cancel all previous I/O proceed requests to the specified lu.

2.2.2.1 Halt Input/Output (I/O)

When a halt I/O request is issued, any previous I/O proceed requests, whether they are in progress or queued to the specified lu, are cancelled. When the I/O is terminated, the task that issued the I/O proceed request takes a trap (if enabled), the request is queued, and the status of the I/O operation (data transfer or command function) is returned to the status fields of the parameter block issuing the request. The time of actual termination is asynchronous to the time the halt I/O is issued. The independent status codes are listed in Table 2-3 and the dependent status codes are listed in Table 2-4.

When an I/O request is issued to an lu and a previous I/O proceed request exists for that same lu, the second request and any subsequent requests to that lu cannot be serviced until the previous I/O request is completed. By issuing a halt I/O request, the first I/O request is cancelled, allowing I/O requests issued after the cancellation to be started on the device.

If the IOBLOCK option was specified by Link and at least one I/O request to a specified lu is queued, execution of a halt I/O request cancels any I/O to that specified lu already queued or in progress. See the OS/32 System Level Programmer Reference Manual for the devices supporting the halt I/O request.

2.2.3 Logical Unit (lu)

An lu is a decimal number ranging from 0 to 254. The highest lu number to which a task can be assigned is determined by the lu parameter of the Link OPTION command. After loading the task into memory, the lu should be assigned to a particular file or device through SVC7 or an ASSIGN command. If no actual I/O operation is desired, the lu should be assigned to NULL:, causing a no-operation (no-op) to occur.

2.2.4 Device-Independent Status

Logical units provide device-independent I/O by causing all I/O requests to be made directly to the lu and not to the device. The execution status of an I/O request that is independent of the physical characteristics of the device being used is returned to the device-independent status field of the parameter block (see Table 2-3). The data remaining in this field from a previous I/O request is not modified until a subsequent I/O is completed or an error occurs.

TABLE 2-3 DEVICE-INDEPENDENT STATUS CODES

STATUS CODE	MEANING
X'C0'	Illegal function - An error is present in the function code; the requested function is not supported by the device or assigned access privilege or the buffer transfer is too small. (When using tape, minimum buffer size is four bytes.)
X'A0'	Device unavailable - The device is either inoperative or not configured into the system.

TABLE 2-3 DEVICE-INDEPENDENT STATUS CODES (Continued)

STATUS CODE	MEANING
X'90'	End of medium (EOM) - The I/O directed to the lu reached the physical end of the device; e.g., end of tape. During magnetic tape operations, this status can be combined with one of the next three status codes, yielding X'98', X'94' and X'92'.
X'88'	End of file (EOF) - The logical end of file specified by the assigned lu was reached.
X'84'	Unrecoverable error - An error occurred and the I/O request, which terminates task execution, cannot be retried.
X'82'	Parity - An even or odd parity error occurred on a data transfer request. Recoverable error - The I/O request is recoverable and can be retried. A write request was issued to a write-protected device. No I/O currently being processed - If a halt I/O request is executed, this bit is set, indicating that no I/O is being processed at this time.
X'81'	Illegal or unassigned lu - The lu specified in the parameter block is either incorrect or was not previously assigned.
X'00'	Normal execution or successful I/O is completed, and no error occurred.

2.2.5 Device-Dependent Status

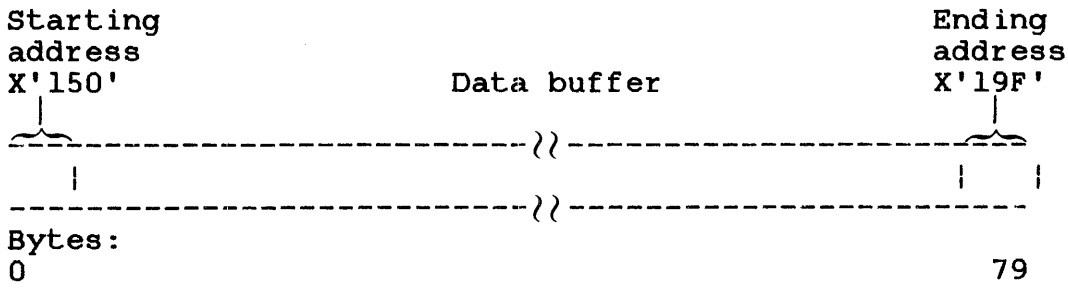
The execution status of an I/O request that is directly related to the unique characteristics of the device being used is returned to the device-dependent status field of the parameter block (see Table 2-4). The data remaining in this field from a previous I/O request is not modified until a subsequent I/O request is completed or an error occurs.

TABLE 2-4 DEVICE-DEPENDENT STATUS CODES

STATUS CODE	MEANING
X'85'	Exhausted retries on seeks - Seeks on disk devices have been retried the maximum number of times.
X'84'	Queued I/O terminated - A queued I/O request is terminated because a previous I/O request failed.
X'83'	Device is write-protected - A write operation to a write-protected device occurred.
X'82'	Read/write time-out - A read or write time-out condition occurred.
X'81'	Terminated by halt I/O - I/O was terminated by a halt I/O operation.
X'00'	Normal execution - I/O was completed and no error occurred.

2.2.6 Buffer Start/Buffer End Addresses

The buffer start/buffer end addresses specify the buffer to be used for data transfer requests. The start address is the first byte in the buffer. The end address is the last byte in the buffer that is included in the transfer.



2.2.7 Extended Options

If bit 7 of the function code is set to 1 and the XSVC1 option was specified at Link time, the options specified by the SVC1 extended option field are executed. The extended options fullword format is dependent upon the device to which an I/O request is directed. In general, there are two formats: one for nonmagnetic tape devices and one for magnetic tape devices.

2.2.7.1 Nonmagnetic Tape Devices

If a device is supported by the data communications subsystem, the extended options provide device-dependent, communication-dependent and device-independent features when a read or write operation is performed.

Figure 2-3 illustrates the fullword format of the extended options field of the SVC1 parameter block for devices supported by the communications subsystem.

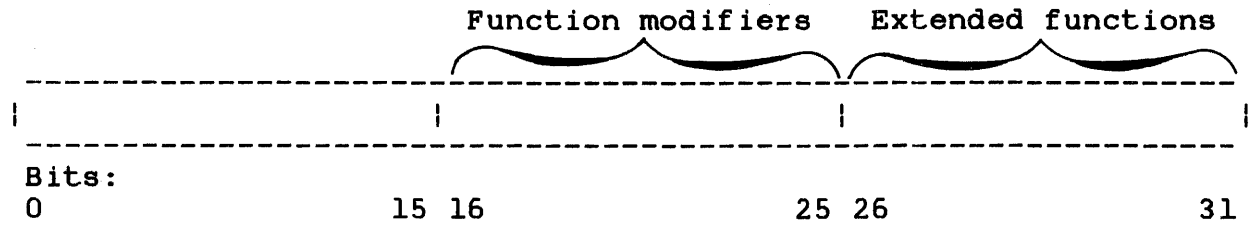


Figure 2-3 Extended Options Fullword Format for Nonmagnetic Tape Devices

Bits 0 through 15 are for general use in both local and remote communications.

Bits 16 through 25 are used to expand a function's capability. For example, the write edit function can be expanded to write blinking by using a function modifier.

Up to 64 device-dependent I/O functions can be specified by bits 26 through 31. These extended functions are mutually exclusive; however, an I/O with multiple requests or operations can be performed.

Table 2-5 describes the SVC1 extended options that can be specified for both local and remote communications. See the OS/32 Basic Data Communications Reference Manual for a listing of device-dependent extended functions along with their applicable function modifiers.

TABLE 2-5 SVC1 EXTENDED OPTIONS FOR LOCAL AND REMOTE COMMUNICATIONS

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
0	Connect (CON)	1 = terminal manager answers a telephone ring on a dial-in line during a read or write line initialization sequence.
1	Disconnect (DCT)	1 = terminal manager disconnects from a switched line following final data transfer.
2	Image/format (IMG/FMT)	0 = data being transmitted is in image mode and is not formatted. 1 = terminal manager performs normal record buffering, inserts or deletes line control characters and recognizes appropriate data format control characters on transmitted data.
3	No echo read	1 = no characters will be echoed on read. Specifies the state of input character echo. Applies to each read request only.
4	Transparent mode	1 = any nonprintable data will not be translated by driver. Specified the transparent state for the read request. Meaningful for a formatted read only.
5	Monitor read	1 = any data input will be placed on driver type-ahead queue. Specifies a read request is not intended to get input data. Illegal unless type-ahead has been turned on. Break key and halt I/O terminates the request.

TABLE 2-5 SVC1 EXTENDED OPTIONS FOR LOCAL AND REMOTE COMMUNICATIONS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
6	Read prompt	1 = data being written from a buffer. Specifies data is to be written from a buffer until a CR or buffer limit. Then read (without line formatting) is to be performed. Meaningful for formatted reads only.
7		0 = reserved
8	Vertical forms control (VFC)	1 = requests VFC option for an ASCII I/O operation.
9-11		000 = reserved.
12	Bit check	1 = when set, the device-dependent bits are checked. This option deals exclusively with mirror disk configurations. This bit is set in conjunction with bits 16, 17, 19, 20 or 21.
13-15		0000000 = reserved
16	Synchronization complete	1 = set when synchronization of mirror disk is complete; it clears the resynchronization in progress field in the DCB DXFL.REB.
17	Synchronization of data	1 = when set, data is read from the primary disk and written to the same sector on the secondary disk. (Pertains to mirror disk configurations.)
18		0000000 = reserved
19	Mirror read	1 = when set, data is read from the same position on both the mirror disks into a double size I/O buffer.

TABLE 2-5 SVC1 EXTENDED OPTIONS FOR LOCAL AND REMOTE COMMUNICATIONS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
20	SWOP	1 = when set, the primary and secondary disks of a mirrored pair are exchanged. The disk that was the secondary disk becomes the primary disk and vice versa. Read operations are now performed on the new primary disk.
21	Read failure	1 = when set, the calling task is notified when a read operation fails on the primary disk. (Pertains to mirror disk configurations.)

2.2.7.2 Magnetic Tape Devices

The extended options fullword format differs when I/O is being directed to a magnetic tape device. Figure 2-4 illustrates the fullword format of the extended options field of SVC1 parameter blocks used for magnetic tape I/O operations.

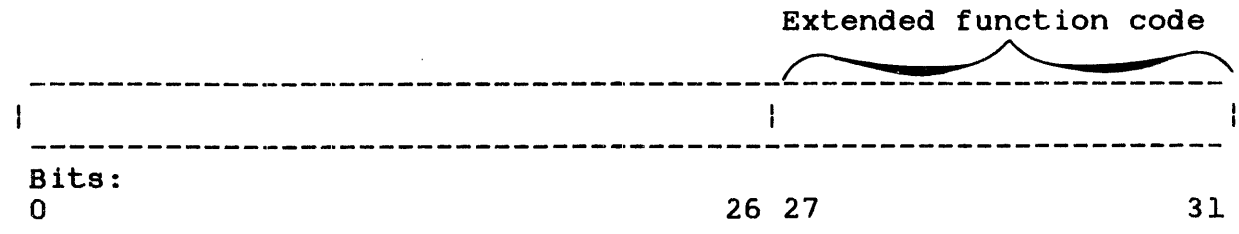


Figure 2-4 Extended Options Fullword Format for Magnetic Tape I/O Operations

If the extended function code requires an additional parameter, the most significant bits (MSBs) (0 through 7) contain the parameter value.

Bits 8 through 26 are not used during magnetic tape I/O operations. Bits 27 through 31 contain the extended function code that indicates the type of I/O operation to be performed. The extended function codes available for use in this field are dependent upon the standard function code setting in the SVC1 parameter block. Table 2-6 contains the extended function codes available when the standard function code bit setting indicates a control operation.

TABLE 2-6 EXTENDED FUNCTION CODES FOR CONTROL OPERATIONS

EXTENDED FUNCTION CODE VALUE	OPERATION/EXPLANATION
0	Rewind and unload - The tape is rewound to its beginning, then unloaded. Requires hardware support.
1-6	Reserved
7	Create a gap - The drive is instructed to erase a section of tape (approximately 3 to 3.5 inches) in the forward direction.
8	Read drive status - A task can read eight status halfwords into the buffer space specified in the SVC1 parameter block. The status returned depends on the type of drive in use. See the High Performance Magnetic Tape System (HPMTS) 125 Programming Manual for a list of the status halfwords. Requires hardware support.
9	Reserved
10	Erase tape - Erases a variable length of tape, beginning at the current position. The length of tape erased is determined by the following formula: $\text{Length of Tape Erased} = \frac{\text{Number of Bytes in User Buffer}}{\text{Current Tape Density}}$

TABLE 2-6 EXTENDED FUNCTION CODES FOR CONTROL OPERATIONS (Continued)

EXTENDED FUNCTION CODE VALUE	OPERATION/EXPLANATION
10 (Continued)	The result is rounded up to a multiple of the length of a hardware gap (approximately 3 to 3.5 inches). The maximum number of bytes that can be erased depends upon the tape density (see Table 2-7). If an erase tape request exceeds the maximum number of bytes for the current tape density, the operating system will erase the maximum number of bytes, then output a message indicating that the remaining bytes in the buffer were not erased. The erase tape function is illegal if the tape is at load point. **NOTE** For device code 65, the current density is assumed to be 800 bits per inch (bpi). If the current density for device code 65 is 1600 bpi, the length of tape erased is twice as long as requested.
11-31	Reserved

TABLE 2-7 MAXIMUM NUMBER OF BYTES ERASED

TAPE DENSITY (BPI)	NUMBER OF BYTES
800	200,000
1,600	400,000
6,250	1,000,000

Table 2-8 contains the extended function code available when the standard function code bit setting indicates a data transfer operation.

TABLE 2-8 EXTENDED FUNCTION CODES FOR DATA TRANSFER OPERATIONS

EXTENDED FUNCTION CODE VALUE	OPERATION/EXPLANATION
0	No extended functions - The bit settings of the standard function (byte 1 of the SVC1 parameter block) are read and used to determine the operation to be performed.
1	Read backward - The tape drive reads previous records on a tape while the tape is moved in the backward (rewind) direction. The task buffer is filled, from start address to end address, with bytes in the order they are read; i.e., reverse. If an error occurs during a read backward operation, the magnetic tape drive performs retries on that operation up to a number of times corresponding to the value set in the system generation (sysgen) macro library. (The read bit of the SVC1 function code should be set.) Requires hardware support.
2	Gapless operation - The driver reads or writes multiple data buffers to or from magnetic tape with no interrecord buffer gaps, using only one SVC1. Gapless operation requires the use of a special SVC1 parameter block. The read or write bit in this parameter block should be set. Gapless operation is explained in Section 2.3. Requires hardware support.
3	Gapless operation with buffer transfer reporting - The driver reads or writes multiple data buffers to or from magnetic tape with no interrecorder gaps, using only one SVC1. The task receives a buffer gap each time the driver uses another buffer. Gapless operation requires the use of a special SVC1 parameter block. The read or write bit in this parameter block should be set. Gapless operation is explained in Section 2.3. Requires hardware support.

TABLE 2-8 EXTENDED FUNCTION CODES FOR DATA TRANSFER OPERATIONS (Continued)

EXTENDED FUNCTION CODE VALUE	OPERATION/EXPLANATION
4	Read forward and ignore data transfer errors - The tape drive reads from the tape and ignores data transfer errors if encountered. If a data transfer error occurs, the status halfword is set to indicate normal completion of the read. The position of the tape after the read is the same as if no error had occurred. Since some errors terminate data transfer, the user should check the length of the data transfer field to verify that all of the specified data was actually read. (The read bit of the SVC1 function code should be set.)
5	Read backward and ignore data transfer errors - The tape drive reads previous records on a tape while the tape is moved in the backward (rewind) direction and will ignore data errors, if encountered. If a data error occurs, the status halfword is set to indicate normal completion of the read. The position of the tape after the read is the same as if no error had occurred. Since some errors terminate data transfer, the user should check the length of data transfer field to verify that all of the specified data was actually read. The user buffer is filled, from start address to end address, with bytes in the order they are read; i.e., reverse. (The read bit of the SVC1 function code should be set.) Requires hardware support.
6	User control of retries for data transfer errors - If an error occurs during a data transfer operation, the magnetic tape drive will repeat the operation up to the number of retries specified by the user in the first byte of the extended options field. The maximum number of retries that can be specified for a read operation is 255. The maximum number of retries that can be specified for a write operation is 45. (The read or write bit of the SVC1 function code should be set.)

TABLE 2-8 EXTENDED FUNCTION CODES FOR DATA TRANSFER OPERATIONS (Continued)

EXTENDED FUNCTION CODE VALUE	OPERATION/EXPLANATION
	NOTE If extended function code 6 is not specified, the number of retries defaults to the value set in the sysgen macro library.
7	Read backwards and allow user control of retries for data transfer errors - The tape drive reads previous records on a tape while the tape is moved in the backward (rewind) direction. The user buffer is filled, from start address to end address, with bytes in the order they are read; i.e., reverse. If an error occurs, the magnetic tape drive repeats the operation up to the number of retries specified by the user in the first byte of the extended options field. The maximum number of retries that can be specified is 255. (The read bit of the SVC1 function code should be set.)
8-31	Reserved

In both cases, extended function codes are mutually exclusive, that is, only one extended function code can be specified in a single SVC1.

2.2.7.3 Device-Dependent Status Codes for Magnetic Tape Operations

The device-dependent and device-independent status fields of the SVC1 parameter block indicate the execution status of an I/O operation performed to a magnetic tape. Table 2-9 lists the status codes returned to these fields. Additional status codes for gapless I/O operations are listed in Table 2-12. A magnetic tape I/O operation ceases upon detection of most of these errors.

TABLE 2-9 MAGNETIC TAPE DEVICE-DEPENDENT STATUS CODES

STATUS CODE	MEANING
8282	Time-out - A read or write time-out condition occurred during data transfer.
8283	Device write-protected - A write, write filemark, create gap, or erase tape operation was attempted to a write-protected device.
82F9	Maximum buffer size exceeded - The buffer for the erase tape control operation is too large.
82FA	Retries exhausted - A read, read backward, or write operation was retried the maximum number of times.
82FC	Time-out - A control operation time-out occurred.
82FD	Time-out - A read, read backward, read drive status, write or write filemark time-out condition occurred.
82FE	Read backward at load point - Load point was reached before a read backward operation terminated.
82FF	Time-out - A read, read backward or write time-out condition occurred while waiting for a prior operation to be completed.
8301	Short read - The buffer specified was too small for the tape block. This status is supported only by the high performance tape systems.
8400	Bottom of tape/end of tape check malfunction - An error occurred during an attempt to position the tape to determine whether the beginning or end of tape was detected.
84FB	Selector channel (SELCH) malfunction - The SELCH malfunctioned during a read, read backward or write operation. Retries exhausted for write filemark - A write filemark operation was retried the maximum number of times. Retry malfunction - An error occurred while attempting to position the tape to retry a read, read backward, write or write filemark operation that resulted in a recoverable error.
A000	Device unavailable - the device is either inoperative or not configured into the system.

TABLE 2-9 MAGNETIC TAPE DEVICE-DEPENDENT STATUS CODES (Continued)

STATUS CODE	MEANING
C000	Illegal function - The function code indicated a data transfer operation, but neither the read nor write bit was set. The function code indicated a control operation, but none of the other bits in the function code were set. The function code indicated an extended control operation, but the extended SVC1 task option was disabled. The requested function is not supported by the device or assigned access privileges. Illegal extended function code - an undefined function, or a function not supported by the specified tape drive, was indicated. The extended function code indicated a read operation, but the standard function code has the write bit set. Buffer size too small - the buffer for a read, read backward, or write operation was less than four bytes. The buffer for the read drive status was smaller than 16 bytes. Erase tape at load point - an erase tape operation was attempted when a tape was at load point. User retries too large - the maximum number of retries specified for a write operation was greater than 45.

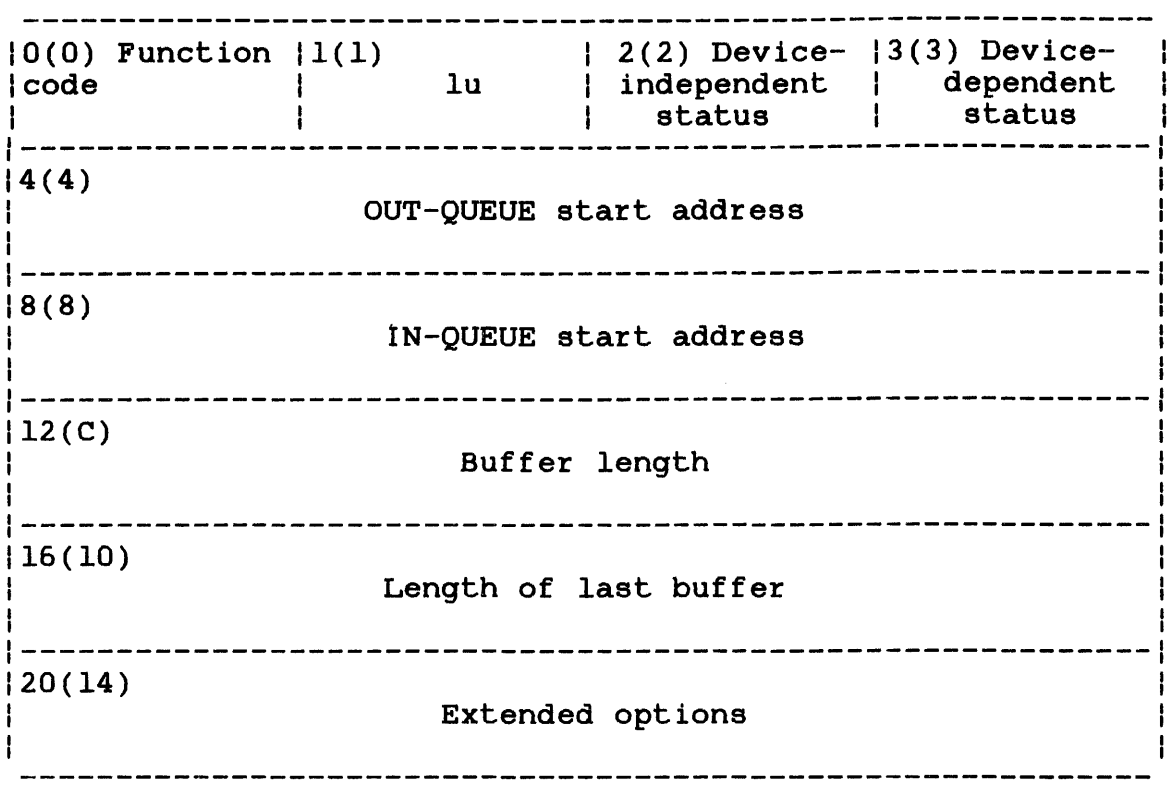
2.3 GAPLESS INPUT/OUTPUT (I/O) OPERATIONS

Data transfer operations in gapless mode consist of a task reading or writing data to or from a magnetic tape with no interrecord gaps, using only one SVC1. A task can have only one ongoing gapless SVC1 at a time. The format of a gapless mode SVC1 parameter block differs from the standard SVC1 parameter block. The gapless SVC1 parameter block cannot be reused until the gapless operation has been completed. To perform a gapless I/O operation, the XSVC1 Link option must be specified before an I/O request is issued. Then, the task must issue an SVC1 call that specifies, among other things, a pair of buffer queues, the IN-QUEUE and the OUT-QUEUE. The driver takes buffers from the IN-QUEUE and returns used buffers to the OUT-QUEUE. The task processes the buffers from the OUT-QUEUE and returns these buffers to the IN-QUEUE for reuse by the driver.

The use and reuse of buffers during gapless I/O enables an amount of data much greater than memory capacity to be transferred by breaking the data into smaller segments, then transferring these small segments of data sequentially. The gapless mode SVC1 parameter block can only be used for gapless I/O operations.

2.3.1 Gapless Mode Supervisor Call 1 (SVC1) Parameter Block Format

The gapless mode SVC1 parameter block must be 24 bytes long, fullword boundary-aligned and located in a task-writable segment. Location within a task-writable segment is necessary so that the status of an I/O request can be returned to the status fields of the SVC1 parameter block. Figure 2-5 presents the gapless mode SVC1 parameter block and a coding example.




```
          SVC   1,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'function code'
          DB    X'lu'
          DS    2 bytes for status
          DC    A (OUT-QUEUE buffer start address)
          DC    A (IN-QUEUE buffer start address)
          DS    4 bytes for buffer length
          DS    4 bytes for length of last buffer
          DC    Y'extended options'
```

Figure 2-5 SVC1 Gapless Mode Parameter Block Format and Coding

Fields:

Function code — is a 1-byte field indicating that the request is a data transfer request. This field also specifies the operation to be performed (read or write) and the extended options pointer. Bit settings for this field are presented in Table 2-8.

lu	is a 1-byte field containing the logical unit currently assigned to the device where the I/O request is directed.
Device-independent status	is a 1-byte field receiving the execution status of an I/O request after completion. The status received is not directly related to the type of device used. Table 2-3 presents device-independent status codes.
Device-dependent status	is a 1-byte field receiving the execution status of a gapless I/O request after completion. The status received contains information unique to the type of device used. Table 2-11 presents device-dependent status codes for gapless operation.
OUT-QUEUE	is a 4-byte field containing the fullword address of a queue where the driver places the starting address of each buffer used in a gapless I/O operation. If the operation is a gapless write, these buffers have been successfully written to tape. If the operation is a gapless read, these buffers contain data read from the tape.
IN-QUEUE	is a 4-byte field containing the fullword address of a queue where the task places the starting address of each buffer to be used in a gapless I/O operation. If the operation is a gapless write, these buffers are written to tape. If the operation is a gapless read, these buffers are filled with data read off from a tape.
Buffer length	is a 4-byte field containing the length of each buffer whose starting address is present on the IN-QUEUE. Buffer length must be an even number of bytes for both read and write operations. All buffers, except the last, must be the same length within a single gapless I/O operation. The amount of space used in the last buffer, however, can vary.
Length of last buffer	is a 4-byte field whose contents depend upon the operation (read or write) being performed. If the operation is a gapless read, the driver fills this field with the length of the last buffer read off tape. The length of the last buffer can be optionally supplied by the task. If the operation is a gapless write, the task supplies the driver with the length of the last buffer to be written.

Extended options	is a 4-byte field containing one of two possible extended function codes indicating gapless mode I/O. Table 2-12 presents the extended function codes available for gapless mode I/O.

2.3.2 Standard Function Code Format - Gapless Mode

Figure 2-6 shows the standard function code format for a gapless mode data transfer request, and Table 2-10 defines each function code bit setting.

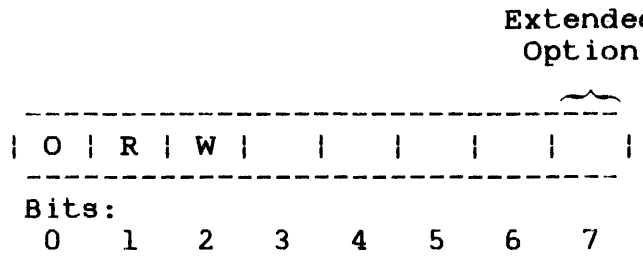


Figure 2-6 Function Code Format for Gapless Mode Data Transfer Requests

TABLE 2-10 FUNCTION CODE BIT POSITIONS FOR GAPLESS MODE DATA TRANSFER REQUESTS

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
0	Function code type	0 = data transfer request. Must be set for gapless I/O operations.
1	Read	1 = read operation. (Bit 2 must be set to 0.)
2	Write	1 = write operation. (Bit 1 must be set to 0.)
3	Not used in gapless mode	

TABLE 2-10 FUNCTION CODE BIT POSITIONS FOR GAPLESS MODE DATA TRANSFER REQUESTS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
4	I/O proceed	0 = if the device is not busy, return control to the calling task after initiation of data transfer to the device. However, if the device is busy, the request is queued and task execution continues. Suggested for gapless mode.
	Wait I/O	1 = stop task execution, initiate data transfer to the device, and wait until the completion of I/O.
	Wait only	1 = task execution stops and waits until the completion of all queued I/O proceed requests to the specified lu. When a wait only request is issued, bit 4 is the only bit set in the function code.
5	Not used in gapless mode	
6	Conditional proceed	0 = after the I/O request is issued, put the task into a wait state if the requested device is busy and the total number of queued requests exceed the maximum. Once the I/O request is completed, the task resumes execution. If the maximum number of queued requests is 1, a pending request causes the task to be placed in a wait state.

TABLE 2-10 FUNCTION CODE BIT POSITIONS FOR GAPLESS MODE DATA TRANSFER REQUESTS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
	Unconditional proceed	1 = any I/O request made to a device that is busy is rejected if the total number of queued requests exceeds the maximum and task execution continues.
	Test I/O complete	1 = test to check for the completion of I/O to a specified lu. If a previous I/O proceed request or queued I/O proceed request does exist, the CC is set to X'F'. However, if there is no outstanding I/O proceed request, the CC is set to X'0'. When a test I/O complete request is issued, bit 6 is the only bit in the function code set. If bit 4 is set, it is ignored.
7	Extended option	1 = test to see if XSVC1 option was specified at Link time. If set, the extended options fullword in the parameter block is checked for specified gapless option. Both the XSVC1 option and this bit must be set for gapless operation.

2.3.3 Logical Unit (lu)

An lu is a decimal number ranging from 0 to 254. The highest lu number to which a task can be assigned is determined by the Link OPTION command. After loading the task into memory, the lu must be assigned to a tape drive which supports gapless I/O (device codes 68-70) through SVC7 or an ASSIGN command.

If no actual I/O operation is desired, the lu should be assigned to NULL:, causing a no-op to occur.

2.3.4 Device-Independent Status Codes

Logical units provide device-independent I/O by causing all I/O requests to be made directly to the lu and not to the device. The execution status of a gapless I/O request that is independent of the physical characteristics of the device being used is returned to the device-independent status field of the parameter block. See Table 2-3. The data remaining in this field from a previous I/O request is not modified until a subsequent I/O is completed or an error occurs.

2.3.5 Device-Dependent Status Codes

The device-dependent status field, together with the device-independent status field, indicates the execution status of a gapless I/O request that is directly related to the unique characteristics of the device being used. Tables 2-11 and 2-12 present the error status codes for gapless operation. A gapless operation ceases upon detection of any one of these errors.

TABLE 2-11 MAGNETIC TAPE DEVICE-DEPENDENT STATUS CODES (GAPLESS ONLY)

STATUS CODE	MEANING
X'8485'	A read or write time-out condition occurred.
X'8487'	The end address read/written by the SELCH does not match the expected end address.
X'8489'	End address returned from SELCH is greater than the expected end address on gapless read.
X'C081'	No buffer is available on the task IN-QUEUE.
X'C082'	Address provided by the user on the IN-QUEUE is outside user's address space.
X'C083'	Address of a queue is not on a fullword boundary.
X'C084'	Length of buffer is an odd number of bytes. Length of last buffer is an odd number of bytes for a write operation.

2.3.6 Buffer Queues

The OUT-QUEUE field and IN-QUEUE field are each 4-byte fields that contain the address of a queue, where:

- The driver places the starting address of each buffer used in a gapless operation (OUT-QUEUE).
- The task places the starting address of each buffer to be used in a gapless operation (IN-QUEUE).

The address of the IN-QUEUE must be greater than the address of the OUT-QUEUE or the SVC1 handler rejects the operation. Figure 2-7 presents the format of both the OUT-QUEUE and IN-QUEUE.

The user sets up a queue via the DLIST xx command, where xx is the total number of buffer entries allowed. See the Common Assembly Language/32 (CAL/32) Reference Manual for instructions.

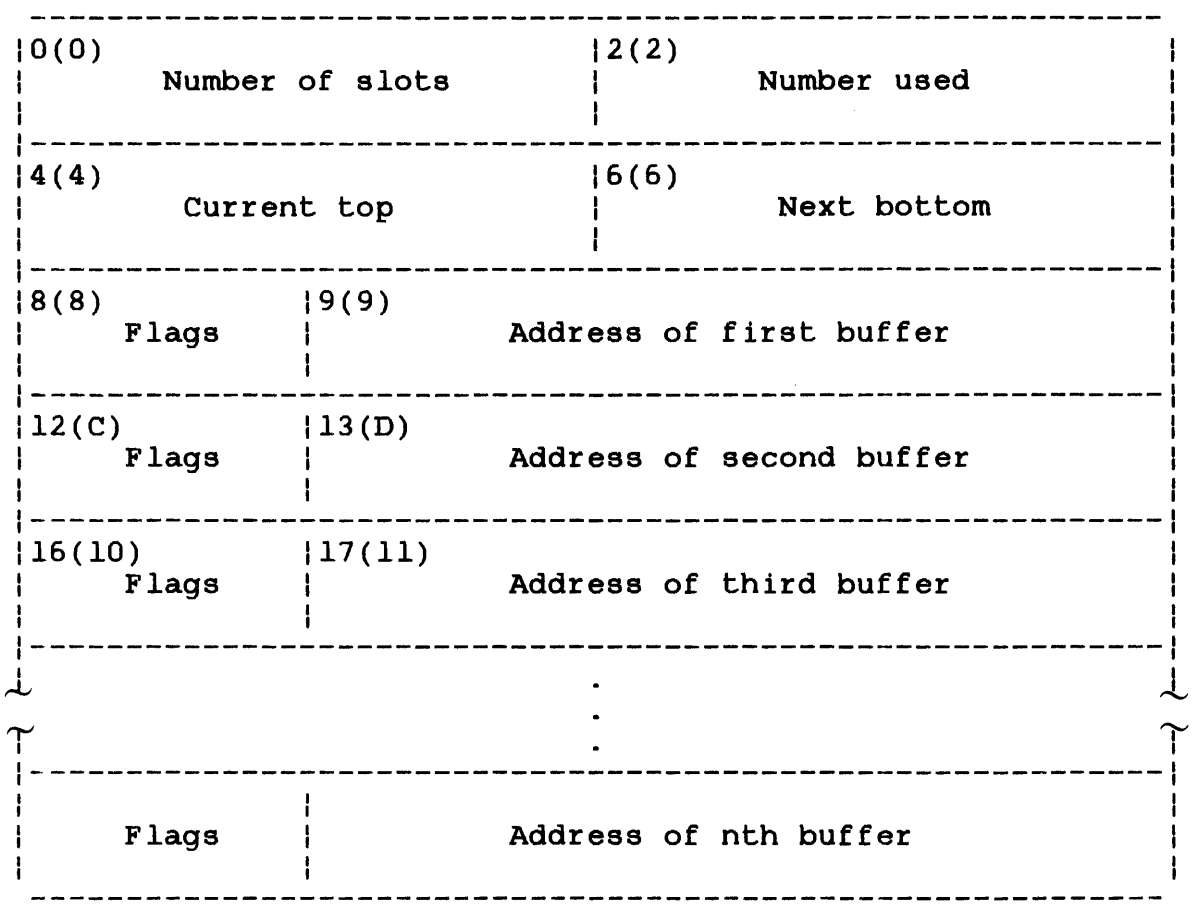


Figure 2-7 IN-QUEUE or OUT-QUEUE Structure

Fields:

Number of slots	is a standard list parameter that is explained in the Common Assembly Language/32 (CAL/32) Reference Manual.
Number used	is a standard list parameter that is explained in the Common Assembly Language/32 (CAL/32) Reference Manual.
Current top	is a standard list parameter that is explained in the Common Assembly Language/32 (CAL/32) Reference Manual.
Next bottom	is a standard list parameter that is explained in the Common Assembly Language/32 (CAL/32) Reference Manual.
Flags	is a 1-byte field. The setting of bit 0 in this field identifies whether the buffer is the last buffer in the list. If bit 0 is set to 0, the buffer is not the last buffer. If bit 0 is set to 1, the buffer is the last buffer in the queue. Under abnormal conditions, the last buffer on the OUT-QUEUE may not have the flag bit set.

NOTE

To properly terminate a gapless write operation, the flags field for the address of the last buffer to be written should have bit 0 set to 1. But a gapless read operation can be terminated in two ways. If the user wishes to read only part of a record or the user knows how long the record is, the flags field for the address of the last buffer read should have bit 0 set to 1. If the user wishes to read the entire record but does not know how long it is, the flags field for the address of all buffers should have bit 0 set to 0. In this case, it is mandatory for the user to retain buffers on the IN-QUEUE until the I/O proceed has been completed. If exactly the number of buffers needed is placed on the IN-QUEUE, the last buffer must be so indicated.

Address of nth buffer	is a 3-byte field containing the hexadecimal starting address of a buffer.

2.3.6.1 Using the Buffer Queue

Gapless operations should be specified as I/O proceed completion operations; therefore, task execution can continue during gapless I/O. One of the functions a task can perform during gapless I/O is to prevent the task from running out of buffer space. The task can accomplish this by removing buffer entries from the OUT-QUEUE and placing them on the IN-QUEUE after a buffer transfer is completed. For example, if a task is required to write 440kb in gapless mode using only five 64kb buffers, the total buffer space available is 320kb (or 120kb less than is required to complete the write operation). After the first buffer has been written, the starting address of the buffer is placed on the OUT-QUEUE. While the second buffer is being written, the task can transfer the address of the first buffer from the OUT-QUEUE to the IN-QUEUE. This gives the task 64kb more buffer space.

Similarly, the task can transfer the address of the second buffer to the IN-QUEUE while the third buffer is being read. This transfer provides the task with enough buffer space for the remaining 56kb. Note that when the task transfers the address of the second buffer from the OUT-QUEUE to the IN-QUEUE, bit 0 of the flags field should be set to 1. The length of the last buffer should be placed in the length of last buffer field of the SVC1 parameter block prior to the start of the operation.

The task should use an add to the bottom of the list (ABL) instruction to add buffer entries to the IN-QUEUE and a run-time library (RTL) instruction to remove buffer entries from the OUT-QUEUE. See the Common Assembly Language/32 (CAL/32) Reference Manual for more information on how to use the ABL and RTL instructions.

2.3.6.2 Trap-Causing Events Resulting from Gapless Input/Output (I/O) Operations

Because a gapless I/O operation should be specified as an I/O proceed completion operation, the task can be notified that a gapless read or write has been completed via a task queue trap. If the SVC1 extended function code 3 (gapless I/O with buffer transfer reporting) has been specified, the task can also receive a task queue trap each time a buffer address has been added to the OUT-QUEUE.

Before a task can be notified of gapless I/O completion or a buffer transfer, the task has to be prepared to receive and handle a task queue handle trap. See the OS/32 Application Level Programmer Reference Manual for information on preparing a task to handle traps.

2.3.7 Buffer Length

The buffer length field is given to the driver by the task to inform the driver of the length of the buffers whose starting addresses are on the IN-QUEUE. Buffer length must equal an even number of bytes for both read and write operations. All buffers must be of the same length with the possible exception of the last buffer (see Section 2.3.8).

2.3.8 Length of Last Buffer

The use of this field is dependent upon the gapless I/O operation being performed (read or write). The length of this buffer cannot be greater than that of the other buffers. If a gapless write operation is being performed, this field is given to the driver by the task and contains the length of the last buffer to be written. This information must be given even if the last buffer is the same length as the previous buffers and should be placed in the SVC1 parameter block before the write is started.

On a gapless read operation, the driver fills this field with the length of the last buffer read from the tape. For example, if a 150kb record is to be read gapless from a tape and 64kb buffers are used, a total of three buffers is required. The first two buffers contains 128kb of information; however, the third buffer contains only 22kb of information. The value 22kb is returned to the length of last buffer field in this example. If desired, this field can be given to the driver by the task. If the last buffer is specified for a read (i.e., the flags field of the address has bit 0 set to 1), this field must be given to the driver by the task.

NOTE

If a gapless read does not reach normal completion (status code 0), the contents of the length of last buffer field are meaningless.

On a gapless write operation, the length of the last buffer must be an even number of bytes.

2.3.9 Extended Options Field

The extended options field in a gapless mode SVC1 parameter block functions as detailed previously in Section 2.2.7. However, only two extended function codes are recognized as valid in a gapless mode SVC1. These codes are presented in Table 2-12.

TABLE 2-12 EXTENDED FUNCTION CODES FOR GAPLESS I/O OPERATION

EXTENDED FUNCTION CODE	OPERATION
2	Gapless operation
3	Gapless operation with buffer transfer reporting

Codes 0 through 1 and 4 through 31 are not used with the gapless mode SVC1 parameter block.

2.4 SERIES 3200 INPUT/OUTPUT (I/O) BUS SWITCH

The Perkin-Elmer Series 3200 I/O Bus Switch Driver (device code 143) provides software control of the I/O bus switch hardware. This switch allows the sharing of I/O devices by two or more Series 3200 Processors equipped with a multiplexor (MUX) and/or a SELCH bus. It may also be used as a bus extender. The bus switch hardware must be strapped for programmable mode. Strapping options are available for normal request, master request or multiple master request contention modes.

2.4.1 Normal Request Contention Mode

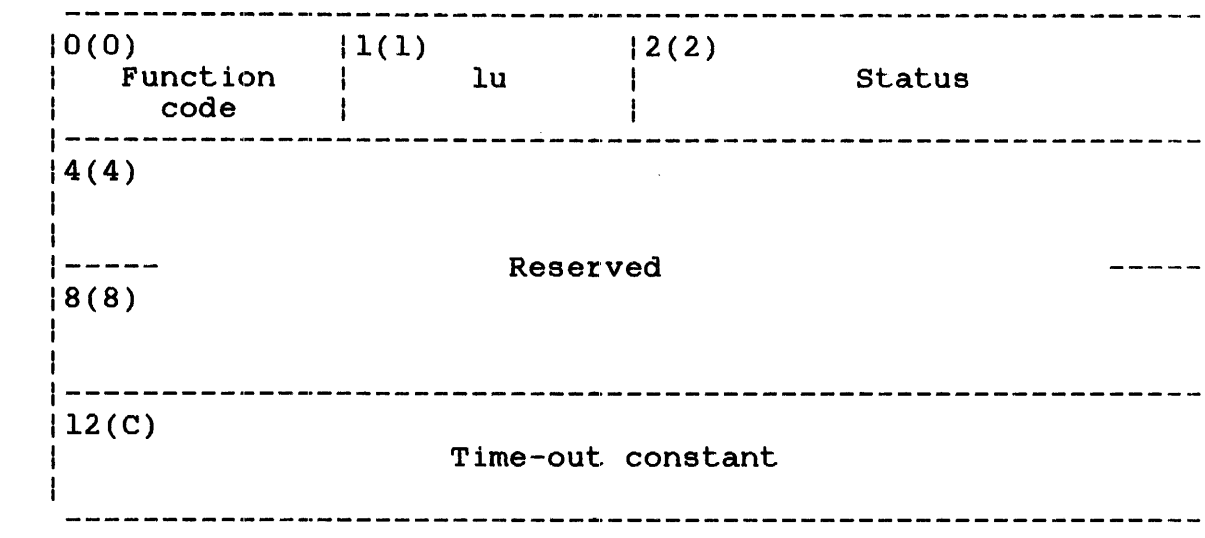
In the normal request contention mode, any central processing unit (CPU) in the configuration can issue a request for the services of the common bus. If the common bus is idle, control is immediately granted to the requesting CPU. If the common bus is in use (controlled by another CPU in the configuration) the request is queued until the controlling CPU relinquishes the bus.

2.4.2 Master Request Contention Mode

In the master request contention mode, one CPU may be designated as the master CPU. When this CPU issues a master request, it is immediately granted control regardless of the state of the bus.

2.4.3 Multiple Master Request Contention Mode

In the multiple master request mode, several or all of the CPUs in the configuration may issue a master request.




```
          SVC   1,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'function code'
          DB    X'lu'
          DS    2 bytes for status
          DCF   Y'00',Y'00'
          DCF   X'Time-out constant'
```

Figure 2-8 SVC1 Parameter Block and Coding for Control of I/O Bus Switch

Fields:

Function code	is a 1-byte field indicating the switch operation to be performed. Hexadecimal function codes for bus switch operations are described in Table 2-13.
lu	is a 1-byte field containing the logical unit to which the bus switch is currently assigned.
Status	is a 2-byte field. The first byte receives the execution status of the switching request. The second receives the hardware status of the switch. Table 2-14 describes the hexadecimal constants returned to this field.

Reserved	is an 8-byte field that must contain zeros.
Time-out constant	is a hexadecimal value ranging from X'1' to X'7FFE' specifying the time-out delay in seconds. The driver waits the indicated number of seconds for connection before time-out. This field is required only for MASTER CONNECT, CONNECT and CLEAR requests.

Table 2-13 defines the function codes for I/O bus switch command function requests.

TABLE 2-13 FUNCTION CODES FOR THE I/O BUS SWITCH DRIVER

FUNCTION CODE	MEANING
X'4x'	MASTER CONNECT - If the switch is strapped for the master or multiple master options, the processor issuing this command is granted control of the common bus, provided another processor does not control the bus via a MASTER CONNECT. Any active normal connection is immediately disconnected and all queued normal CONNECT requests are cleared. The value of x determines whether the call is a WAIT, PROCEED or UNCONDITIONAL PROCEED. For possible values of x, see Table 2-1.
X'3x'	CONNECT - is the normal request/contention sequence. If the common bus is idle, the processor issuing this command is immediately granted control. If the bus is busy, the CONNECT request is queued. The value of the random field of the SVC1 parameter block is used to specify the number of seconds the driver is to wait for connections before time-out. The default is three seconds. The value of x determines whether the call is WAIT, PROCEED or UNCONDITIONAL PROCEED. For possible values of x, see Table 2-1.
X'2x'	CLEAR - activates the common bus system clear (SCLR0). This command causes all interfaces on the common bus to be reset. The value of x determines whether the call is WAIT, PROCEED or UNCONDITIONAL PROCEED. For possible values of x, see Table 2-1.
X'C0'	ENABLE - This command enables interrupts on the common bus.

TABLE 2-13 FUNCTION CODES FOR THE I/O BUS SWITCH DRIVER (Continued)

FUNCTION CODE	MEANING
X'A0'	DISABLE - This command prevents interrupts on the common bus. Interrupts are queued, but not serviced while this command is in effect.
X'84'	DISARM - prevents the queuing and servicing of interrupts on the common bus.
X'88'	DISCONNECT - disconnects the common bus from the controlling processors. Once disconnected, the common bus is available to all processors.
X'90'	RETURN STATUS - returns the hardware status of the bus switch to the second byte of the device-dependent status halfword of the SVC1 parameter block. The only operation performed by this command is a sense status of the switch hardware. The state of the switch is not altered.

TABLE 2-14 I/O BUS SWITCH STATUS CODES

STATUS CODE	MEANING
X'00nn'	Normal completion of requested operation
X'C0nn'	Illegal function code
X'A0nn'	Common bus unavailable
X'84nn'	Hardware failure, bad status returned from connect or CLEAR
X'82nn'	Time-out or connect requested with bus connected, or clear requested with bus not connected

NOTE

'nn' always indicates the switch hardware status. The following list presents some possible hardware status values. See the Input/Output Switch (IOS) Installation and Maintainence Manual for more information.

- 00 - indicates that the switch is selected by a normal or master request.
- 01 - indicates that the switch is unavailable due to power loss on the common bus or disconnected cables.
- 02 - indicates that the bus is busy during a CLEAR interval (100-200ms).
- 08 - indicates that the switch is idle.
- 0A - indicates that the bus is busy servicing another processor.

2.4.4 Programming Considerations

After acquiring the common bus via a MASTER CONNECT, the acquiring processor should immediately issue a CLEAR. This is necessary because a MASTER CONNECT clears any active normal connects, thereby leaving the state of interfaces unknown. The CLEAR command causes the common bus to be initialized (same as depressing the processor INIT button), which places interfaces in a known state.

In a multiple master configuration, a processor acquiring the bus via MASTER CONNECT should immediately relinquish the bus and reacquire it via a NORMAL CONNECT. This will allow any other master processors to acquire the bus via MASTER CONNECT.

After issuing a NORMAL CONNECT or a MASTER CONNECT, the calling procedure should ENABLE interrupts. It is the user's responsibility to know what devices were hung on the common bus at sysgen. If a common bus is not connected with interrupts enabled, a driver call to any device on the common bus results in a device unavailable status return on a time-out from the requested device's driver.

The parameter block fields used for switching operations are the function code, status halfword and, optionally, the random address field (used to specify wait time).

The following examples illustrate the ways in which the I/O bus switch may be used.

Examples:

This example shows the inclusion of the switch in the Sysgen/32 DEVICES statement.

```
DEVICES
   .
   .
   .
   IOS1:,32,143
   .
   .
   .
ENDD
```

The I/O bus switch may then be assigned in the normal manner (i.e, via SVC7 or an operating system multi-terminal monitor (OS/MTM) command).

```
LOAD DMO,SWCHDEMO        *any task that uses switch
TASK DMO                 *not necessary from MTM
.
.
.
AS 7,IOS1:               *switch
.
.
.
START
```

The switch may then be controlled via standard SVC1 function codes, as illustrated by the following simplistic CAL routines.

```
BEGIN    EQU   *
         .
         .
         .
SWCHLU   EQU   7                  SWITCH ASSIGNED TO LU 7
DELAY    EQU   5                  WAIT 5 SEC BEFORE TIME OUT
NRML     EQU   X'38'              NORMAL CONNECT W/WAIT FC
ENAB     EQU   X'C0'              ENABLE FC
DISCON   EQU   X'88'              DISCONNECT FC
         .
```

```
            .
            .
            BAL   15,SWCHPREP          GO GET BUS
            .
            .
            .
            BAL   15,RLSEBUS           RELEASE BUS
            .
            .
            .
            SVC   3,EOTCODE
            .
            .
            .
SWCHPREP    EQU   *
            STM   12,SAVEREGS          SAVE REGISTERS
            LHI   12,SWCHLU            PICK UP LU #
            LHI   13,DELAY             PICK UP DELAY
            LHI   14,NRML              PICK UP FC
            LHI   15,ENAB              PICK UP FC
GETBUS      EQU   *
            STB   12,SVC1.FC           LU # TO PARBLK
            STB   13,SVC1.RAD          DELAY TO PARBLK
            STB   14,SVC1.FC           CONNECT W/WAIT TO PARBLK
            SVC   1,SVC1.FC            ISSUE CONNECT REQUEST
            LHI   14,SVC1.STA          PICK UP STATUS
            CLHI  14,X'00'             CHECK FOR SUCCESSFUL CONNECT
            BNE   CONNERR              GO TO ERR ROUTINE
            STB   15,ENAB              ENABLE FC TO PARBLK
            SVC   1,SVC1.FC            ISSUE ENABLE INTERRUPTS REQUEST
            LHI   15,SVC1.STA          PICK UP STATUS
            CLHI  15,X'00'             CHECK FOR SUCCESSFUL ENABLE
            BNE   ENABERR              GO TO ERR ROUTINE
            LM    12,SAVEREGS          RESTORE REGISTERS
            BR    15                   BUS READY - GO DO I/O
            .
            .
            .
RLSEBUS     EQU   *
            ST    12,SAVEREGS          SAVE REGISTERS
            LHI   14,DISCON            PICK UP DISCONNECT FC
            LHI   12,SWCHLU            PICK UP LU #
            STB   14,SVC1.FC           DISCONNECT TO PARBLK
            STB   12,SVC1.LU           LU # TO PARBLK
            SVC   1,SVC1.FC            REQUEST DISCONNECT
            LHI   14,SVC1.STA          PICK UP STATUS
            CLHI  14,X'00FF'           CHECK STAT - HARDWARE N/A
            BG    DSCNERR              GO TO ERR ROUTINE
            LM    12,SAVEREGS          RESTORE REGISTERS
            BR    15                   DISCONNECTED - DO WHATEVER
            .
            .
            .
CONNERR     EQU   *
            .
            .
```

```
          .
ENABERR   EQU   *
          .
          .
          .
DSCNERR   EQU   *
          .
          .
          .
*
* DEFINITIONS
*
          .
          .
          .
SAVEREGS  DSF   9                    REGISTER SAVE AREA
EOTCODE   DS    2                    HALFWORD FOR EOT
          $SVC1                      PICK UP SVC1 STRUC
          .
          .
          .
          END   BEGIN
```

CHAPTER 3
GENERAL SERVICE FUNCTIONS SUPERVISOR CALL 2 (SVC2)

3.1 INTRODUCTION

SVC2 provides general service functions distinguished from one another by a specific function code number. Each SVC2 function requires a specific parameter block for proper operation. Refer to each individual code for its parameter block format and required coding. Table 3-1 lists all available SVC2 function codes with a brief description of each.

TABLE 3-1 SVC2 FUNCTION CODES

SVC2 CODE	NAME	FUNCTION
SVC2 code 0	Make journal entries	Makes an entry into the system journal from an executive task (e-task).
SVC2 code 1	Pause	Places the task in a suspended state.
SVC2 code 2	Get storage	Reserves a workspace area for external subroutines called by the task during execution.
SVC2 code 3	Release storage	Releases the temporary storage locations obtained by a previous SVC2 code 2. Gets storage by decreasing the task UTOP by the number of user-specified bytes.
SVC2 code 4	Set status	Modifies the arithmetic fault interrupt bit and condition code (CC) in the program status word (PSW).

TABLE 3-1 SVC 2 FUNCTION CODES (Continued)

SVC2 CODE	NAME	FUNCTION
SVC2 code 5	Fetch pointer	Copies the address of UTOP, CTOP and UBOT from the task control block (TCB) and stores them in the task user-dedicated location (UDL).
SVC2 code 6	Convert binary to ASCII hexadecimal or ASCII decimal	Converts a binary number to either an ASCII hexadecimal or ASCII decimal number.
SVC2 code 7	Log message	Sends a message to the appropriate log device regardless of the current logical unit (lu) assignments.
SVC2 code 8	Interrogate clock	Sends the user the current time of day calculated in seconds from midnight in binary or in formatted ASCII.
SVC2 code 9	Fetch date	Sends the user the current date in formatted ASCII.
SVC2 code 10	Time of day wait	Places the calling task in a wait state until a specific time of day.
SVC2 code 11	Interval wait	Places the calling task in a wait state for an interval, which is specified in milliseconds from the time the call is executed.
SVC2 code 14	Internal reader	Allows a foreground task loaded from the system console to invoke operator and command substitution system (CSS) commands.
SVC2 code 15	Convert ASCII hexadecimal or ASCII decimal to binary	Converts an ASCII hexadecimal or ASCII decimal number to a binary number.

TABLE 3-1 SVC2 FUNCTION CODES (Continued)

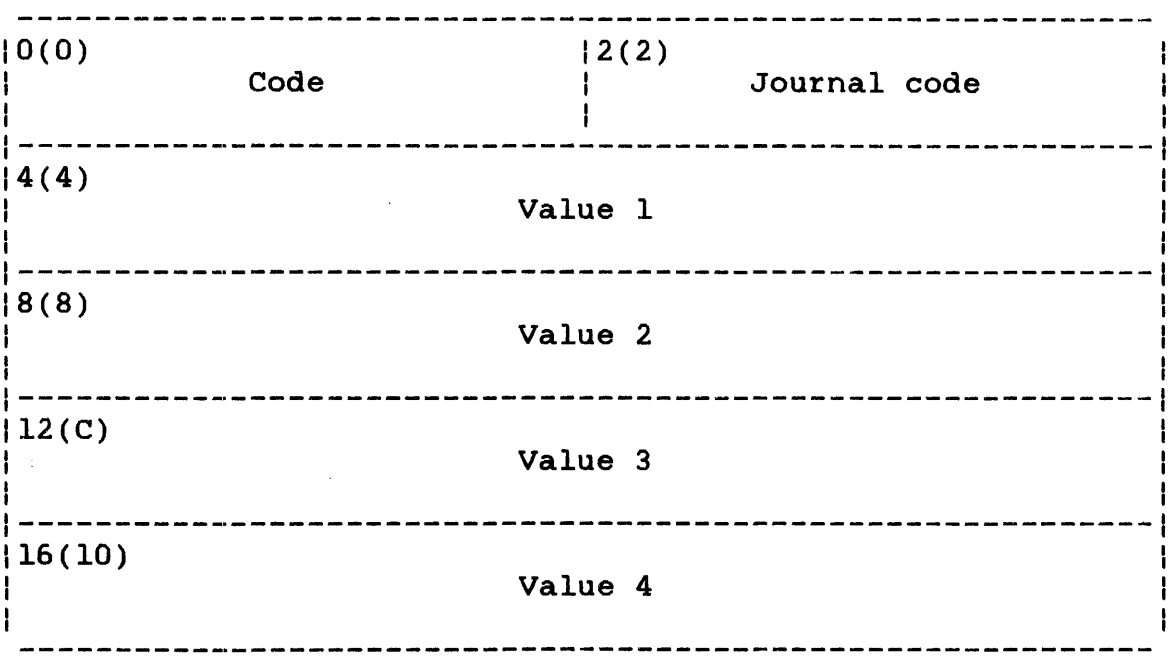
SVC2 CODE	NAME	FUNCTION
SVC2 code 16	Pack file descriptor	Processes a user-specified unpacked file descriptor (fd) into a packed format to be used by the operating system.
SVC2 code 17	Scan mnemonic table	Scans for an ASCII character string in a mnemonic table and compares it with the user-specified ASCII character string for a match.
SVC2 code 18	Move ASCII characters	Moves a specified number of ASCII characters in memory from the sending location to a receiving location.
SVC2 code 19	Peek	Obtains user-related information from operating system data structures.
SVC2 code 20	Expand allocation	Reserved for sequential tasking machines. Provides for compatibility with current 32-bit operating systems.
SVC2 code 21	Contract allocation	Reserved for sequential tasking machines. Provides for compatibility with current 32-bit operating systems.
SVC2 code 23	Timer management	Schedules the addition of a parameter to a task queue on completion of a specified interval or a repetitive interval. Puts a task in a wait state until completion of an interval. Determines the time remaining for a previously established interval to expire. Cancels a previously established interval.

TABLE 3-1 SVC2 FUNCTION CODES (Continued)

SVC2 CODE	NAME	FUNCTION
SVC2 code 24	Set accounting information	Stores eight bytes of user-supplied information in the accounting transaction file (ATF) on task completion or data overflow of accounting records.
SVC2 code 25	Fetch accounting information	Fetches accounting information and stores it in a user-specified receiving area.
SVC2 code 26	Fetch device name	Searches the volume mnemonic table (VMT) for a user-supplied volume name and returns the name of the device on which that volume is mounted.
SVC2 code 27	Memory management	Allows a user task (u-task) to access and modify entries (except shared ones) within the private segment table (PST) in its TCB.
SVC2 code 29	Unpack file descriptor	Converts a packed fd from the file directory or an SVC7 paramter block to its unpacked format.

3.2 SVC2 CODE 0: MAKE JOURNAL ENTRIES

SVC2 code 0 makes an entry into the system journal from an e-task. The system journal provides a method to trace back important events (SVCs, input/output (I/O) operations, task switching) that occurred during system operation. For example, the journal is useful for tracing the cause of a system failure. The parameter block format for SVC2 code 0 is shown in Figure 3-1.




```
          SVC   2,parblk
          .
          .
          .
parblk    DC    H'0'
          DC    H'journal code'
          DC    F'value 1'
          DC    F'value 2'
          DC    F'value 3'
          DC    F'value 4'
```

Figure 3-1 SVC2 Code 0 Parameter Block Format and Coding

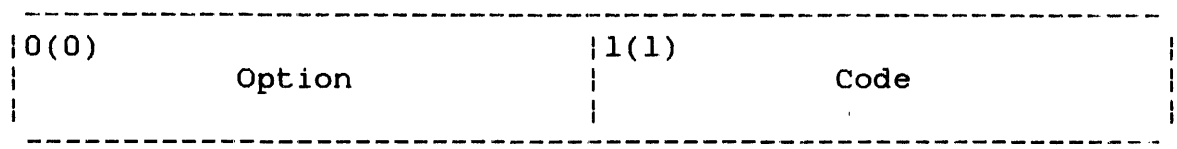
During execution, a logical-OR operation is performed on a mask and the journal code to indicate that the entry originates from an SVC2 code 0, rather than from within the system. The value 1, 2, 3 and 4 fields of the parameter block are stored following the journal code and calling task name in the journal. These values can contain any desired information to be preserved for system debugging.

NOTE

This call has an effect only if the journal is included in the system at (source) system generation (sysgen).

3.3 SVC2 CODE 1: PAUSE

SVC2 code 1 stops task execution and places the task into a suspended state. This is accomplished through the SVC2 code 1 parameter block shown in Figure 3-2.




```
                SVC   2,parblk
                .
                .
                .
                ALIGN 4
parblk          DB    0,1
```

Figure 3-2 SVC2 Code 1 Parameter Block Format and Coding

This parameter block must be two bytes long, fullword boundary-aligned, and does not have to be located within a task-writable segment. Following is a description of each field in the parameter block.

Fields:

Option	is a 1-byte field that must contain a value of 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal value 1 to indicate code 1 of SVC2.

After executing SVC2 code 1, the following message is displayed on the system console:

```
TASK PAUSED
```

If the task is running under MTM, the above message is displayed on the user console.

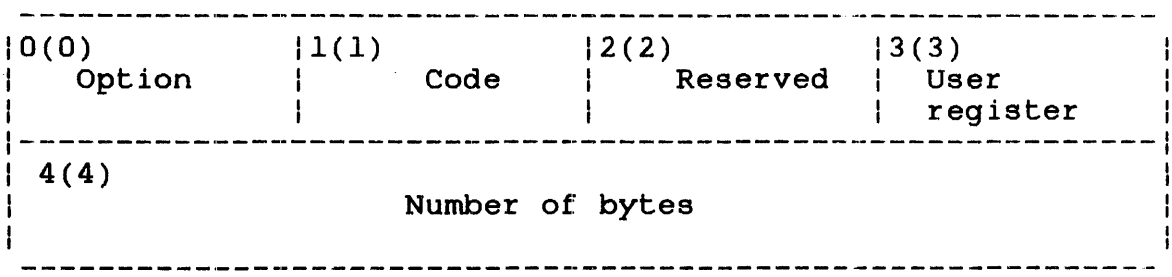
While the task is paused, the operator can issue commands directed to the paused task to change the task environment. To continue task execution, enter the CONTINUE command. Task execution resumes with the instruction immediately following SVC2 code 1.

3.4 SVC2 CODE 2: GET STORAGE

SVC2 code 2 reserves a workspace area for external subroutines called by the task during execution (e.g., FORTRAN run-time library (RTL) routines). This workspace is reserved in the unused portion of the task's impure segment between UTOP and CTOP. For more information on this segment, see the OS/32 Application Level Programmer Reference Manual.

The SVC2 code 2 operation does not increase the task's allocated memory size.

Figure 3-3 illustrates the parameter block for SVC2 code 2.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,2,0
          DB    user register
          DC    F'number of bytes'
```

Figure 3-3 SVC2 Code 2 Parameter Block Format and Coding

This parameter block must be eight bytes long, fullword boundary-aligned, and located in a task-writable segment when option X'80' is used. A general description of each field in the parameter block follows.

Fields:

Option — is a 1-byte field that must contain one of the following options:

- Option X'00' reserves the user-specified number of bytes in fullword increments in the unused portion of the task impure segment between UTOP and CTOP.
- Option X'80' reserves the remaining unused portion of the task impure segment between UTOP and CTOP.

Code — is a 1-byte field that must contain the decimal value 2 to indicate code 2 of SVC2.

Reserved — is a reserved 1-byte field that must contain a 0.

User register — is a 1-byte field that must contain a decimal number ranging from 0 to 15 specifying the register to receive the starting address of the reserved workspace area.

Number of bytes — is a 4-byte field containing different information for each option.

- Option X'00' contains the user-specified number of bytes to be reserved for the workspace area.
- Option X'80' receives the number of bytes actually reserved for the workspace area.

When a task is link-edited, the default task workspace (the difference between CTOP and UTOP) should be large enough to provide enough workspace for both the task and the external subroutines. The task workspace can be increased through the WORK= parameter of the Link OPTION command, the LOAD command or an SVC6.

After executing SVC2 code 2, the CC is set as follows.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	User-specified number of bytes is a negative value or a value greater than the task's allocated memory size

NOTE

When SVC2 code 2 is executed and the task UTOP changes, the UTOP address stored in the task UDL is not updated to contain the most current UTOP. SVC2 code 5 updates the address in the UDL.

3.4.1 SVC2 Code 2, Option X'00'

If option X'00' is specified, the address of the task's current UTOP is adjusted to include the number of user-specified bytes in the parameter block. Once the UTOP address is adjusted, the starting address of the reserved workspace area, which is the original or previous UTOP, is stored in the user-specified register. This option can reserve new workspace areas until they are needed during task execution in subsequent calls.

The number of bytes should be specified in fullword increments because the UTOP address is rounded up to the nearest fullword boundary.

Example:

```
            SVC   2,GET
            .
            .
            .
            ALIGN 4
GET         DB    0,2,0
            DB    5
            DC    Y'600'              1.5K
```

This example is illustrated in Figure 3-4. A task is loaded with a task workspace area of 5.5kb specified in the LOAD command. After the task is loaded, UTOP is located at X'C78' and CTOP is located at X'1FFE'. After executing SVC2 code 2, UTOP is adjusted to X'1278'. The remaining unused portion (area between X'1278' and X'2000') can be used by subsequent routines when needed during task execution.

If the user-specified number of bytes for option X'00' is a negative value or greater than the task current allocated memory size (CTOP):

- The UTOP address is not adjusted
- An address of 0 is returned in the user-specified register
- The CC is set to 4 (V bit set)

038-1

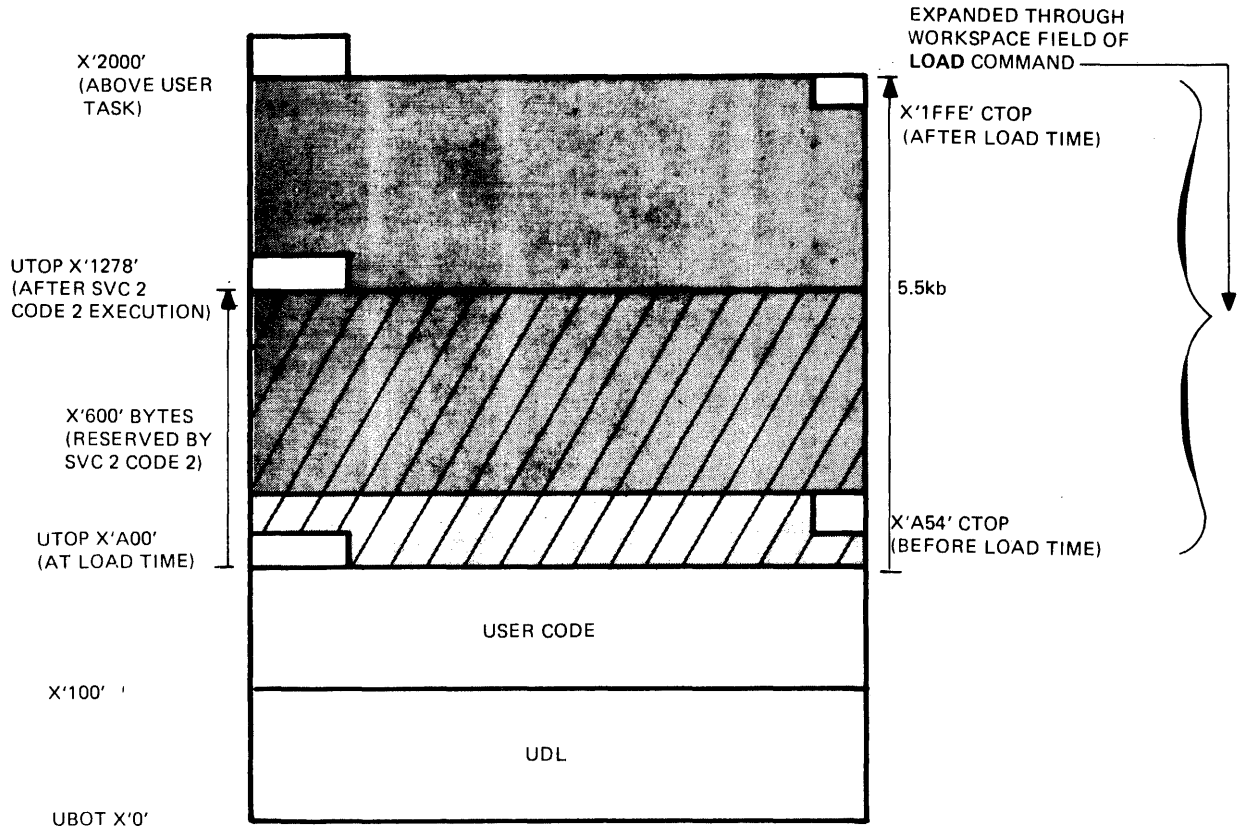


Figure 3-4 Task Impure Segment for SVC2 Code 2, Option X'00'

3.4.2 SVC2 Code 2, Option X'80'

If option X'80' is specified, the parameter block must be located in a writable segment. The address of the task's current UTOP is adjusted to include all of the remaining unused portion in the impure segment, making UTOP equal CTOP+2. Once the UTOP address is adjusted, the starting address of the reserved workspace area, which is the address of the original or previous UTOP, is stored in the user-specified register. Also, the number of bytes actually reserved is stored in the number of bytes field in the parameter block.

Example:

```
            SVC    2,GET
            .
            .
            .
            ALIGN  4
GET         DB     X'80',2,0
            DB     5
            DS     4
```

This example is illustrated in Figure 3-5. A task is linked with a workspace greater than 5.5kb. After the task is loaded with a load expand factor of 5.5kb, UTOP is located at X'C78'. After executing SVC2 code 2, UTOP is adjusted to X'2000'.

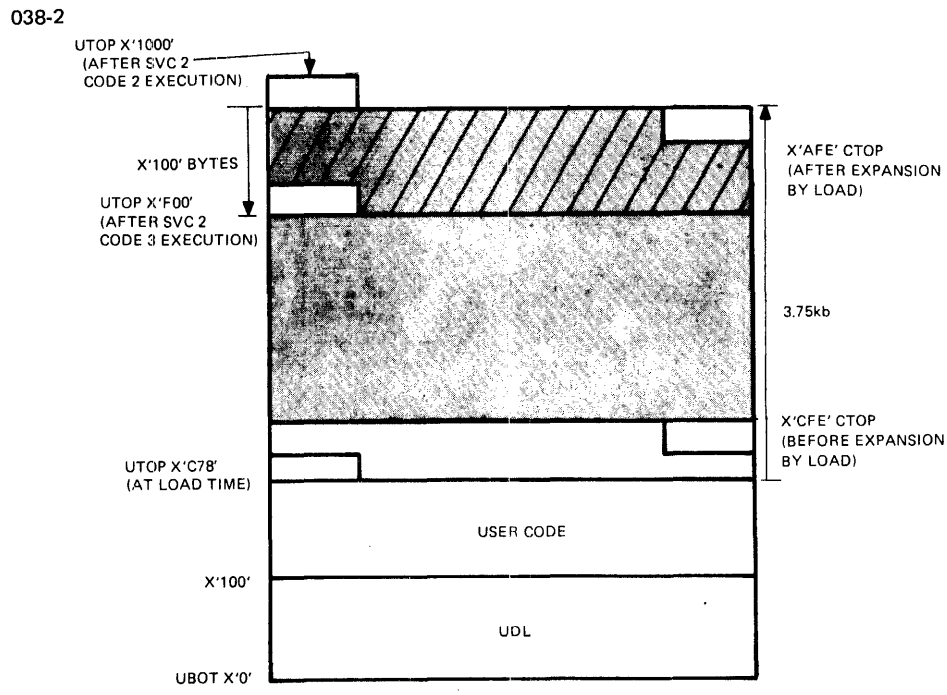
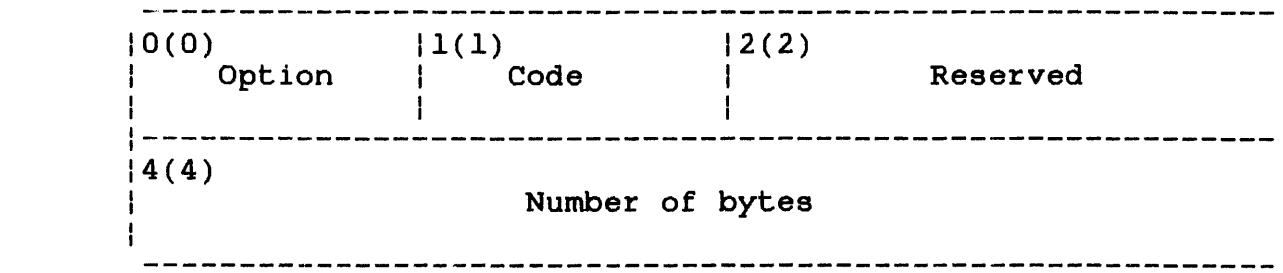


Figure 3-5 Task Impure Segment for SVC2 Code 2, Option X'80'

3.5 SVC2 CODE 3: RELEASE STORAGE

SVC2 code 3 releases the workspace area in the unused portion of of the task impure segment that had been reserved by a previous SVC2 code 2 (see Section 3.3). Releasing the reserved workspace for external subroutines does not decrease the task's allocated memory size. The SVC2 code 3 parameter block is shown in Figure 3-6.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,3
          DC    H'0'
          DC    F'number of bytes'
```

Figure 3-6 SVC2 Code 3 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that contains a value of 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal value 3 to indicate code 3 of SVC2.

Reserved	is a reserved 2-byte field that must contain zeros.
Number of bytes	is a 4-byte field that must contain the user-specified number of bytes of the reserved workspace to be released.

When executing this SVC, the address of the task's current UTOP is adjusted to exclude the user-specified number of bytes of reserved workspace. If the number of bytes is not specified in fullword increments, the UTOP address is adjusted by rounding down to the nearest fullword boundary. After executing SVC2 code 3, the CC is set as follows.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	User-specified number of bytes is a negative value or a value greater than the task's allocated memory size

Example:

```
            SVC    2,RELEASE
            .
            .
            .
            ALIGN  4
RELEASE     DB     0,3
            DC     H'0'
            DC     F'256'
```

Figure 3-7 illustrates this example. A task was linked with a workspace of 3.75kb and loaded into memory. After the task is loaded, UTOP is located at X'C78' and CTOP is located at X'FFE'. After executing SVC2 code 2, UTOP is adjusted to X'1000'. After executing SVC2 code 3, 256 bytes of reserved storage are released, adjusting UTOP to X'F00'.

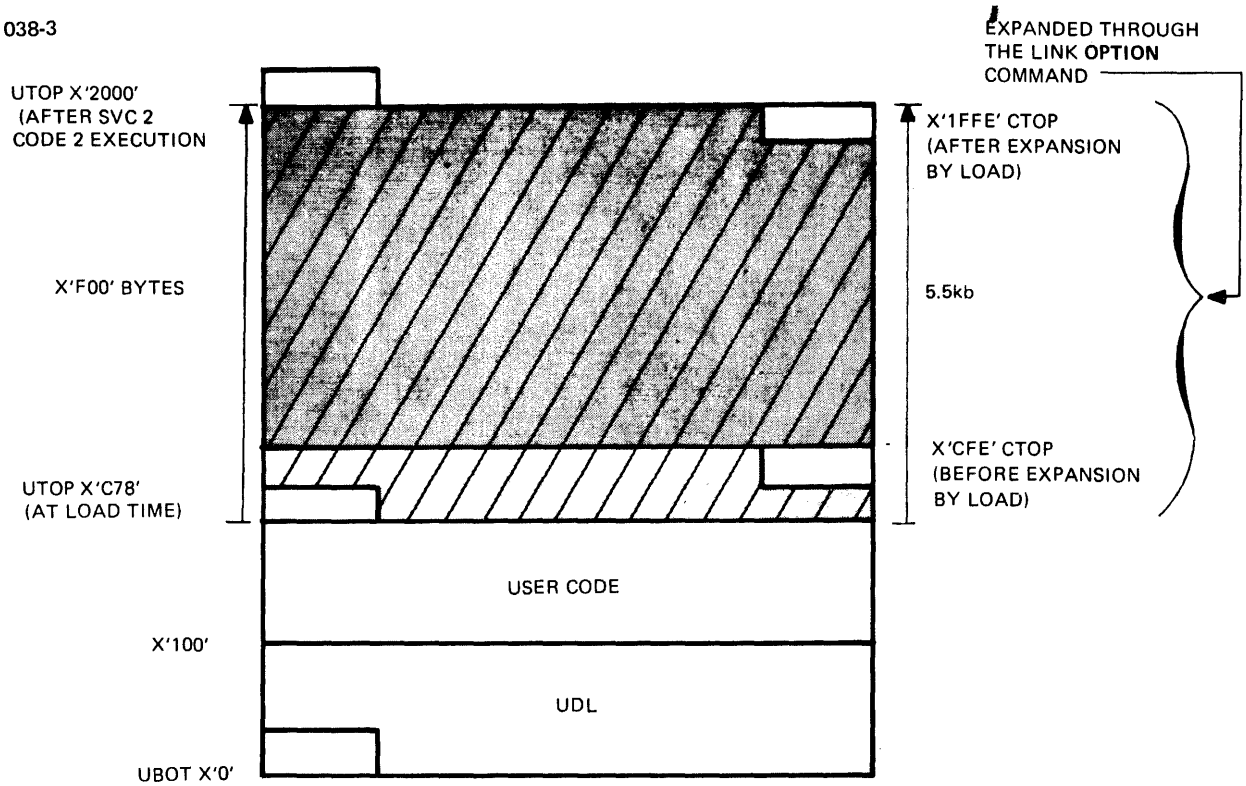


Figure 3-7 Task Impure Segment for SVC2 Code 3

If the user-specified number of bytes is a negative number or is more than the number specified by Link, the UTOP address is not adjusted and the CC is set to 4 (V bit set).

3.6 SVC2 CODE 4: SET STATUS

SVC2 code 4 modifies the arithmetic fault interrupt bit and the CC settings in the PSW. Figure 3-9 shows the PSW and the bits affected by the set status operation. When the arithmetic fault interrupt bit setting is modified, interrupts are enabled or disabled. When the CC setting is modified, the current 4-bit setting is replaced with a new 4-bit setting. This is accomplished through the SVC2 code 4 parameter block shown in Figure 3-8.

0(0) Option	1(1) Code	2(2) Arithmetic fault parameter	3(3) Condition code parameter

```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,4
          DB    arithmetic fault parameter, condition
                code parameter
```

Figure 3-8 SVC2 Code 4 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option — is a 1-byte field that must contain one of the following options:

- Option X'00' modifies the arithmetic fault bit and CC in the PSW.
- Option X'80' modifies only the CC in the PSW (see Figure 3-9).

Code — is a 1-byte field that must contain the decimal value 4 to indicate code 4 of SVC2.

Arithmetic fault parameter — is a 1-byte field that must contain one of the following parameters when option X'00' is specified. For option X'80', this field must contain zeros.

- X'00' disables all arithmetic fault interrupts for Models 7/32 and 8/32 Processors. For Perkin-Elmer Series 3200 Processors, only arithmetic fault interrupts due to floating point underflow are disabled.
- X'10' enables all arithmetic fault interrupts.

Condition code parameter — is a 1-byte field that must contain a hexadecimal value ranging from X'00' to X'0F'.

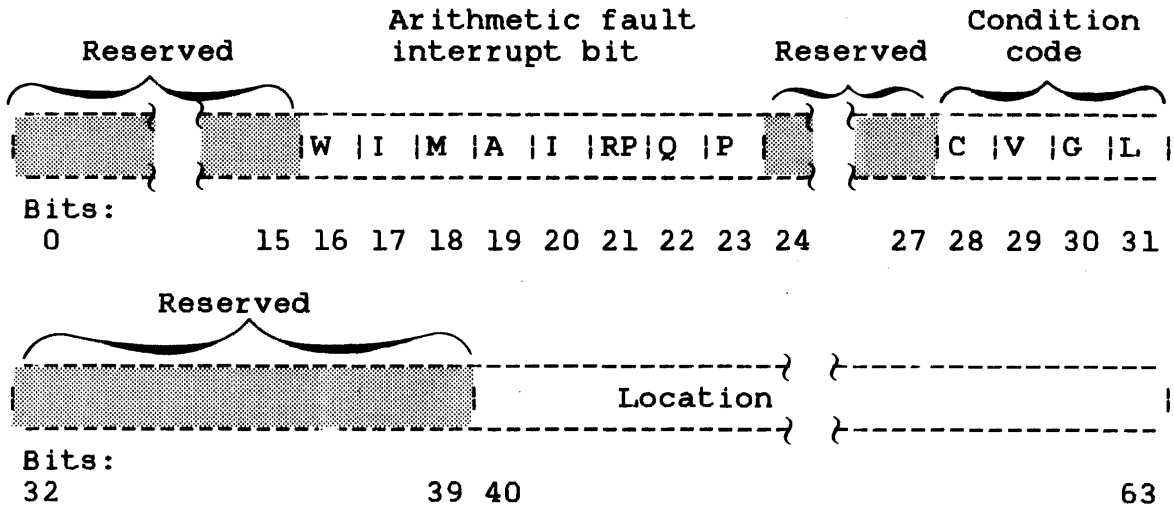


Figure 3-9 Program Status Word (PSW)

An arithmetic fault occurs during an arithmetic operation for any of the following conditions:

- Fixed point quotient overflow
- Fixed point division by 0
- Floating point overflow and underflow
- Floating point division by 0

The CC (bits 28 through 31) is set after executing certain instructions. Each bit in the CC corresponds to a result or condition caused by executing an instruction. The CC settings for arithmetic operations follow.

Condition Code:

C	V	G	L	
1	0	0	0	Arithmetic carry, borrow or shifted carry
0	1	0	0	Arithmetic overflow
0	0	1	0	Greater than 0
0	0	0	1	Less than 0

These four bits have different meanings for logical operations, branching operations and I/O operations. For the definitions of the bit settings for each particular operation, see the appropriate processor manual.

3.6.1 SVC2 Code 4, Option X'00'

If the SVC2 code 4 parameter block contains X'00' in the option field, X'00' in the arithmetic fault field, and a value ranging from X'00' to X'0F' in the CC field, all arithmetic faults are ignored for Models 7/32 and 8/32 Processors. For Series 3200 Processors, only arithmetic faults resulting from floating point underflow are ignored. For more information on Series 3200 arithmetic fault interrupts, see the appropriate Series 3200 Processor Manual. The current CC value in the PSW is replaced with the value specified in the CC field of the parameter block.

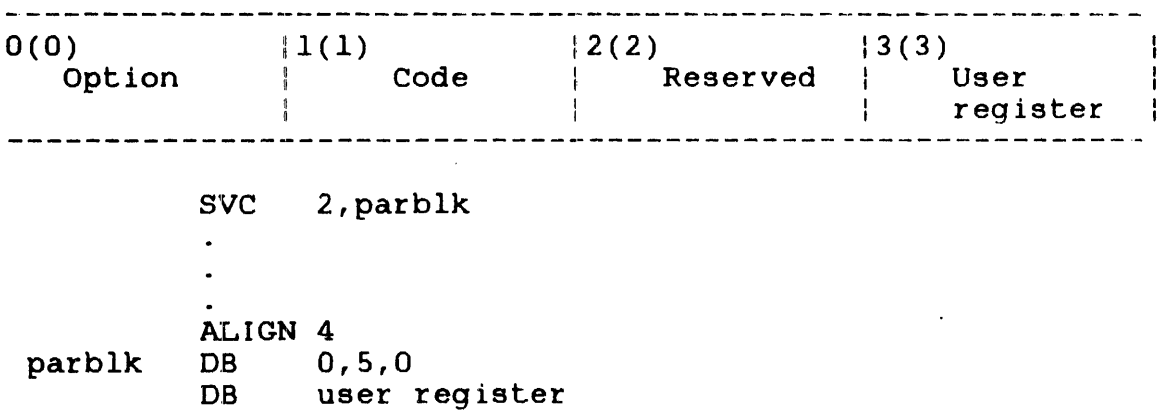
If the SVC2 code 4 parameter block contains X'00' in the option field, X'10' in the arithmetic fault field, and a value ranging from X'00' to X'0F' in the CC field, all arithmetic fault interrupts are enabled. The current CC value in the PSW is replaced with the value specified in the CC field of the parameter block.

3.6.2 SVC2 Code 4, Option X'80'

If option X'80' is specified and the CC parameter field contains a value of X'00' through X'0F', the current CC value of the PSW is replaced with the value specified in the CC field of the parameter block. The arithmetic fault field is ignored.

3.7 SVC2 CODE 5: FETCH POINTER

SVC2 code 5 loads the starting address of a task's UDL into a user-specified register. It then stores the current addresses of UBOT, UTOP and CTOP, located in the TCB, into their corresponding locations in the task UDL. This is accomplished through the SVC2 code 5 parameter block shown in Figure 3-10.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,5,0
          DB    user register
```

Figure 3-10 SVC2 Code 5 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the value 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 5 to indicate code 5 of SVC2.
Reserved	is a reserved 1-byte field that must contain a 0.

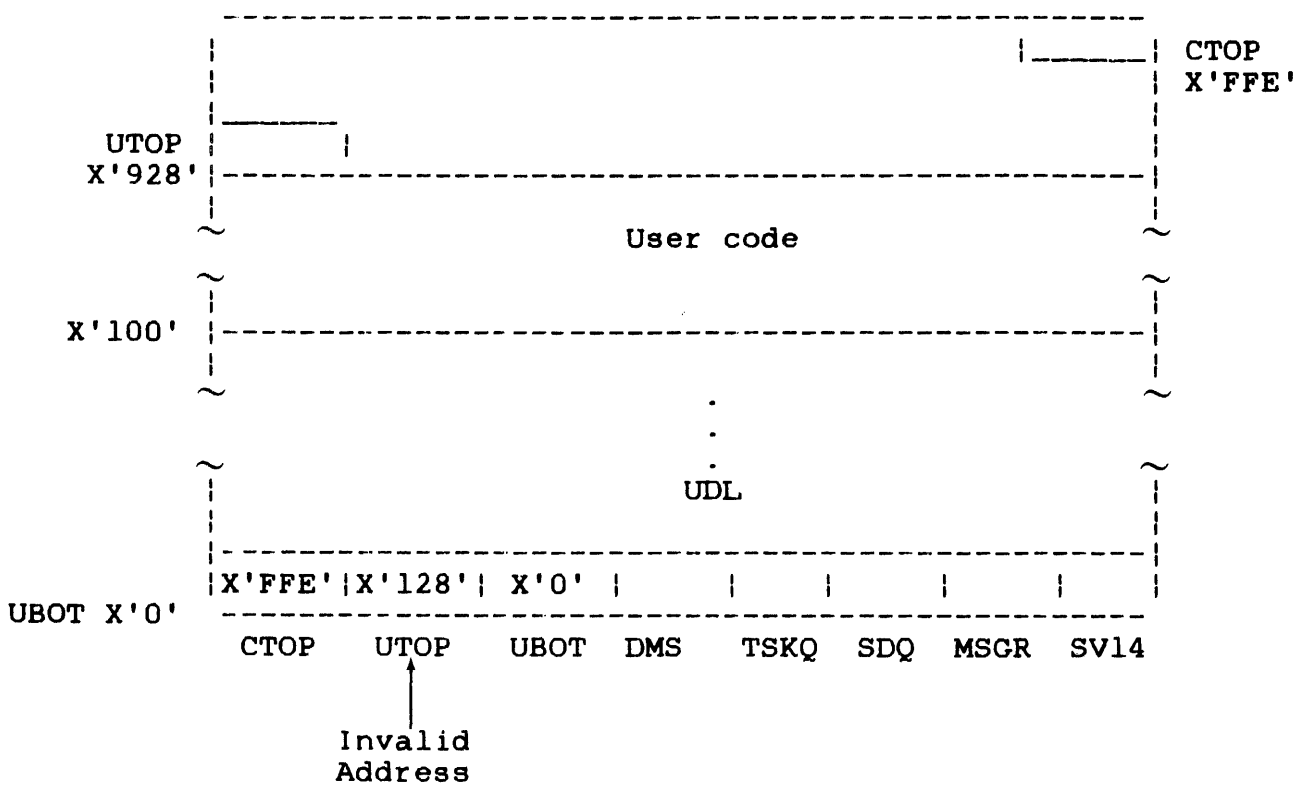
User register	is a 1-byte field that must contain a decimal number from 0 to 15, indicating the register that receives the UDL starting address.

When executing this call, the UDL starting address, which is loaded into the user-specified register, varies for u-tasks and e-tasks. The starting address for a u-task is the relative address, which is always 0. The starting address for an e-task is the absolute address, which depends on the task memory location.

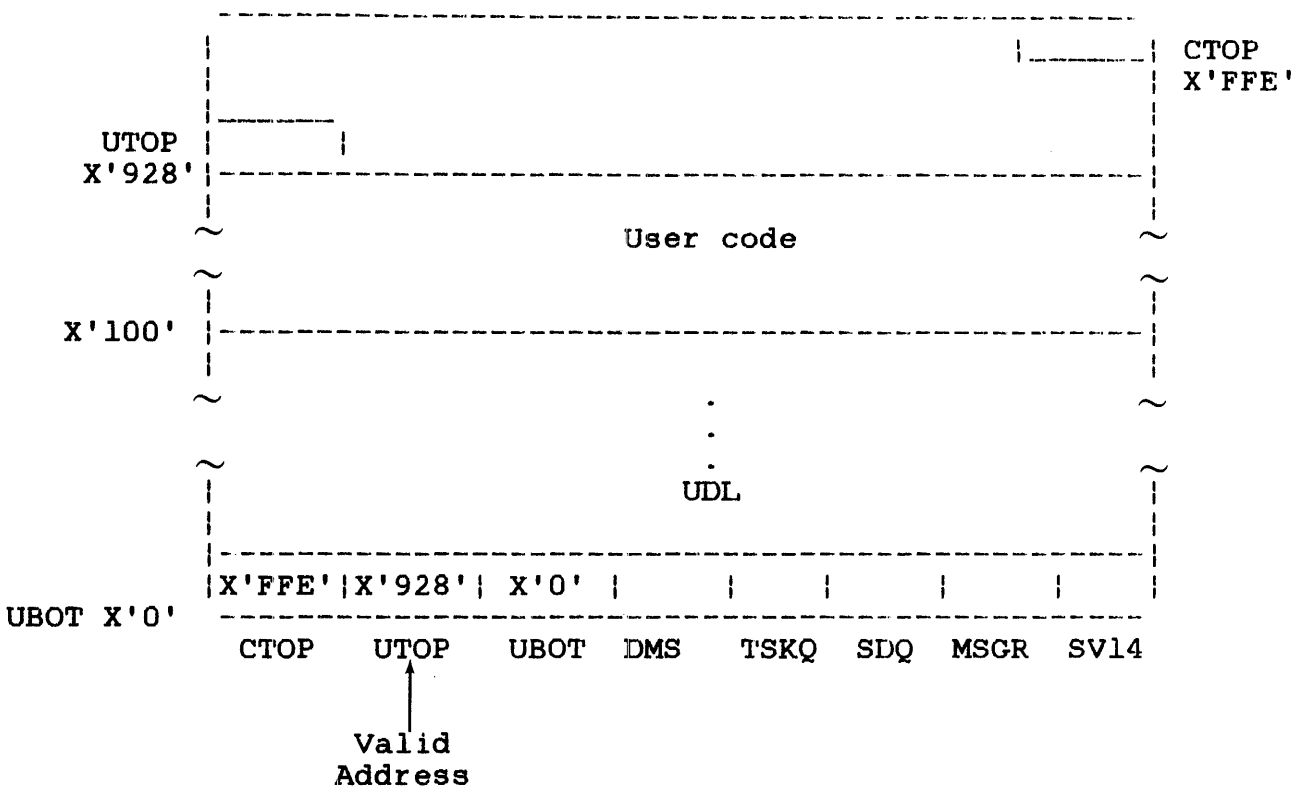
If the user modified the UDL by changing address pointers or if UTOP was changed by a GET or RELEASE STORAGE, the contents of CTOP, UTOP and UBOT in the UDL might not be valid. SVC2 code 5 restores this data to a valid state by storing the current values of CTOP, UTOP and UBOT into the UDL.

Example:

UDL after execution of SVC2 code 2 and before execution of SVC2 code 5:



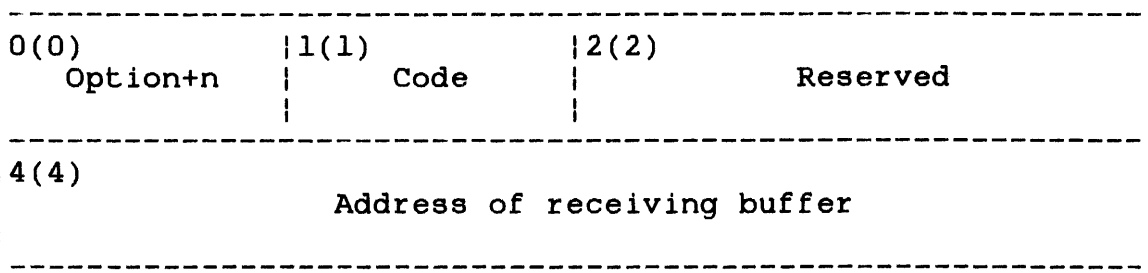
UDL after execution of SVC2 code 5:



For more information on the UDL, see the OS/32 Application Level Programmer Reference Manual.

3.8 SVC2 CODE 6: CONVERT BINARY NUMBER TO ASCII HEXADECIMAL OR ASCII DECIMAL

SVC2 code 6 converts an unsigned 32-bit binary number located in user register 0 to an ASCII hexadecimal number or an ASCII decimal number. This is accomplished through the SVC2 code 6 parameter block shown in Figure 3-11.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option+n,6
          DC    H'0'
          DCF   A(receiving buffer)
```

Figure 3-11 SVC2 Code 6 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and must be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option+n — is a 1-byte field that must contain the sum of one of the following options and n (n specifies a decimal number from 0 to 63 indicating the number of bytes in the buffer specified in the SVC2 code 6 parameter block).

- Option X'00'+n converts a binary number to ASCII hexadecimal.
- Option X'40'+n converts a binary number to ASCII hexadecimal, suppressing leading zeros.
- Option X'80'+n converts a binary number to ASCII decimal.
- Option X'C0'+n converts a binary number to ASCII decimal, supressing leading zeros.

Code — is a 1-byte field that must contain the decimal number 6 to indicate SVC2 code 6.

Reserved — is a reserved 2-byte field that must contain zeros.

Address of receiving buffer — is a 4-byte field that must contain the address of the previously allocated buffer that receives the converted number. This address can be located on any byte boundary.

The receiving buffer should be defined to receive the largest number, which is is 2,147,483,647 (2^{31} -1), that can be converted from register 0. Allocate an 8-byte buffer for binary to ASCII hexadecimal. Allocate a 10-byte buffer for binary to ASCII decimal. If the user's largest number to be converted is less than 2^{31} -1, the receiving buffer can be less than the suggested length of the buffer.

When the user-specified binary number located in register 0 is converted, the result is stored right-justified in the receiving buffer with the left-most significant digits filled with ASCII zeros. However, if the converted number is longer than the buffer, the left-most digits of the converted number are lost. If suppression of leading zeros is requested, the left-most zeros in the receiving buffer are filled with spaces (hexadecimal 20).

3.8.1 SVC2 Code 6, Option X'00'+n

If option X'00'+n is specified, the unsigned 32-bit binary number located in the user register 0 is converted to an ASCII hexadecimal number. The resulting number is stored right-justified in the receiving buffer with the left-most significant digits filled with ASCII zeros (hexadecimal 30).

Example:

```
            LI     0,F'8520'
            .
            .
            .
            SVC    2,CONVERT
            .
            .
            .
            ALIGN  4
CONVERT     DB     X'00'+8,6
            DC     H'0'
            DCF    A(BUF)
BUF         DS     8
```

Register 0 before and after execution of SVC2 code 6:

```
 ----------------
|0 0| 0 0|2 1|4 8| Hex
 ----------------
```

Receiving buffer after execution of SVC2 code 6:

```
  Zero filled
 ---------------------------------
|3 0| 3 0|3 0|3 0|3 2|3 1|3 4|3 8| Hex
 ---------------------------------
```

3.8.2 SVC2 Code 6, Option X'40'+n

If option X'40'+n is specified, the unsigned 32-bit binary number located in the user register 0 is converted to an ASCII hexadecimal number. The resulting number is stored right-justified in the receiving buffer with the left-most significant digits filled with ASCII spaces (hexadecimal 20).

3.8.3 SVC2 Code 6, Option X'80'+n

If option X'80'+n is specified, the unsigned 32-bit binary number located in the user register 0 is converted to an ASCII decimal number. The resulting number is stored right-justified in the buffer with the left-most significant digits filled with ASCII zeros (hexadecimal 30).

Example:

```
            PROG   CONVERT
            .
            .
            .
            LI     0,F'16322'
            .
            .
            .
            SVC    2,CONVERT
            .
            .
            .
            ALIGN 4
CONVERT     DB     X'80'+10,6
            DC     H'0'
            DCF    A(BUF)
BUF         DS     10
```

Register 0 before and after execution of SVC2 code 6:

```
 ---------------
|0 0| 0 0|3 F|C 2| Hex
 ---------------
```

Receiving buffer after execution of SVC2 code 6:

```
  Zero-filled
 ------------------------------------------
|3 0| 3 0|3 0|3 0|3 0|3 1|3 6|3 3|3 2|3 2| Hex
 ------------------------------------------
```

3.8.4 SVC2 Code 6, Option X'C0'+n

If option X'C0'+n is specified, the unsigned 32-bit binary number located in the user register 0 is converted to an ASCII decimal number. The resulting number is stored right-justified in the receiving buffer with the left-most significant digits containing zeros filled with ASCII spaces (hexadecimal 20).

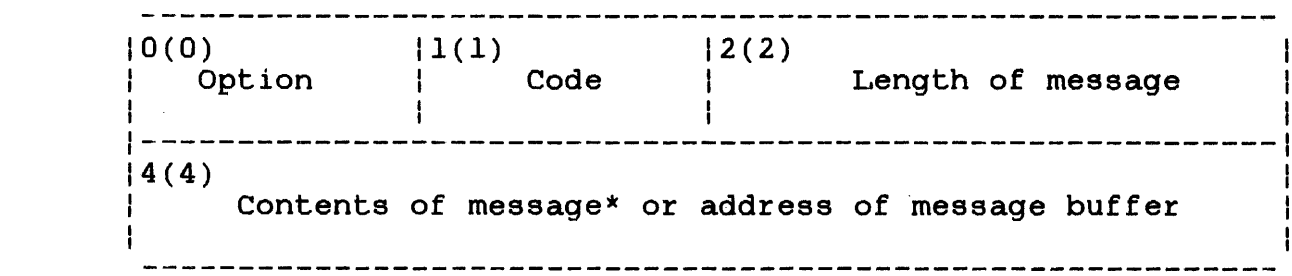
```
 ---------
|  SVC2   |
| CODE 7  |
 ---------
```

3.9 SVC2 CODE 7: LOG MESSAGE

SVC2 code 7 sends a user-specified message to the system console, user terminal or user-specified log device, depending on the task environment. This is accomplished through the SVC2 code 7 parameter block in Figure 3-12. Log devices for specific task environments are:

- System console for background tasks
- System console for foreground tasks
- User MTM terminal for MTM terminal tasks
- User-specified log device for MTM batch task

If no user-specified log device is allocated for MTM batch tasks, the message is lost.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,7
          DC    H'length of message'
          DC    C'contents of message' or
                A(message buffer)
```

* When the contents of message field is used, the size of the parameter block can vary.

Figure 3-12 SVC2 Code 7 Parameter Block Format and Coding

This parameter block is eight bytes long if the address of message buffer field is used. It is variable in length if the contents of message field is used. It must be fullword boundary-aligned and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option — is a 1-byte field that must contain one of the following options:

- Option X'00' indicates message contents, formatted.
- Option X'20' indicates message contents are formatted and sent to the system console only.
- Option X'40' indicates message at specified buffer address, formatted.
- Option X'60' indicates message at specified buffer address is formatted and sent to the system console only.
- Option X'80' indicates message contents, image mode.
- Option X'A0' indicates message contents are sent in image mode to system console only.
- Option X'C0' indicates message at specified buffer address, image mode.
- Option X'E0' indicates message at specified buffer address is sent in image mode to the system console only.

Code — is a 1-byte field that must contain the decimal number 7 to indicate SVC2 code 7.

Length of message — is a 2-byte field that must contain a decimal number indicating the number of bytes the message occupies. The message can be truncated by the log device driver. If the message is being logged to the system console, its maximum length is determined at sysgen time.

Contents of message	is a variable-length field that must contain the message to be sent to the log device.
Address of message buffer	is a 4-byte field that must contain the starting address of the buffer that contains the message to be sent to the log device. This buffer can be on any byte boundary.

When the message is sent to the appropriate log device, it is either formatted or in image mode. When a formatted message is sent to a device:

- All trailing blanks in the buffer or at the end of the message are eliminated.
- A carriage return (CR) and line feed (LF) are automatically appended to the message.
- The message terminates when the end of the buffer or message is reached or when a CR is found in the message.

When a message is sent to a device in image mode, the message terminates when the end of the buffer or message is reached. If in image mode, a message with multiple lines can be sent by executing a single SVC2 code 7 for each line. However, each line should include a CR and LF at the end. The image options should be used with caution because the amount of time that must remain for a CR to occur differs on various console devices.

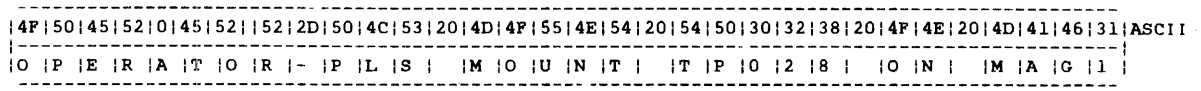
3.9.1 SVC2 Code 7, Option X'00'

If option X'00' is specified, the message specified in the parameter block is formatted and sent to the appropriate log device.

Example:

```
            SVC   2,LOGMSG
            .
            .
            .
            ALIGN 4
LOGMSG      DB    X'00',7
            DC    H'32'
            DC    C'OPERATOR-PLS MOUNT TP028 ON MAG1'
```

Contents of message buffer before and after execution of SVC2 code 7:



Log device after execution of SVC2 code 7:

```
OPERATOR-PLS MOUNT TP028 ON MAG1
```

3.9.2 SVC2 Code 7, Option X'20'

If option X'20' is specified, the message specified in the parameter block is formatted as for option X'00'. The message is then sent unconditionally to the system console.

Option X'20' is used exclusively for tasks running under MTM.

3.9.3 SVC2 Code 7, Option X'40'

If option X'40' is specified, the contents of the message buffer pointed to by the address specified in the parameter block are formatted and sent to the appropriate log device.

3.9.4 SVC2 Code 7, Option X'60'

If option X'60' is specified, the contents of the message buffer are formatted as for option X'40'. The message is then sent unconditionally to the system console.

Option X'60' is used exclusively for tasks running under MTM.

3.9.5 SVC2 Code 7, Option X'80'

If option X'80' is specified, the message specified in the parameter block is in image mode and is sent to the appropriate log device.

Example:

```
            SVC   2,LOGMSG1
            SVC   2,LOGMSG2
            .
            .
            .
            ALIGN 4
LOGMSG1     DB    X'80',7
            DC    H'32'
            DC    C'OPERATOR-PLS MOUNT TP028 ON MAG1'
            ALIGN 4
LOGMSG2     DB    X'80',7
            DC    H'19'
            DC    C'SET TAPE AT 800 BPI'
```

Contents of message buffer before and after execution of SVC2 code 7:

4F	50	45	52	41	54	4F	52	2D	50	4C	53	20	4D	4F	55	4E	54	20	54	50	30	32	38	20	4F	4E	20	4D	41	46	31	ASCII
O	P	E	R	A	T	O	R	-	P	L	S		M	O	U	N	T		T	P	0	2	8		O	N		M	A	G	1	

Contents of message buffer before and after execution of second SVC2 code 7:

53	45	54	20	54	41	50	45	20	41	54	20	38	30	30	20	42	50	49	ASCII
S	E	T		T	A	P	E		A	T		8	0	0		B	P	I	

Log device after execution of second SVC2 code 7:

```
SET TAPE AT 800 BPI TP028 ON MAG1
```

(no line feed appended, message overwritten)

3.9.6 SVC2 Code 7, Option X'A0'

If option X'A0' is specified, the message specified in the parameter block is in image mode, as for option X'80', but the message is sent unconditionally to the system console.

Option X'A0' is used exclusively for tasks running under MTM.

3.9.7 SVC2 Code 7, Option X'C0'

If option X'C0' is specified, the contents of the message buffer pointed to by the address specified in the parameter block are in image mode and are sent to the appropriate log device.

3.9.8 SVC2 Code 7, Option X'E0'

If option X'E0' is specified, the contents of the message buffer are in image mode as for option X'C0', but the message is sent unconditionally to the system console.

Option X'E0' is used exclusively for tasks running under MTM.

3.10 SVC2 CODE 8: INTERROGATE CLOCK

SVC2 code 8 sends the current time of day to a user-specified buffer. This is accomplished through the SVC2 code 8 parameter block shown in Figure 3-13.

0(0) Option	1(1) Code	2(2) Reserved
4(4) Address of receiving buffer		

```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,8
          DC    H'0'
          DCF   A(receiving buffer)
```

Figure 3-13 SVC2 Code 8 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	This 1-byte field must contain one of the following options: • Option X'00' returns the time of day as two fullwords of ASCII data in the form hh:mm:ss. • Option X'80' returns the time of day as a fullword of binary data indicating the number of seconds past midnight. • Option X'40' returns the time of day as three fullwords of ASCII data in the form hh:mm:ss.sss. • Option X'C0' returns the time of day as two fullwords of binary data. The first fullword indicates the number of seconds past midnight; the second fullword indicates the number of milliseconds past midnight.
Code	is a 1-byte field that must contain the decimal number 8 to indicate SVC2 code 8.
Reserved	is a reserved 2-byte field that must contain zeros.
Address of receiving buffer	is a 4-byte field that must contain the starting address of the buffer to receive the current time of day.

The current time of day is calculated as seconds from midnight (midnight equals 0) and is taken from the line frequency clock (LFC) maintained by the system.

3.10.1 SVC2 Code 8, Option X'00'

If option X'00' is specified, the current time of day is returned in ASCII format to a user-specified buffer located in a task-writable segment This buffer must be at least eight bytes long. The current time of day is returned as follows.

Format:

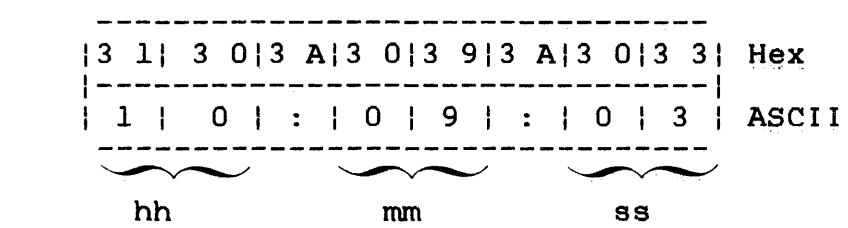
```
hh:mm:ss
```

Parameters:

hh	are two ASCII characters representing the number of hours.
mm	are two ASCII characters representing the number of minutes.
ss	are two ASCII characters representing the number of seconds.

Example:

Contents of buffer after execution of SVC2 code 8 option X'00' when the current time of day is 10:09:03:

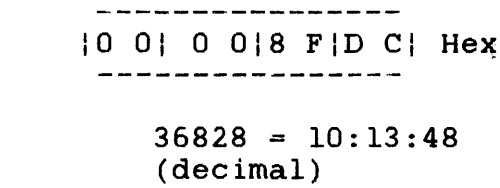


3.10.2 SVC2 Code 8, Option X'80'

If option X'80' is specified, the current time of day in seconds from midnight is sent in binary format to a user-specified buffer located in a task-writable segment. This buffer must be at least four bytes long and aligned on a fullword boundary.

Example:

Contents of buffer after execution of SVC2 code 8 option X'80' when the current time of day is 10:13:48:



The contents of this buffer represent 36,828 seconds from midnight.

3.10.3 SVC2 Code 8, Option X'40'

If option X'40' is specified, the current time of day is returned in ASCII format to a user-specified buffer in a task-writable segment. This buffer must be at least 12 bytes long. The current time of day is returned as follows.

Format:

hh:mm:ss:sss

Parameters:

hh	are two ASCII characters representing the number of hours.
mm	are two ASCII characters representing the number of minutes.
ss	are two ASCII characters representing the number of seconds.
sss	are three ASCII characters representing the number of milliseconds.

Example:

Contents of buffer after execution of SVC2 code option X'40', when the current time of day is 10:41:32.8:

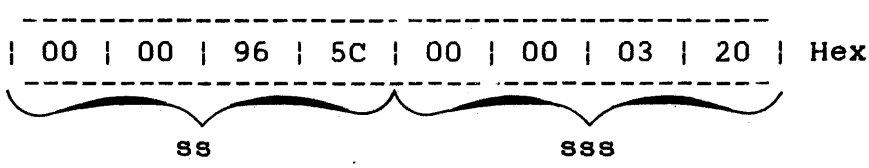
31	30	3A	34	31	3A	33	32	3A	38	30	30	Hex
1	0	:	4	1	:	3	2	:	8	0	0	ASCII
	hh			mm			ss			sss		

3.10.4 SVC2 Code 8, Option X'C0'

If option X'C0' is specified, the current time of day in seconds and milliseconds from midnight is sent in binary format to a user-specified buffer located in a task-writable segment. This buffer must be eight bytes long and fullword boundary-aligned.

Example:

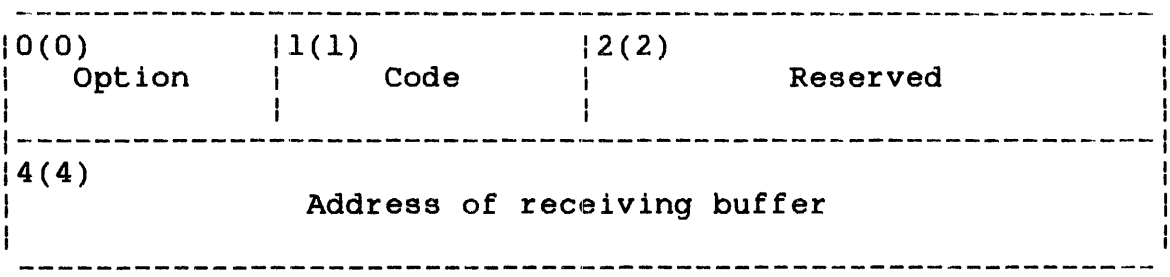
Contents of buffer after execution of SVC2 Code 8 option X'C0' when the current time of day in ASCII is 10:41:32.8:



The contents of this buffer represent 38,492 seconds and 800ms from midnight.

3.11 SVC2 CODE 9: FETCH DATE

SVC2 code 9 sends the current date to a user-specified buffer. This is accomplished through the SVC2 code 9 parameter block shown in Figure 3-14.




```
            SVC   2,parblk
            .
            .
            .
            ALIGN 4
parblk      DB    0,9
            DC    H'0'
            DCF   A(receiving buffer)
```

Figure 3-14 SVC2 Code 9 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain a 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 9 to indicate SVC2 code 9.
Reserved	is a reserved 2-byte field that must contain zeros.

Address of receiving buffer	is a 4-byte field that must contain the starting address of the buffer receiving the current date. The buffer must be eight bytes long and located in a task-writable segment. The buffer can be located on any boundary.

SVC2 code 9 sends the current date to the receiving buffer in either one of the following:

Format:

mm/dd/yy

or

dd/mm/yy

Parameters:

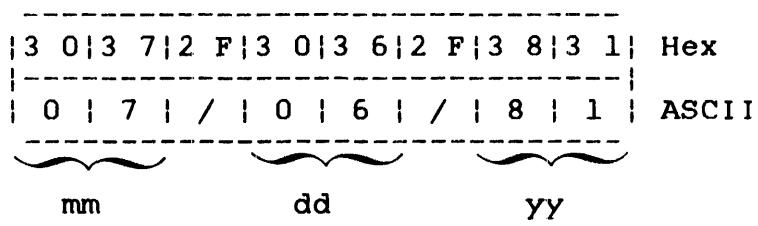
mm	are two ASCII characters representing the month.
dd	are two ASCII characters representing the day.
yy	are two ASCII characters representing the year.

When the system is installed, one of these formats is chosen as the default for all operations. To return the current date in the alternate format, use the DATE command at sysgen.

Example:

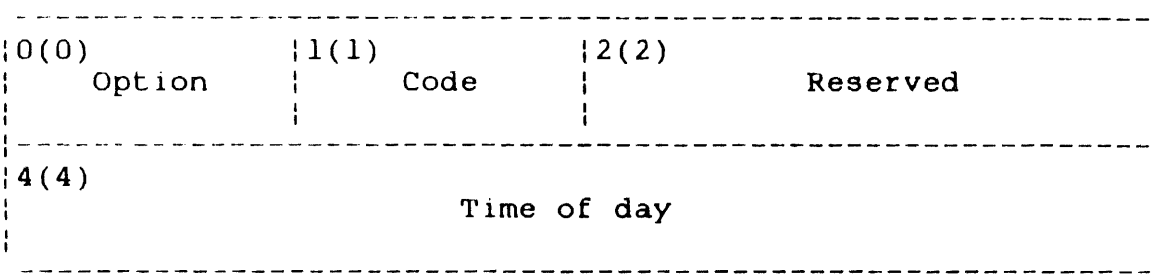
```
            SVC    2,DATE
            SVC    2,PAUSE
            .
            .
            .
            ALIGN  4
DATE        DB     0,9
            DC     H'0'
            DCF    A(BUF)
            ALIGN  4
PAUSE       DB     0,1
BUF         DS     8
```

Contents of receiving buffer after execution of SVC2 code 9 when the current date in ASCII is 07/06/81:



3.12 SVC2 CODE 10: TIME OF DAY WAIT

SVC2 code 10 suspends the SVC calling task until a user-specified time of day occurs. Then the calling task resumes execution. This is accomplished through the SVC2 code 10 parameter block shown in Figure 3-15.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,10
          DC    H'0'
          DC    Y'time of day'
```

Figure 3-15 SVC2 Code 10 Parameter Block Format and Coding

The SVC2 code 10 parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain a 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 10 to denote SVC2 code 10.
Reserved	is a reserved 2-byte field that must contain zeros.

Time of day

is a 4-byte field that must contain a decimal number from 0 to 268,435,455 (2^{28} -1) representing in seconds a specific time of day when the calling task is to start execution. The decimal number specified must be calculated in seconds from midnight.

- 0 seconds equals 00:00:00 a.m. (midnight) of the current day.
- 86,399 seconds equals 23:59:59 p.m. of the current day.

See Table 3-2 for a range of values in seconds and their corresponding time of day. Any value greater than 86,399 refers to days in the future. If the specified time of day has passed, the same time on the following day is assumed.

TABLE 3-2 TIME OF DAY VALUES CALCULATED IN SECONDS FROM MIDNIGHT

DAY	TIME OF DAY 00:00:00 HOURS (MIDNIGHT)	TIME OF DAY 23:59:59 HOURS (P.M.)
1st (current)	0	86,399
2nd	86,400	172,799
3rd	172,800	259,199
4th	259,200	345,599
5th	345,600	431,999
6th	432,000	518,399
7th	518,400	604,799
.	.	.
.	.	.
.	.	.
3,107th (maximum)	268,358,400	268,435,455* (maximum)

* 268,435,455 seconds equals 21:24:15 hours of the final day

After executing SVC2 code 10, the CC is set to indicate if the call was successful. The possible CC settings follow.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	Sufficient system space is unavailable; no wait occurred

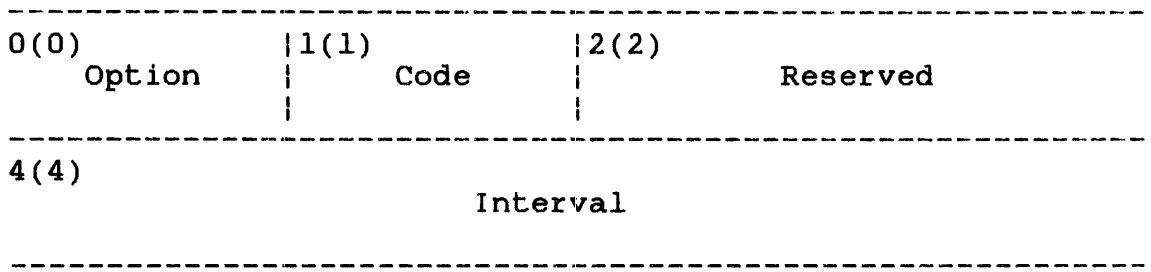
If this call is executed and insufficient system space exists, no wait occurs and the CC is set to 4 (V bit set).

Example:

```
            SVC    2,WAITDAY
            SVC    2,PAUSE
            .
            .
            .
            ALIGN  4
WAITDAY     DB     0,10
            DC     H'0'
            DC     F'12165'            EQUAL to 03:22:45 AM
            ALIGN  4
PAUSE       DB     0,1
```

3.13 SVC2 CODE 11: INTERVAL WAIT

SVC2 code 11 suspends the SVC calling task until a user-specified interval occurs. When the specific interval elapses, the calling task begins execution. This is accomplished through the SVC2 code 11 parameter block shown in Figure 3-16.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,11
          DC    H'0'
          DC    F'interval'
```

Figure 3-16 SVC2 Code 11 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 11 to indicate SVC2 code 11.

Reserved — is a reserved 2-byte field that must contain zeros.

Interval — is a 4-byte field that must contain a decimal number from 0 to 268,435,455 (2^{31} -1) representing in milliseconds the interval that must elapse before the calling task resumes execution. The interval starts when this call is executed and ends after the specified milliseconds elapse.

After executing SVC2 code 11, the CC is set to indicate if the call was successful. The possible CCs are:

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	Sufficient system space is unavailable; no wait occurred

If this call is executed and insufficient system space exists, no wait occurs and the CC is set to 4 (V bit set).

Example:

```
            SVC   2,WAITINT
            SVC   2,PAUSE
            .
            .
            .
            ALIGN 4
WAITINT     DB    0,11
            DC    H'0'
            DC    F'32768'              EQUAL TO 32.768 SECONDS
            ALIGN 4
PAUSE       DB    0,1
```

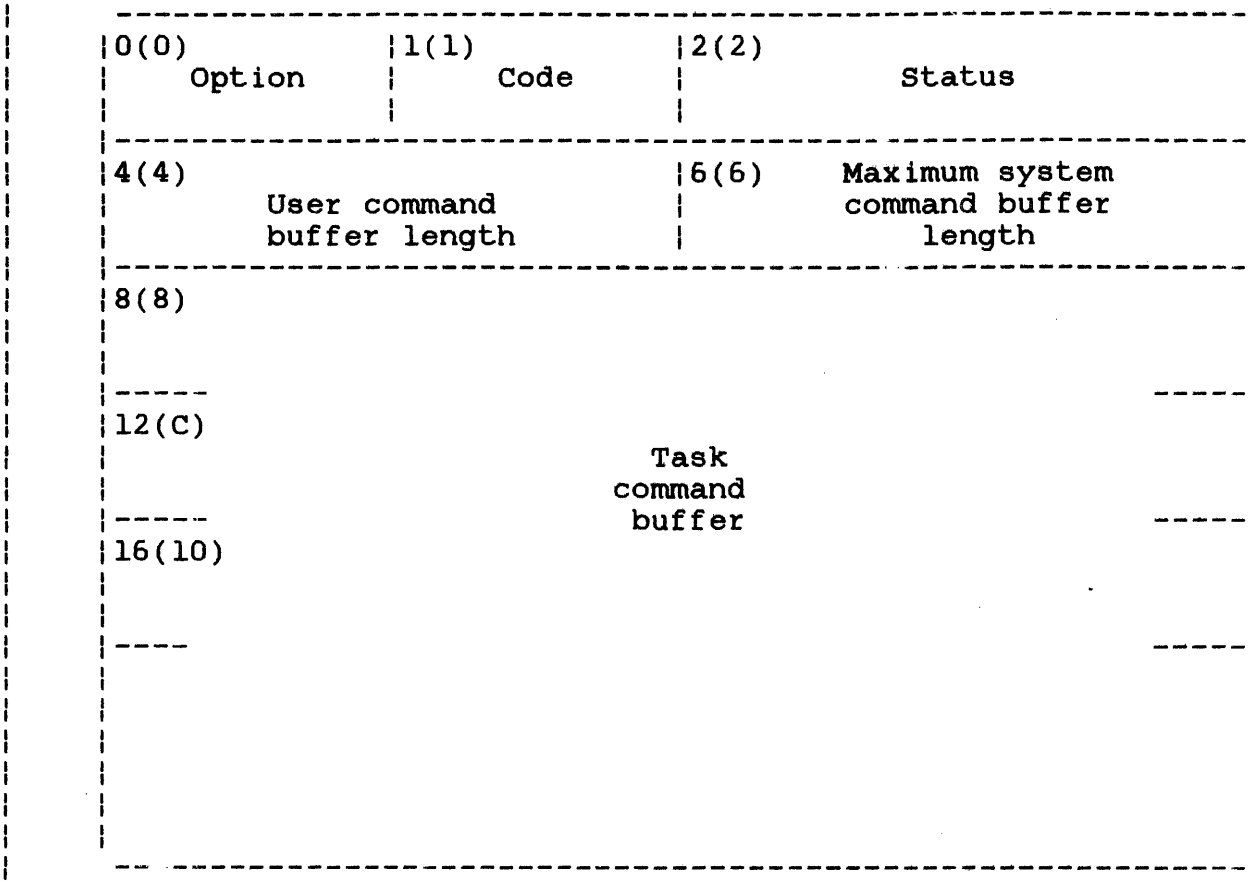
3.14 SVC2 CODE 14: INTERNAL READER

SVC2 code 14 allows a foreground task loaded from the system console to invoke operator and CSS commands. These commands are sent to the command processor where they are executed as if they were entered from the system console. SVC2 code 14 provides two options for sending commands to the command processor. Option 0 allows the user to place the commands directly in the task command buffer field of the SVC2 code 14 parameter block. Option 1 allows the user to store the commands in a task command buffer located on a fullword boundary within the task's address space. The address of this buffer is placed in the parameter block.

SVC2 code 14 transfers the commands in a task command buffer until the end of the buffer is reached. The parameter blocks for both SVC2 code 14 options are described in the following sections.

3.14.1 SVC2 Code 14, Parameter Block for Option 0

The parameter block format for option 0 of SVC2 code 14 is shown in Figure 3-17.




```
          SVC   2,parblk
            .
            .
            .
parblk    DB    0,14,0,0
          DC    H'user command buffer length'
          DC    H'0'
          DC    'operator commands'
```

Figure 3-17 SVC2 Code 14 Parameter Block Format and Coding for Option 0

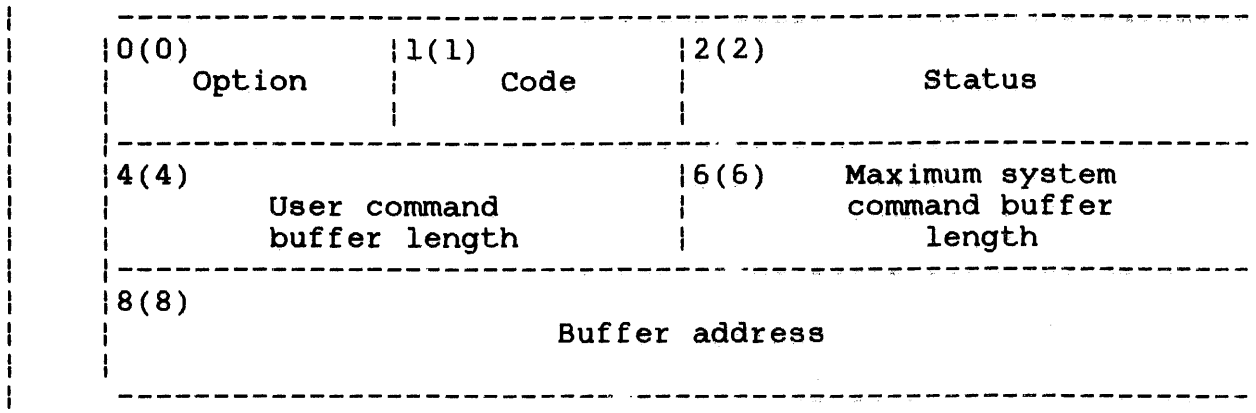
This parameter block can be up to 1,032 bytes long. It must be aligned on a fullword boundary and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that contains a value of 0 to indicate that the task command buffer is contained in the SVC2 code 14 parameter block.
Code	is a 1-byte field that contains the decimal value 14 to indicate SVC2 code 14.
Status	is a 2-byte field that receives a status code indicating the status of the SVC processing.
User command buffer length	is a 2-byte field specifying a decimal number indicating the maximum length allowed for the user command buffer.
Maximum system command buffer length	is a 2-byte field to which the operating system returns the system command buffer length established by CMDLEN at sysgen. This value is returned only for status code X'0003' (see Table 3-3).
Task command buffer	is a variable length field with a maximum length of 1,024 bytes. This field contains the commands to be sent to the command processor.

3.14.2 SVC2 Code 14, Parameter Block for Option 1

The parameter block format for option 1 of SVC2 code 14 is shown in Figure 3-18.




```
          SVC   2,parblk
            .
            .
            .
parblk    DB    1,14,0,0
          DC    H'user command buffer length'
          DC    H'0'
          DAC   BUFADR
```

Figure 3-18 SVC2 Code 14 Parameter Block Format and Coding for Option 1

This parameter block is 12 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that contains a value of 1 to indicate that the parameter block contains the address of the task command buffer.
Code	is a 1-byte field that contains the decimal value 14 to indicate SVC2 code 14.
Status	is a 2-byte field that receives a status code indicating the status of the SVC processing. See Table 3-2 for a list of the SVC2 code 14 status codes.
User command buffer length	is a 2-byte field specifying a decimal number indicating the maximum length allowed for the task command buffer.

Maximum system command buffer	is a 2-byte field to which the operating system returns the system command buffer length established by CMDLEN at sysgen. This value is returned only for status code X'0003' (see Table 3-3).
Buffer address	is a 4-byte field specifying the address of the task command buffer. This buffer must be located on a fullword boundary within the task's address space.

3.14.3 SVC2 Code 14, Status Codes

The status codes for each of the SVC2 code 14 options are listed in Table 3-3.

TABLE 3-3 SVC2 CODE 14, STATUS CODES

CODE	MEANING
X'0000'	Successful completion - commands are sent to the command processor for execution.
X'0001'	No free internal reader buffers are available.
X'0002'	Option error - invalid option is specified for SVC.
X'0003'	User-specified the length of command buffer incorrectly. The length of the maximum allowed system command buffer is returned to the maximum system command length field.
X'FFFF'	No internal reader command buffers defined.

3.14.4 SCV2 Code 14, Programming Considerations

Support for the internal reader must be included in the system at sysgen. This is accomplished through the Sysgen/32 command, IREADER. See the OS/32 System Generation (Sysgen/32) Reference Manual. If the internal reader is not included at sysgen, an attempt to execute an SVC2 code 14 results in an execution error and an illegal SVC message is sent to the user console.

The internal reader requires a set of buffers to receive the commands sent to it by SVC2 code 14. The OS/32 operator command, IRBUFFER, is used to create command buffers for the internal reader. See the OS/32 Operator Reference Manual. The IRBUFFER command can also be used to increase the number of command buffers when no free buffers are available (status code X'0001'). IRBUFFER can be used at any time if support for the internal reader has been generated into the system.

The following program demonstrates the use of SVC2 code 14.

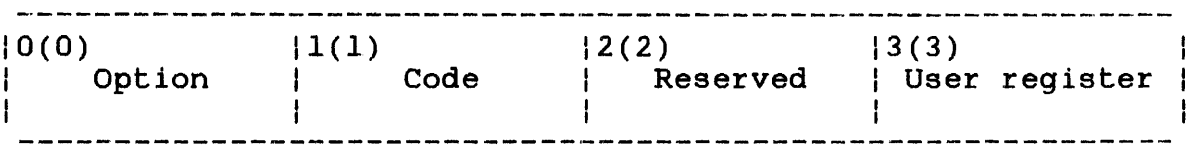
Sample SVC2 code 14 program:

```
SVC214   PROG  SVC2,14 EXAMPLE
         SVC   2,COMMAND0          SEND COMMAND
         LH    0,COMMAND0+2        WAS IT SUCCESSFUL?
         BNZ   STOP                NO  - ERROR
         SVC   2,COMMAND1          SEND COMMAND
         LH    0,COMMAND1+2        WAS IT SUCCESSFUL?
         BNZ   STOP                NO  - ERROR
         SVC   3,0                 EOT

STOP     EQU   *
         SVC   2,PAUSE             PAUSE
         SVC   3,0                 EOT



         ALIGN 4
PAUSE    DB    0,1,0,0



         ALIGN 4
COMMAND0 DB    0,14,0,0            DIRECT COMMAND BUFFER
         DC    Z(4)
         DCX   0
         DC    C' D M '



         ALIGN 4
COMMAND1 DB    1,14,0,0             INDIRECT COMMAND BUFFER ADDRESS
         DC    Z(CMDBUFFE-CMDBUFF)
         DCX   0
         DC    A(CMDBUFF)
         ALIGN 4
CMDBUFF  DC    C'$WR ** CSS CALL BY IREADER ***;    CALLCSS.CSS    '
CMDBUFFE EQU   *
         END
```

3.15 SVC2 CODE 15: CONVERT ASCII HEXADECIMAL OR ASCII DECIMAL TO BINARY

SVC2 code 15, the inverse of SVC2 code 6, converts an ASCII decimal or hexadecimal number to an unsigned 32-bit binary number. Character strings can be input in either upper-or lower-case.

The result is stored in the user register 0. This is accomplished through the SVC2 code 15 parameter block shown in Figure 3-19.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,15,0
          DB    user register
```

Figure 3-19 SVC2 Code 15 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option — is a 1-byte field that must contain one of the following options:

- Option X'00' converts ASCII hexadecimal to binary.

- Option X'40' converts ASCII hexadecimal to binary, skips leading spaces.
- Option X'80' converts ASCII decimal to binary.
- Option X'C0' converts ASCII decimal to binary, skips leading spaces.

Code	is a 1-byte field that must contain the decimal number 15 to denote SVC2 code 15.
Reserved	is a reserved 1-byte field and must contain 0.
User register	is a 1-byte field that must contain the user-specified register number. This register should contain the address of the buffer that contains the ASCII hexadecimal or ASCII decimal number to be converted. This buffer can be located on any boundary. After executing SVC2 code 15, register 0 contains the result, and the user-specified register contains the address of the byte following the last number to be converted.

The valid ASCII hexadecimal numbers are 0 through 9 and A through F. The valid ASCII decimal numbers are 0 through 9. Any character other than those ASCII hexadecimal and ASCII decimal numbers specified causes the conversion process to stop, the nonconverted byte address to be stored in the user-specified register, and the CC to be set to 0. The possible CC settings that can occur after executing SVC2 code 15 follow.

Condition Code:

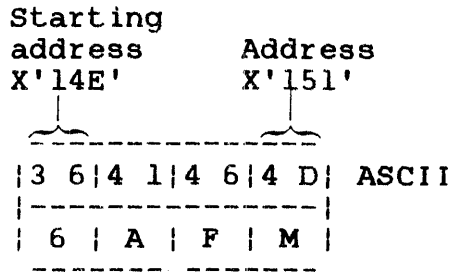
C	V	G	L	
0	0	0	0	Normal termination
0	0	0	1	No numbers converted; register 0 contains zeros
0	1	0	0	Value of the number to be converted is greater than 2,147,483,647 (2 -1)

3.15.1 SVC2 Code 15, Option X'00'

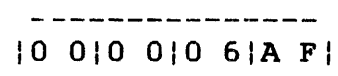
If option X'00' is specified, the ASCII encoded hexadecimal number in the buffer specified by the address in the user register is converted to a binary number. The resulting number is stored right-justified in register 0 with the left-most significant bits (MSBs) filled with zeros.

Example:

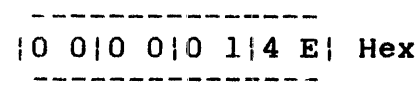
Buffer before and after execution of SVC2 code 15:



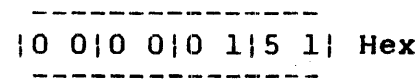
Register 0 after execution of SVC2 code 15:



User-specified register before execution of SVC2 code 15:



User-specified register after execution of SVC2 code 15:



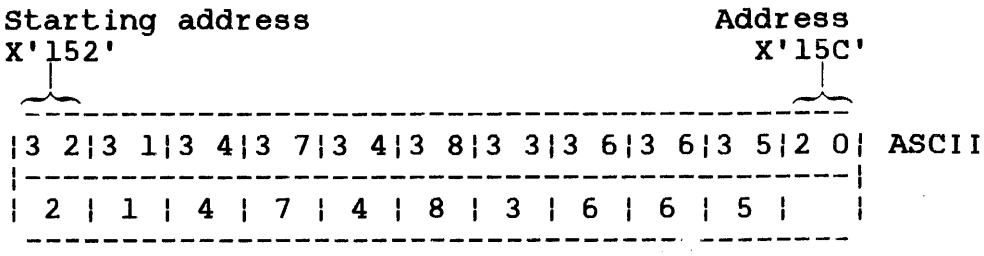
If a number other than a valid ASCII number is specified, that number is not converted, and the address is stored in the user-specified register.

If an ASCII number is preceded by at least one space, no processing takes place, the contents of the user-specified register remain the same, register 0 contains all zeros, and the CC is set to 1.

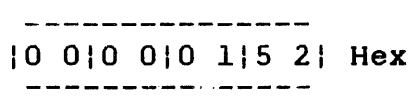
If the value of the ASCII number is greater than 2,147,483,647 (2^{31} -1), the number is converted, the resulting number is stored right-justified in register 0 with the left-most significant bits truncated, and the CC is set to 4 (V bit set).

Example:

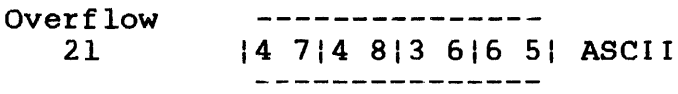
Buffer before and after execution of SVC2 code 15:



User-specified register before execution of SVC2 code 15:



Register 0 after execution of SVC2 code 15:



User-specified register after execution of SVC2 code 15:

Condition Code:

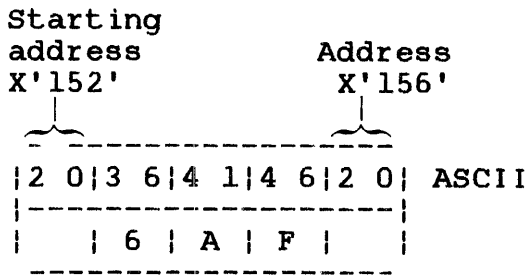
C	V	G	L	
0	1	0	0	ASCII number greater than 2^{31} -1

3.15.2 SVC2 Code 15, Option X'40'

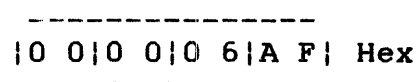
If option X'40' is specified, the ASCII-encoded hexadecimal number in the buffer, specified by the address in the user register, is converted to a binary number with leading spaces ignored during the conversion. The resulting number is stored right-justified in register 0 with the left-most significant bits filled with zeros.

Example:

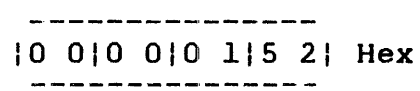
Buffer before and after execution of SVC2 code 15:



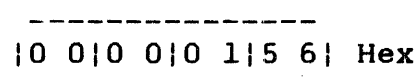
Register 0 after execution of SVC2 code 15:



User-specified register before execution of SVC2 code 15:



User-specified register after execution of SVC2 code 15:



Condition Code:

C	V	G	L	
0	0	0	0	Normal termination

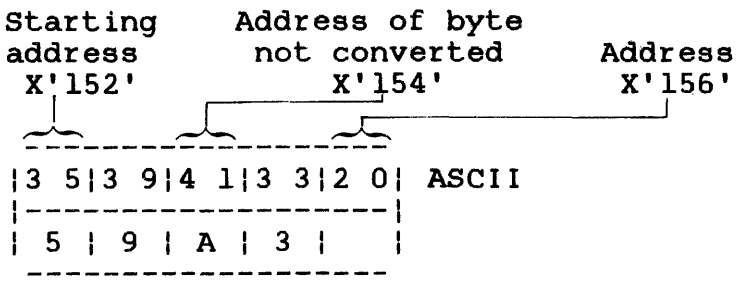
3.15.3 SVC2 Code 15, Option X'80'

If option X'80' is specified, the ASCII-encoded decimal number in the buffer, specified by the address in the user register, is converted to a binary number. The resulting binary number is stored right-justified in register 0 with the left-most significant bits filled with zeros.

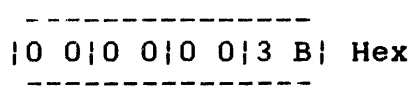
If a character other than a valid ASCII decimal number is specified, that character is not converted and the invalid character address is stored in the user-specified register.

Example:

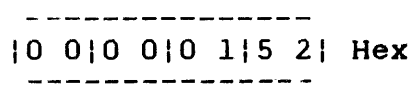
Buffer before and after execution of SVC2 code 15:



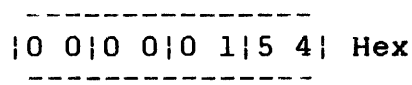
Register 0 after execution of SVC2 code 15:



User-specified register before execution of SVC2 code 15:



User-specified register after execution of SVC2 code 15:



Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	ASCII number greater than 2^{31} -1

If a decimal number represented in ASCII code is preceded by at least one space, no processing takes place, the contents of the user-specified register remain the same, register 0 contains all zeros, and the CC is set to 1.

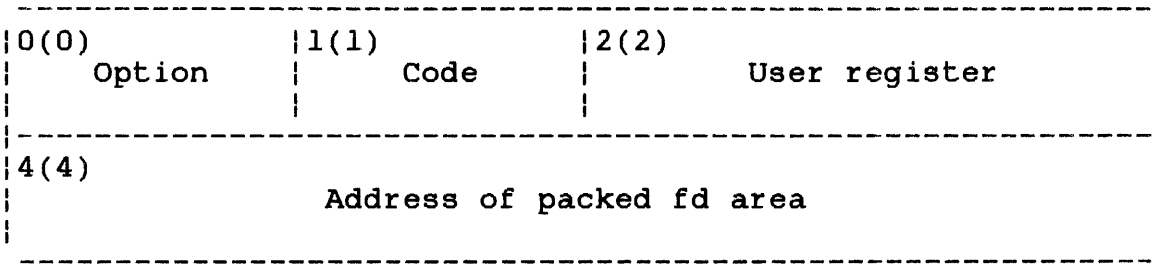
If the value of the ASCII decimal number is greater than 2,147,483,647 (2^{31} -1), the number is converted, the resulting binary number is stored right-justified in register 0 with the left-most significant bits truncated, and the CC is set to 4 (V bit set).

3.15.4 SVC2 Code 15, Option X'C0'

If option X'C0' is specified, the ASCII-encoded decimal number in the buffer, specified by the address in the user register, is converted to a binary number, with leading spaces ignored during the conversion. The resulting number is stored right-justified in register 0 with the left-most significant bits filled with zeros.

3.16 SVC2 CODE 16: PACK FILE DESCRIPTOR

SVC2 code 16 formats a user-specified unpacked fd to the packed format used within the SVC7 parameter block (see bytes 8 through 23 of the SVC7 parameter block). Figure 3-20 illustrates the SVC2 code 16 parameter block format.




```
            SVC   2,parblk
            .
            .
            .
            ALIGN 4
parblk      DB    option,16
            DC    H'user register number'
            DCF   A(packed fd area)
```

Figure 3-20 SVC2 Code 16 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option — is a 1-byte field that must contain one of the following options:

- X'00' indicates the default volume is the user volume.

- X'40' indicates the default volume is the user volume. Skip leading spaces.
- X'10' indicates the default volume is the system volume.
- X'50' indicates the default volume is the system volume. Skip leading spaces.
- X'20' indicates the default volume is the spool time.
- X'60' indicates the default volume is the spool volume. Skip leading spaces.
- X'80' indicates there is no default volume.
- X'C0' indicates there is no default volume. Skip leading spaces.

In a non-MTM environment, the default user volume is the same as the default system volume. Options X'00' or X'40' are preferred, since they are compatible with normal usage in an MTM environment.

NOTE

The above options are intended for use by nonprivileged u-tasks only. These options pack fds that use the /P, /G or /S file classification. If a privileged u-task or e-task uses these options to pack an fd with a /P, /G or /S file classification, the resulting packed fd has an account number in its file class field. See Section 3.14.9 for the SVC2 code 16 options for e-tasks or privileged u-tasks.

Code	is a 1-byte field that must contain the decimal number 16 to indicate SVC2 code 16.
User register	is a 2-byte field that must contain the user-specified register number containing the unpacked fd address.
Address of packed fd area	is a 4-byte field that must contain the address of the area that receives the packed file descriptor.

The CCs set after packing an fd follow.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	0	0	1	No volume name present in unpacked fd
0	0	1	0	An account number or file class present in unpacked fd
0	1	0	0	Syntax error present in unpacked fd
1	0	0	0	No extension present in unpacked fd

If more than one condition results from a pack fd operation, a combination of CCs are set.

NOTE

When a period followed by no valid characters is specified in the unpacked fd, it is treated as an explicit request for an extension containing spaces. The CC is set to 8 (C bit set).

All lower-case characters in the user-specified fd are converted to their equivalent upper-case characters after the pack fd operation occurs. The entire user-specified fd (unpacked format) can be from 1 to 19 characters. Allowable characters are:

- A through Z (upper-case)
- a through z (lower-case)
- 0 through 9 (numerics)
- selected special characters (symbols)

The format of the user-specified fd is:

Format:

$$\left[\left\{\begin{matrix}\text{voln}\\ \text{dev}\end{matrix}\right\}:\right]\ [\text{filename}]\ \left[.[\text{ext}]\right]\left[/\left\{\begin{matrix}\text{P}\\ \text{G}\\ \text{S}\\ \text{actno}\end{matrix}\right\}\right]$$

Parameters:

voln or dev:	is a disk volume or device name from one to four characters.
filename	is a filename from one to eight alphanumeric characters.
.ext	is the extension name of from one to three characters, preceded by a period.
/P /G /S	are single alphabetic characters representing the file class. They are: P for private file, G for group file and S for system file.
actno	is an account number ranging from 0 to 65,535.

The area receiving the packed fd must be 16 bytes long, fullword boundary-aligned, and located in a task-writable segment (see Figure 3-21). Since this area is identical to bytes 8 through 23 of the SVC7 parameter block, these bytes can be designated as the receiving area.

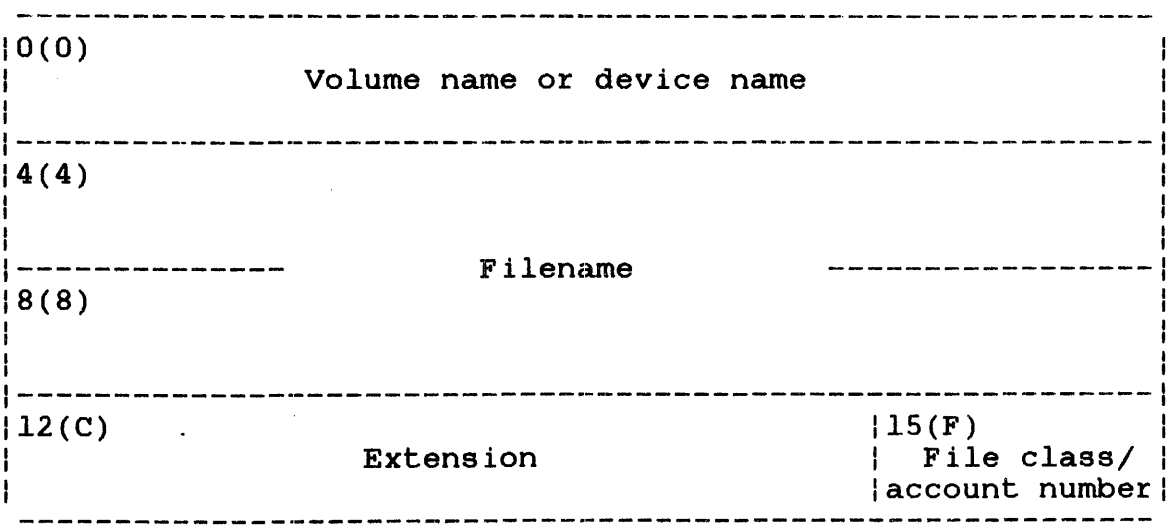


Figure 3-21 Packed File Descriptor Area

Fields:

Volume name or device name	is a 4-byte field that receives the packed format of the volume name or device name. If the volume or device name is less than four bytes, it is left-justified with spaces (X'20'). If no volume or device name is specified, the user-specified option determines the result.
Filename	is an 8-byte field that receives the packed format of the user-specified filename. If the filename is less than eight bytes, it is left-justified with spaces (X'20'). If no filename is specified, this field is filled with spaces.
Extension	is a 3-byte field that receives the packed format of the user-specified extension. If the extension is less than three bytes, it is left-justified with spaces (X'20'). If no extension is specified, this field is filled with spaces.
File class/ account number	is a 1-byte field that receives the packed format of the user-specified file class. Any value other than P, G or S in the file class field of the unpacked fd causes a syntax error. If no file class is specified in the unpacked fd, an S is returned in the class field of the packed fd when running under the OS. P is returned in the class field of the packed fd when running under MTM.

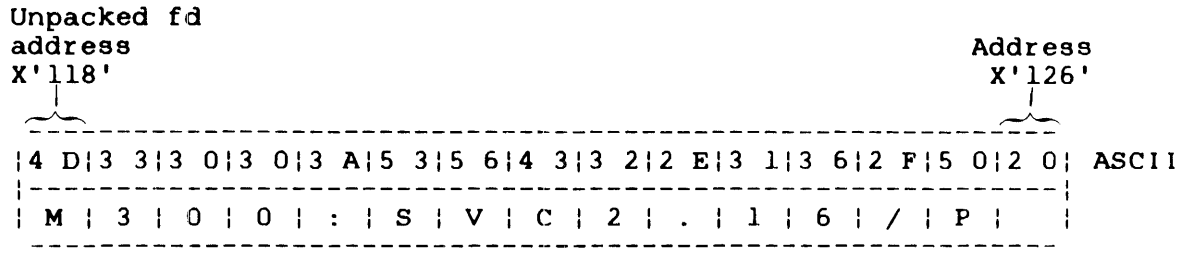
NOTE

If the SVC2 code 16 options for privileged tasks are used, an account number is returned to this field (see Section 3.16.9).

After the pack fd operation occurs, the user-specified register contains the address of the byte following the unpacked fd. If a syntax error is detected, the user-specified register contains the address of the first byte of the unpacked fd. The following examples show the results of issuing an SVC2 code 16 for a task running under MTM. The default system volume is M300.

When a device name is encountered in the user-specified fd, the pack fd operation returns spaces to the filename, extension and file class/account number fields of the packed fd.

Example 1:



User-specified register before packing fd:

00	00	01	18	Hex

Packed fd:

4 D	3 3	3 0	3 0	5 3	5 6	4 3	3 2	2 0	2 0	2 0	2 0	3 1	3 6	2 0	5 0	ASCII
M	3	0	0	S	V	C	2					1	6		P	

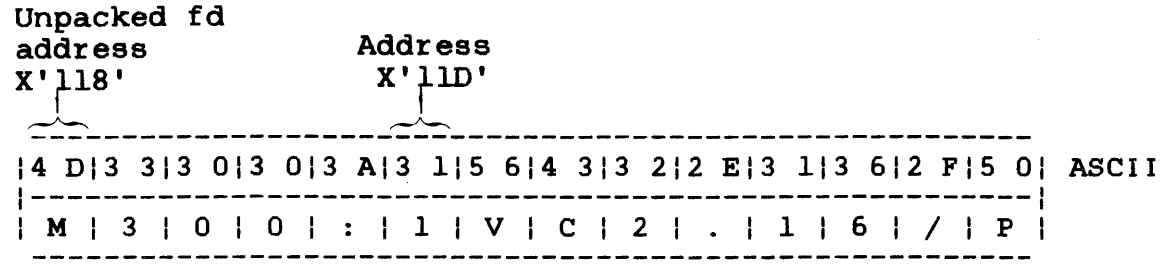
User-specified register after packing fd:

00	00	01	26	Hex

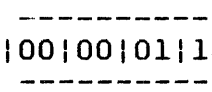
Condition Code:

C	V	G	L	
0	0	0	0	Normal termination

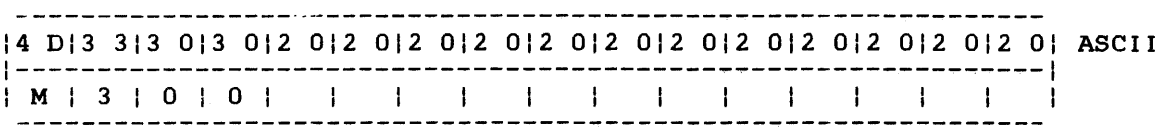
Example 2:



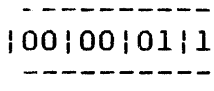
User-specified register before packing fd:



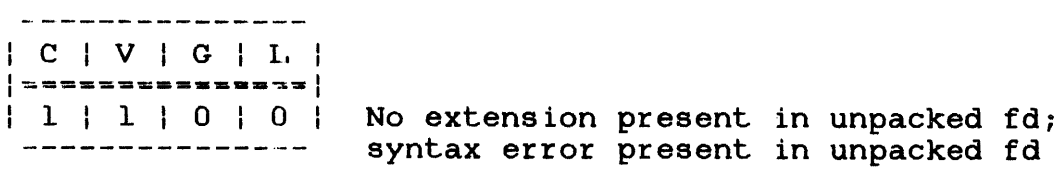
Packed fd:



User-specified register after packing fd:

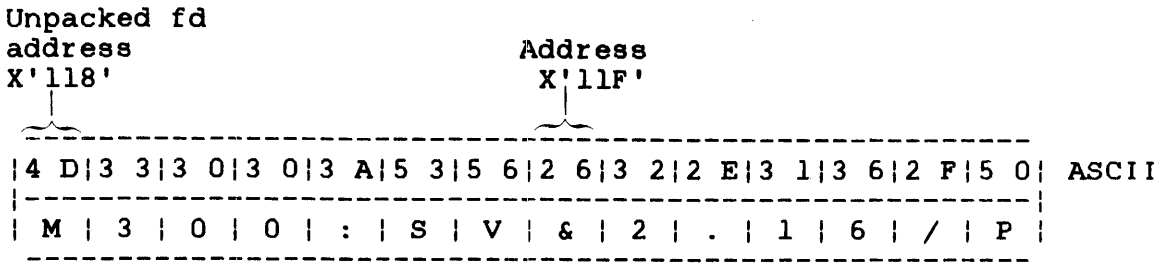


Condition Code:

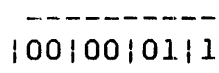


The unpacked fd contains a character that was interpreted as a field separator.

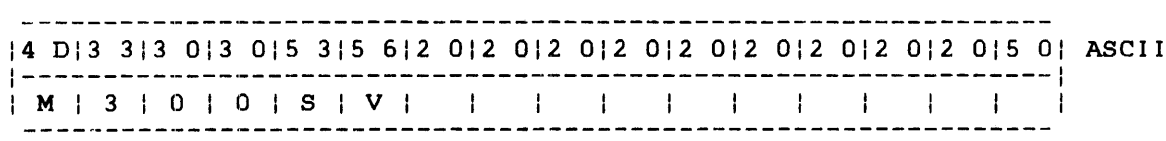
Example 3:



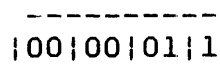
User-specified register before packing fd:



Packed fd:



User-specified register after packing fd:



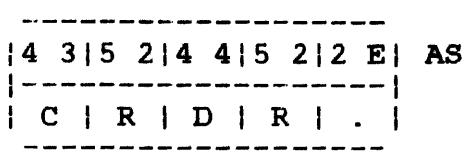
Condition Code:

C	V	G	L		
1	1	0	0	No extension present in unpacked fd; syntax error present in unpacked fd	

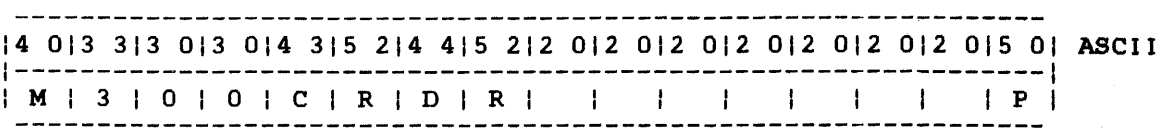
The above example shows an illegal character within the filename.

Example 4:

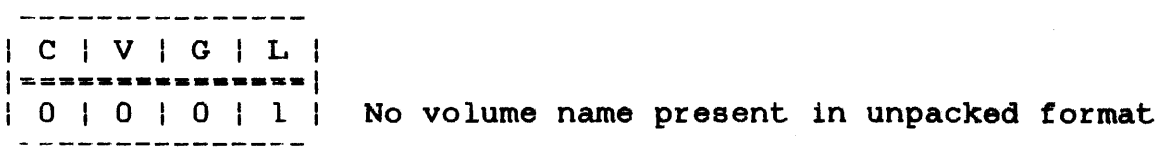
Unpacked fd:



Packed fd:



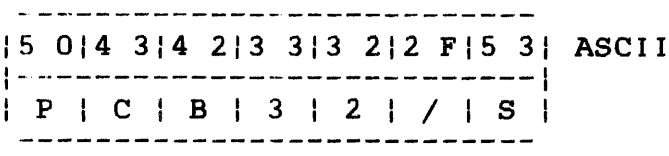
Condition code:



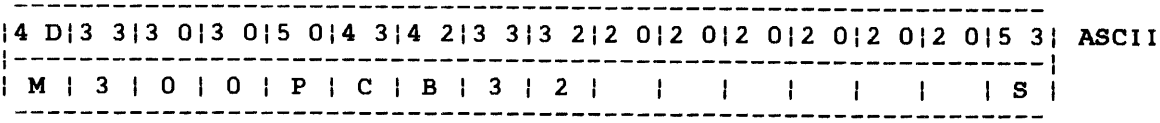
The example above shows a default volume option with an explicit request for an extension containing spaces.

Example 5:

Unpacked fd:



Packed fd:



Condition Code:

C	V	G	L	
1	0	0	1	No extention present in unpacked fd; no volume name present in unpacked fd

If a syntax error occurs, the scan of the unpacked fd terminates at the byte that caused the syntax error and the area receiving the packed fd is filled with indeterminate code. Check the CC to determine if a syntax error occurred.

3.16.1 SVC2 Code 16, Option X'00'

If option X'00' with no volume name is specified, the default is the user volume. The first byte of the unpacked fd (currently pointed to by the user-specified register) is the starting location of the pack fd operation.

The following examples use M67A as the default system volume.

Example 1:

Unpacked fd:

4 3	4 8	3 2	5 0	5 2	4 D	2 E	3 6	3 1	3 3	2 F	4 7	ASCII
C	H	2	P	R	M	.	6	1	3	/	G	

Packed fd:

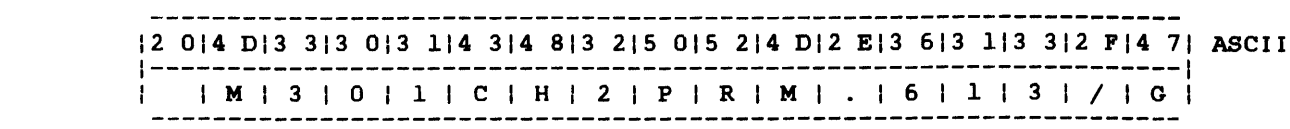
4 D	3 6	3 7	4 1	4 3	4 8	3 2	5 0	5 2	4 D	2 0	2 0	3 6	3 1	3 3	4 7	ASCII
M	6	7	A	C	H	2	P	R	M			6	1	3	G	

Condition Code:

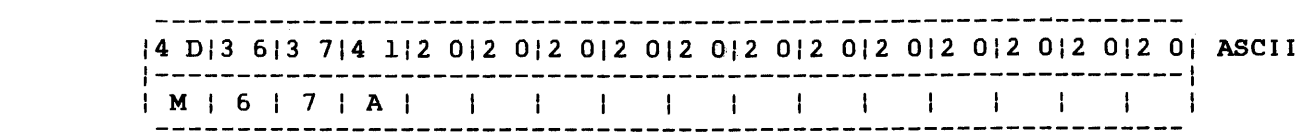
C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

Example 2:

Unpacked fd:



Packed fd:



Condition Code:

C	V	G	L
1	1	0	1

No extension present in unpacked fd; syntax error present in unpacked fd; no volume name present in unpacked fd

As shown in the above example, if the first character in the unpacked fd is not valid, processing stops. The system volume name is the default, and the filename, extension and class fields are modified to blanks.

3.16.2 SVC2 Code 16, Option X'40'

If option X'40' with no volume name is specified, the default user volume and all preceding spaces are ignored. All spaces are ignored from the first byte of the unpacked fd (currently pointed to by the user-specified register) to the first character in the unpacked fd.

The following example uses M67A as the default user volume.

Example:

Unpacked fd:

20	43	48	32	50	52	4D	2E	36	31	33	2F	47	ASCII
	C	H	2	P	R	M	.	6	1	3	/	G	

Packed fd:

4D	36	37	41	43	48	32	50	52	4D	20	20	36	31	33	47	ASCII
M	6	7	A	C	H	2	P	R	M			6	1	3	G	

Condition Code:

C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

3.16.3 SVC2 Code 16, Option X'10'

If option X'10' with no volume name is specified, the default is the system volume. The first byte of the unpacked fd (currently pointed to by the user-specified register) is the starting location of the pack fd operation.

The following examples use M300 as the default volume.

Example 1:

Unpacked fd:

53	56	43	32	2E	31	36	2F	50	ASCII
S	V	C	2	.	1	6	/	P	

Packed fd:

```
 ---------------------------------------------------------------
|4 D|3 3|3 0|3 0|5 3|5 6|4 3|3 2|2 0|2 0|2 0|2 0|3 1|3 6|2 0|5 0| ASCII
|-------------------------------------------------------------|
| M | 3 | 0 | 0 | S | V | C | 2 |   |   |   |   | 1 | 6 |   | P |
 ---------------------------------------------------------------
```

Condition Code:

```
 ---------------
| C | V | G | L |
|===============|
| 0 | 0 | 0 | 1 |
 ---------------
```

No volume name present in unpacked fd

Example 2:

Unpacked fd:

```
 ---------------------------------------------------------------
|2 0|4 D|3 6|3 7|4 1|3 A|5 3|5 6|4 3|3 2|2 E|3 1|3 6|2 F|5 0| ASCII
|-------------------------------------------------------------|
|   | M | 6 | 7 | A | : | S | V | C | 2 | . | 1 | 6 | / | P |
 ---------------------------------------------------------------
```

Packed fd:

```
 ---------------------------------------------------------------
|4 D|3 3|3 0|3 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0| ASCII
|-------------------------------------------------------------|
| M | 3 | 0 | 0 |   |   |   |   |   |   |   |   |   |   |   |   |
 ---------------------------------------------------------------
```

Condition Code:

```
 ---------------
| C | V | G | L |
|===============|
| 1 | 1 | 0 | 1 |
 ---------------
```

No extension present in unpacked fd; syntax error present in the unpacked fd; no volume name present in the unpacked fd

As shown in this example, if the first character in the unpacked fd is not valid, processing stops. The system volume name is the default and the filename, extension and class fields are modified to blanks.

3.16.4 SVC2 Code 16, Option X'50'

If option X'50' with no volume name is specified, the default system volume and all preceding spaces are ignored. All spaces are ignored from the first byte of the unpacked fd (currently pointed to by the user-specified register) to the first character in the unpacked fd.

The following example uses M300 as the default system volume.

Example:

Unpacked fd:

2 0	2 0	2 0	5 4	5 3	2 E	4 3	5 3	5 3	2 F	4 7	ASCII
			T	S	.	C	S	S	/	G	

Packed fd:

4 D	3 3	3 0	3 0	5 4	5 3	2 0	2 0	2 0	2 0	2 0	2 0	4 3	5 3	5 3	4 7	ASCII
M	3	0	0	T	S							C	S	S	G	

Condition Code:

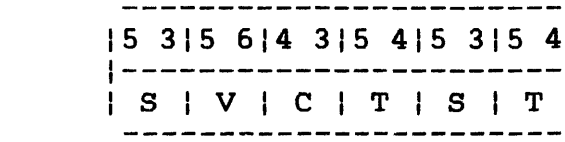
C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

3.16.5 SVC2 Code 16, Option X'20'

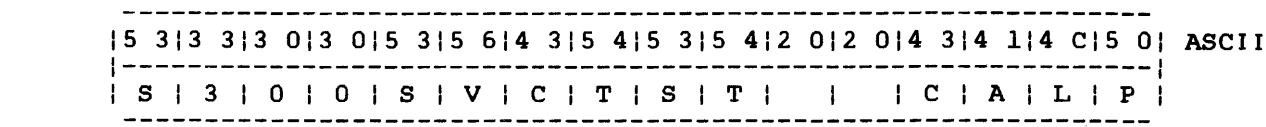
If option X'20' with no volume name is specified, the default is the spool volume. The first byte of the unpacked fd (currently pointed to by the user-specified register) is the starting location of the pack fd operation.

Example 1:

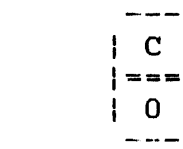
Unpacked fd:



Packed fd:



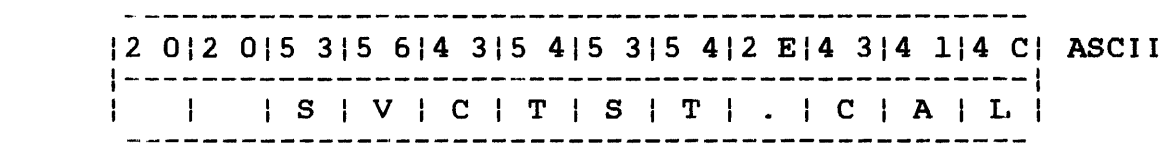
Condition Code:



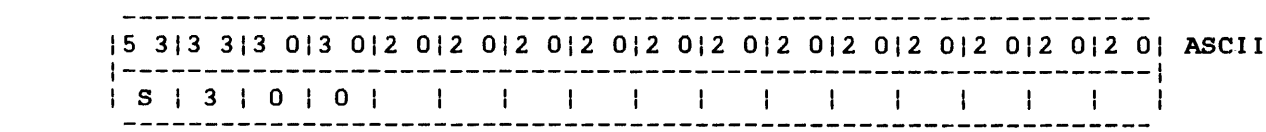
No volume name present in unpacked fd

Example 2:

Unpacked fd:



Packed fd:



Condition Code:

C	V	G	L	
1	1	0	1	No extension present in unpacked fd; syntax error in unpacked fd; no volume name in unpacked fd

As shown in this example, if no volume name is specified and the filename is preceded by at least one space, the spool volume is the default and the filename, extension and class fields are modified to blanks.

3.16.6 SVC2 Code 16, Option X'60'

If option X'60' with no volume name is specified, the default is the spool volume and all preceding spaces are ignored. All spaces are ignored from the first byte of the unpacked fd (currently pointed to by the user-specified register) to the first character in the unpacked fd. The default volume for the following examples is S300.

Example 1:

Unpacked fd:

4 9	4 C	5 4	5 3	5 4	2 E	5 4	5 3	4 B	2 F	5 3	ASCII
I	L	T	S	T	.	T	S	K	/	S	

Packed fd:

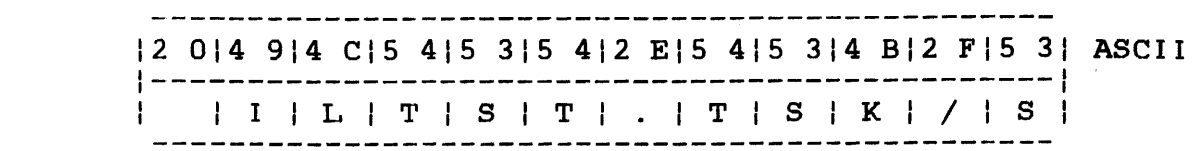
5 3	3 3	3 0	3 0	4 9	4 C	5 4	5 3	5 4	2 0	2 0	2 0	5 4	5 3	4 B	5 3	ASCII
S	3	0	0	I	L	T	S	T				T	S	K	S	

Condition Code:

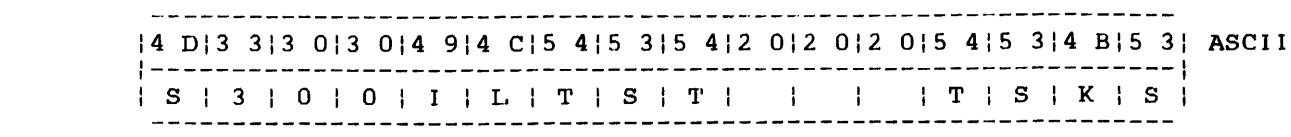
C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

Example 2:

Unpacked fd:



Packed fd:



Condition Code:

C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

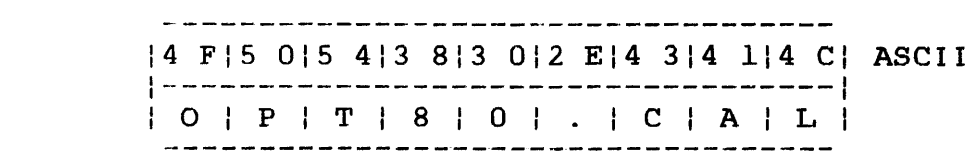
If no volume name is specified and the filename is preceded by at least one space, all preceding spaces are ignored, and the default is the spool volume. The spool volume name and remaining fd are packed.

3.16.7 SVC2 Code 16, Option X'80'

If option X'80' with no volume name is specified, the contents of the volume name field before executing the pack fd operation is used as the volume name. The first byte of the unpacked fd (currently pointed to by the user-specified register) is the starting location of the pack fd operation.

Example 1:

Unpacked fd:



Packed fd location contents before pack fd operation:

3	0	3	0	3	0	3	0	5	7	4	8	4	1	5	4	4	5	5	6	4	5	5	2	4	5	4	C	5	3	4	5	ASCII
0		0		0		0		W		H		A		T		E		V		E		R		E		L		S		E		

Packed fd after pack fd operation:

3	0	3	0	3	0	3	0	4	F	5	0	5	4	3	8	3	0	2	0	2	0	2	0	4	3	4	1	4	C	5	0	ASCII
0		0		0		0		O		P		T		8		0								C		A		L		P		

Condition Code:

C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

If a volume name is specified and is preceded by at least one space, that volume name is ignored and the contents remaining in the volume name field before executing the pack fd operation are used as the volume name. The filename, extension and class fields are modified to blanks as shown in Example 2.

Example 2:

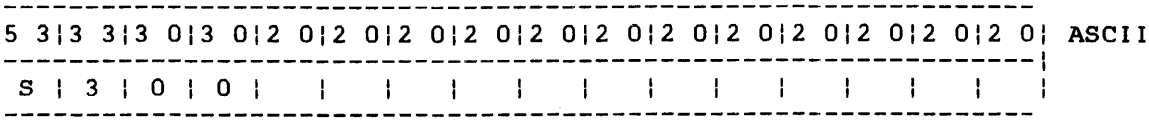
Unpacked fd:

2	0	2	0	4	D	3	1	3	0	4	1	3	A	4	F	5	0	5	4	3	8	3	0	2	E	4	3	4	1	4	C	ASCII
				M		1		0		A		:		O		P		T		8		0		.		C		A		L		

Packed fd location contents before pack fd operation:

5	3	3	3	3	0	3	0	5	7	4	8	4	1	5	4	4	5	5	6	4	5	5	2	4	5	4	C	5	3	4	5	ASCII
S		3		0		0		W		H		A		T		E		V		E		R		E		L		S		E		

Packed fd after pack fd operation:



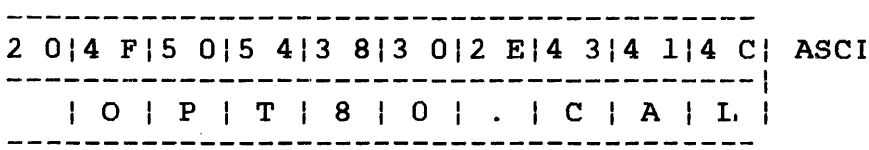
Condition Code:

C	V	G	L	
1	1	0	1	No extension present in unpacked fd; syntax error present in unpacked fd; no volume name present in unpacked fd

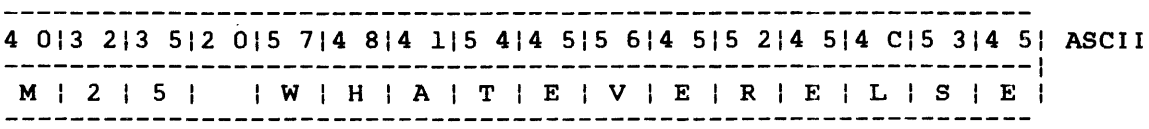
If no volume name is specified and the filename is preceded by at least one space, the contents remaining in the volume name field before executing the pack fd operation are used as the volume name. The filename, extension and class fields are modified to blanks as shown in Example 3.

Example 3:

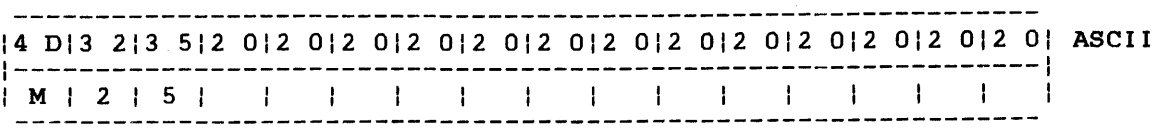
Unpacked fd:



Packed fd location contents before pack fd operation:



Packed fd after pack fd operation:



Condition Code:

C	V	G	L
1	1	0	1

No extension present in unpacked fd;
syntax error present in unpacked fd;
no volume name present in unpacked fd

3.16.8 SVC2 Code 16, Option X'C0'

If option X'C0' with no volume name is specified, the contents of the volume name field before executing the pack fd operation are used as the volume name and all preceding spaces are ignored. All spaces are ignored from the first byte of the unpacked fd (currently pointed to by the user-specified register) to the first character in the unpacked fd.

Example 1:

Unpacked fd:

5 0	5 2	4 D	2 E	3 6	3 1	3 3	2 F	5 0	ASCII
P	R	M	.	6	1	3	/	P	

Packed fd contents before pack fd operation:

4 4	5 3	4 3	3 3	5 7	4 8	4 1	5 4	4 5	5 6	4 5	5 2	4 5	4 C	5 3	4 5	ASCII
D	S	C	3	W	H	A	T	E	V	E	R	E	L	S	E	

Packed fd after pack fd operation:

4 4	5 3	4 3	3 3	5 0	5 2	4 D	2 0	2 0	2 0	2 0	2 0	3 6	3 1	3 3	5 0	ASCII
D	S	C	3	P	R	M						6	1	3	P	

Condition Code:

C	V	G	L	
0	0	0	1	No volume name present in unpacked fd

If a volume name is specified and is preceded by at least one space, all preceding spaces are ignored and that volume name and remaining fd are packed as shown in Example 2.

Example 2:

Unpacked fd:

2 0	2 0	4 D	3 6	3 7	4 1	3 A	5 0	5 2	4 D	2 E	3 6	3 1	3 3	2 F	5 0	ASCII
		M	6	7	A	:	P	R	M	.	6	1	3	/	P	

Packed fd:

4 D	3 6	3 7	4 1	5 0	5 2	4 D	2 0	2 0	2 0	2 0	2 0	3 6	3 1	3 3	5 0	ASCII
M	6	7	A	P	R	M						6	1	3	P	

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination

3.16.9 SVC2 Code 16, Options for Privileged Tasks

Only privileged u-tasks, e-tasks and privileged diagnostic tasks (d-tasks) are allowed to pack an fd so that the resulting packed fd has an account number in its file class/account number field. A u-task becomes privileged if the account privileges task option (ACPRIVILEGE) is specified when the task is link-edited.

ACPRIVILEGE allows u-tasks to access files by account number rather than file class. The range of account numbers available to the task is 0 through 65,535, excluding 255. To access files on account 255, the bare disk I/O task option (DISC) must also be specified when the task is link-edited.

E-tasks always have account privileges.

CAUTION

IF THE OS/32 TASK LOADER HAS THE E-TASK LOAD OPTION DISABLED, ALL U-TASKS WILL BE DENIED ACCOUNT PRIVILEGES REGARDLESS OF THE TASK OPTIONS SPECIFIED BY LINK.

The following SVC2 code 16 options are used by an e-task, privileged d-task or privileged u-task to produce a packed fd that has an account number in its file class/account number field:

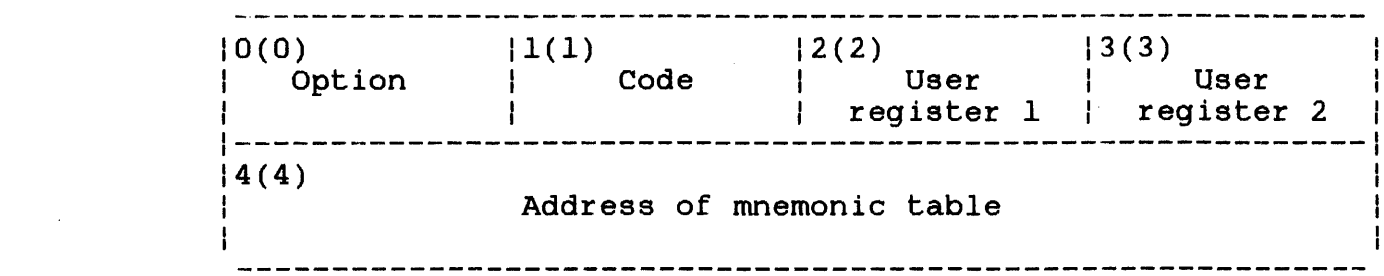
OPTION	MEANING
X'08'	Default volume is the user volume.
X'48'	Default volume is the user volume; skip leading spaces.
X'18'	Default volume is the system volume.
X'58'	Default volume is the system volume; skip leading spaces.
X'28'	Default volume is the spool volume.
X'68'	Default volume is the spool volume; skip leading spaces.
X'88'	There is no default volume.
X'C8'	There is no default volume; skip leading spaces.

When a privileged task uses one of the above options to pack an fd that has either an account number or file class in its file class/account number field, SVC2 code 16 returns an account number to the resulting packed fd and sets the G bit in the CC.

If neither an account number nor a file class is specified in the unpacked fd, the file is packed with account number 0 (if the task is running at the system console) or the user's private account number (if the task is running under MTM).

3.17 SVC2 CODE 17: SCAN MNEMONIC TABLE

SVC2 code 17 compares a user-specified mnemonic character string to a table of previously defined mnemonic strings. If a match is found, the user-specified mnemonic character string is accepted as a valid mnemonic. The SVC2 code 17 parameter block is shown in Figure 3-22.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,17
          DB    user register 1, user register 2
          DCF   A(mnemonic table)
```

Figure 3-22 SVC2 Code 17 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 17 to indicate SVC2 code 17.
User register 1	is a 1-byte field that must contain a user-specified register number. This register should contain the starting address of the buffer with the user-specified mnemonic character string. After executing SVC2 code 17, this register contains the address of the byte following the user-specified mnemonic string or the unchanged starting address.
User register 2	is a 1-byte field that must contain a user-specified register number. This register receives a decimal number from -1 to 2,147,483,647 (2^{31} -1) corresponding to the position of the mnemonic within the table that matches the user-specified mnemonic character string. If no match is found, this register receives a value of -1. The first position in the mnemonic table corresponds to a value of 0.
Address of mnemonic table	is a 4-byte field that must contain the starting address of the mnemonic table. This must be defined before executing SVC2 code 17.

The user-specified mnemonic character string can be any length but can only contain the following characters:

- A through Z (upper-case)
- a through z (lower-case)
- 0 through 9 (can be used only after the first byte of the mnemonic)
- Special characters (can be used only as the first byte of the mnemonic).

All lower-case characters that appear in the user-specified mnemonic character string are accepted as their upper-case equivalent.

3.17.1 Building a Mnemonic Table

The mnemonic table to be used in SVC2 code 17 must be defined in a standard format. The mnemonics entered in the table can be any length but can contain only certain legal characters:

- A through Z (upper-case)
- 0 through 9 (can be used only after the first byte of the mnemonic)
- Special characters (can be used only as the first byte of the mnemonic)

The characters for each mnemonic in the table must be in contiguous order, beginning with the first character and ending with the termination indicator, X'00'. Every mnemonic entered in the table has a minimum abbreviation. Each character required for the minimum abbreviation must have an X'80' added to the character when the mnemonic is defined. The mnemonic table must be terminated by an X'00' after the last mnemonic entry. See the example below.

Example:

```
TABLE    EQU   *
         DB    C'G'+X'80',C'ET',X'00'
         DB    C'R'+X'80',C'E'+X'80',C'W'+X'80',C'IND',X'00'
         .
         .
         .
         DB    C'S'+X'80',C'T'+X'80',C'ART',X'00'
         DB    X'00'
```

When the table is assembled, a logical OR operation is performed on X'80' and the character associated with it. This sets bit 0 of each character on which the OR operation was performed to 1. A bit setting of 1 indicates that it is a required character; a bit setting of 0 indicates that it is not a required character.

3.17.2 Executing SVC2 Code 17

When executing this call, the user-specified mnemonic character string is compared to each entry in the mnemonic table until a match is found. Once a match is found, the address of the byte following the user-specified mnemonic character string is stored in the user register specified by the user register 1 field of the parameter block.

The mnemonic's position (decimal number) within the table that matched the user-specified mnemonic character string is stored in the user register specified by the user register 2 field of the parameter block. After executing SVC2 code 17, the CC is set.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	User-specified mnemonic character string does not match any mnemonic in the table

Example:

```
            LA      3,STRING
            SVC     2,SCAN
            SVC     2,PAUSE
            .
            .
            .
            ALIGN 4
SCAN        DB      0,17
            DB      3,5
            DC      A(TABLE)
STRING      DB      C'map'
            ALIGN 4
PAUSE       DB      0,1
TABLE       EQU     *
            DB      C'A'+X'80',C'L'+X'80',C'LOCATE',X'00'
            DB      C'M'+X'80',C'A'+X'80',C'P'+X'80',X'00
            DB      C'T'+X'80',C'YPE',X'00'
            DB      X'00'
```

User-specified mnemonic string before and after execution of SVC2 code 17:

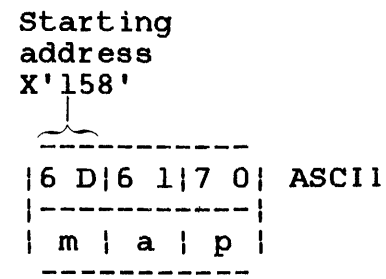


Table (after assembly) before and after execution of SVC2 code 17:

C 1	C C	4 C	4 F	4 3	4 1	5 4	4 5	0 0	C D	C 1	D 0	0 0
A	L	L	O	C	A	T	E	0 0	M	A	P	0 0
D 4	5 9	5 0	4 5	0 0	0 0							
T	Y	P	E	0 0	0 0							

User register 1 before execution of SVC2 code 17:

0 0	0 0	0 1	5 8	Hex

User register 1 after execution of SVC2 code 17:

0 0	0 0	0 1	5 B	Hex

User register 2 after execution of SVC2 code 17:

0 0	0 0	0 0	0 1	Hex

Condition Code:

C	V	G	L
0	0	0	0

If the user-specified mnemonic character string is compared to each entry in the mnemonic table and no match is found, the starting address of the buffer containing the user-specified mnemonic character string remains unchanged in user register 1. A decimal value of -1 is stored in user register 2, and the CC is set to 4 (V bit set).

Example:

```
            LA      3,STRING
            SVC     2,SCAN
            SVC     2,PAUSE
            .
            .
            .
            ALIGN 4
SCAN        DB      0,17
            DB      3,5
            DB      A(TABLE)
STRING      DC      C'AS'
            ALIGN 4
PAUSE       DB      0,1
TABLE       EQU     *
            DB      C'A'+X'80',C'L'+X'80',C'LOCATE',X'00'
            DB      C'M'+X'80',C'A'+X'80',C'P'+X'80',X'00'
            DB      C'T'+X'80',C'YPE',X'0000'
```

User-specified mnemonic string before and after execution of SVC2 code 17:

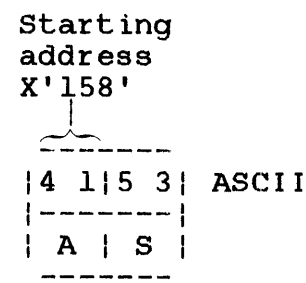


Table (after assembly) before and after execution of SVC2 code 17:

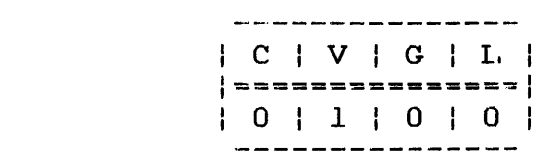
C 1	C C	4 C	4 F	4 3	4 1	5 4	4 5	0 0	C D	C 1	D 0	0 0			
A	L	L	O	C	A	T	E	0 0	M	A	P	0 0			
D 4	5 9	5 0	4 5	0 0	0 0										
T	Y	P	E	0 0	0 0										

User register 1 before and after execution of SVC2 code 17:

0 0	0 0	0 1	5 8	Hex

User register 2 after execution of SVC2 code 17:

Condition Code:



If a nonalphanumeric character follows the first character in a user-specified mnemonic string, the nonalphanumeric character is treated as the end of the mnemonic. The address of the nonalphanumeric character is returned to user register 1.

Example:

```
            LA      3,STRING
            SVC     2,SCAN
            SVC     2,PAUSE
            .
            .
            .
            ALIGN 4
SCAN        DB      0,17
            DB      3,5
            DC      A(TABLE)
STRING      DB      C'TY&E'
            ALIGN 4
PAUSE       DB      0,1
            ALIGN 4
TABLE       EQU     *
            DB      C'A'+X'80',C'L'+X'80',C'LOCATE',X'00'
            DB      C'M'+X'80',C'A'+X'80',C'P'+X'80',X'00'
            DB      C'T'+X'80',C'YPE',X'0000'
```

User-specified string before and after execution of SVC2 code 17:

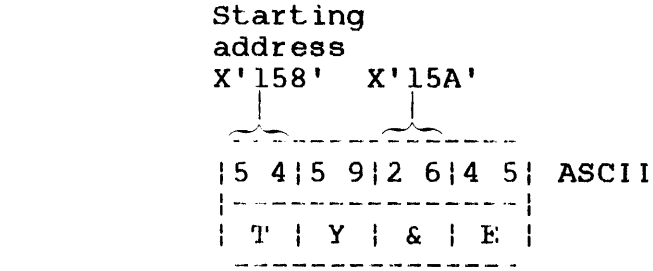


Table (after assembly) before and after execution of SVC2 code 17:

C 1	C C	4 C	4 F	4 3	4 1	5 4	4 5	0 0	C D	C 1	D 0	0 0			
A	L	L	O	C	A	T	E	0 0	M	A	P	0 0			
D 4	5 9	5 0	4 5	0 0	0 0										
T	Y	P	E	0 0	0 0										

User register 1 before execution of SVC2 code 17:

0 0	0 0	0 1	5 8	Hex

User register 1 after execution of SVC2 code 17:

0 0	0 0	0 1	5 A	Hex

User register 2 after execution of SVC2 code 1:

0 0	0 0	0 0	0 2	Hex

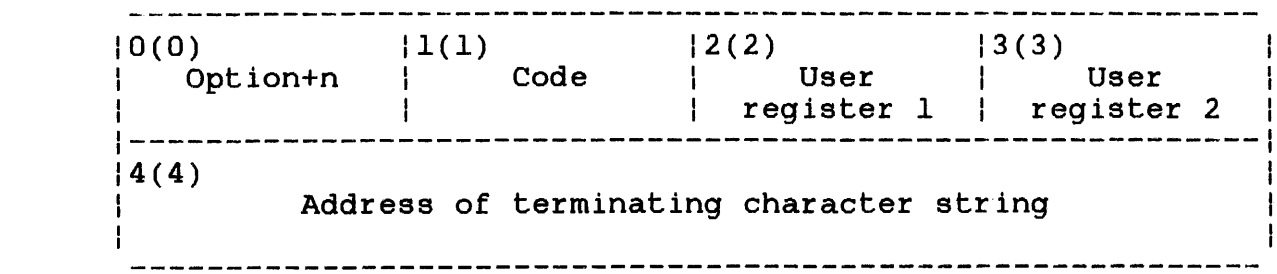
Condition Code:

C	V	G	L
0	0	0	0

In the above example, the user-specified mnemonic "TY&E" is treated as "TY". The address of the byte following the user-specified mnemonic string is then X'15A', which is returned to user register 1. A decimal value of 2 is stored in user register 2, and the CC is set to 0.

3.18 SVC2 CODE 18: MOVE ASCII CHARACTERS

SVC2 code 18 moves a specified number of ASCII characters from a sending buffer to a receiving buffer in memory. The SVC2 code 18 parameter block is shown in Figure 3-23.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option+n,18
          DB    user register 1, user register 2
          DCF   A(terminating character string)
```

Figure 3-23 SVC2 Code 18 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option+n — is a 1-byte field that must contain the addition of the hexadecimal number specified for the option and the decimal number specified as n.

- n is a decimal number ranging from 0 to 127 indicating an explicit number of bytes in the ASCII character string that are to be moved to the receiving buffer in memory.

- Option X'00'+n means no terminating character string is used.
- Option X'80'+n means a terminating character string is used.

Code — is a 1-byte field that must contain the decimal number 18 to indicate SVC2 code 18.

User register 1 — is a 1-byte field that must contain a user-specified register number. This register must contain the starting address of the buffer containing the user-specified ASCII character string to be moved. After executing SVC2 code 18, this register contains the address of the byte in the sending buffer that follows the last moved character.

User register 2 — is a 1-byte field that must contain a user-specified register number. This register must contain the starting address of the buffer that receives the user-specified number of ASCII characters being sent. This buffer must be located in a task-writable segment. After executing SVC2 code 18, this register contains the address of the byte in the receiving buffer that follows the last character received.

Address of terminating character — is a 4-byte field that must contain the starting address of the user-specified string of terminating characters. Each character of this string can be used to indicate the end of the ASCII character string to be moved. This field is only used when option X'80' is specified.

When SVC2 code 18 is executed, the specified number of ASCII characters are moved to the receiving buffer. The starting addresses of the sending and receiving buffers located in the user-specified registers are changed to the address following the last byte sent in the sending buffer and the last byte received in the receiving buffer. The CC is also set after executing SVC2 code 18. The possible CC settings follow.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	No terminating character found in the ASCII character string

3.18.1 SVC2 Code 18, Option X'00'+n

If option X'00'+n is used with a user-specified decimal number, that decimal number determines the number of bytes moved from the ASCII character string to the receiving buffer. After executing SVC2 code 18, user register 1 contains the address of the byte in the sending buffer that follows the user-specified number of ASCII characters that were moved. User register 2 contains the address of the byte in the receiving buffer that follows the user-specified number of ASCII characters just received. The CC is set to 0.

Example:

```
            LA     1,ASTRING
            LA     2,RECBUF
            SVC    2,MOVECHAR
            .
            .
            .
            ALIGN 4
MOVECHAR    DB     X'00'+17,18
            DB     1,2
            DS     4
ASTRING     DB     C'FLORIDA***VERMONT'
RECBUF      DS     17
```

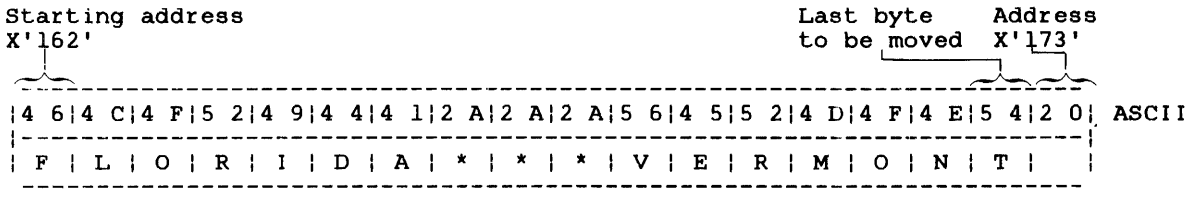
User register 1 before execution of SVC2 code 18:

0 0	0 0	0 1	6 2	Hex

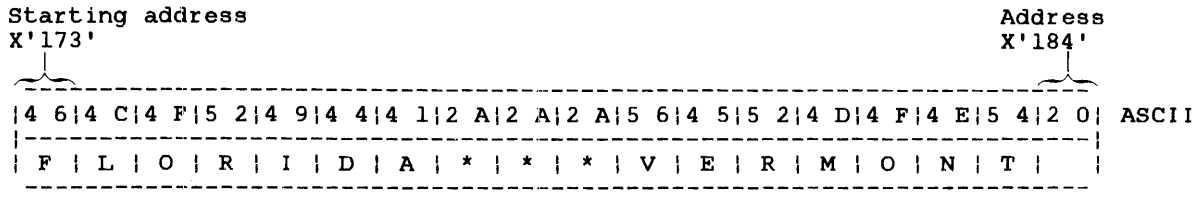
User register 2 before execution of SVC2 code 18:

0 0	0 0	0 1	7 3	Hex

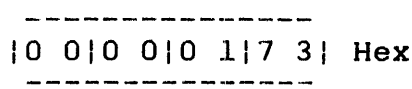
ASCII character string before and after execution of SVC2 code 18:



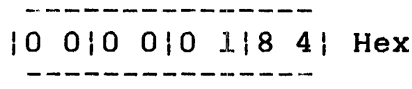
Receiving buffer after execution of SVC2 code 18:



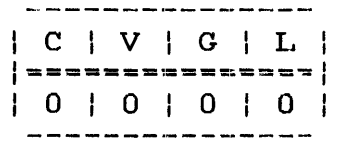
User register 1 after execution of SVC2 code 18:



User register 2 after execution of SVC2 code 18:



Condition Code:



3.18.2 SVC2 Code 18, Option X'80'+n

If option X'80'+n is specified, each character in the ASCII string is compared to each character in the terminating string before it is moved. A match indicates that the end of the ASCII character string to be moved was reached and the decimal number n, which specifies the number of characters to be moved, is ignored. The character or characters in the ASCII string that match the character or characters in the terminating string are not moved, and the SVC terminates. The CC is set to 0.

The string of terminating characters can be any length and can contain any character but must be specified by the user as follows.

Format:

```
label DB m,C'xxx...x'
```

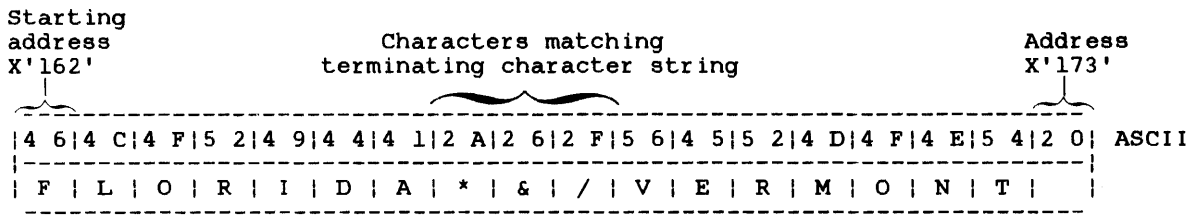
Parameters:

label	is the name of the terminating character string the user specifies.
DB	is the operation code, define byte.
m	is a decimal number indicating the number of characters in the terminating character string.
C'xxx...x'	is a character string indicating that the data enclosed in the single quotation marks are characters.

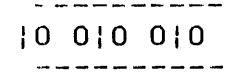
Example:

```
            LA      3,ASTRING
            LA      5,RECBUF
            SVC     2,MOVECHAR
            SVC     2,PAUSE
            .
            .
            .
            ALIGN 4
PAUSE       DB      0,1
            ALIGN 4
MOVECHAR    DB      X'80'+17,18
            DB      3,5
            DC      A(TSTRING)
TSTRING     DB      3,C'/&*'
ASTRING     DB      C'FLORIDA*&/VERMONT'
RECBUF      DB      17
```

ASCII character string before and after execution of SVC2 code 18:



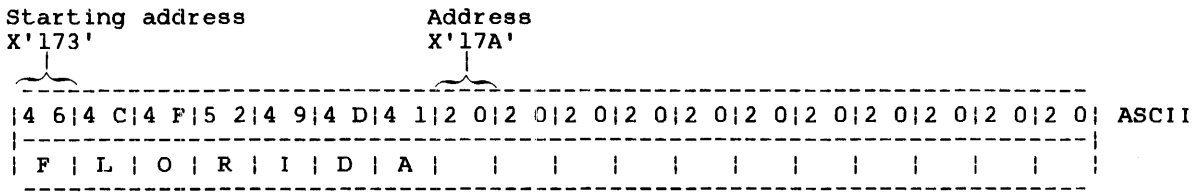
User register 1 before execution of SVC2 code 18:



User register 2 before execution of SVC2 code 18:

0 0	0 0	0 1	7 3	Hex

Receiving buffer after execution of SVC2 code 18:



User register 1 after execution of SVC2 code 18:

0 0	0 0	0 1	6 9	Hex

User register 2 after execution of SVC2 code 18:

0 0	0 0	0 1	7 A	Hex

Terminating character string before and after execution of SVC2 code 18:

2 A	2 6	2 F	ASCII
/	&	*	

Condition Code:

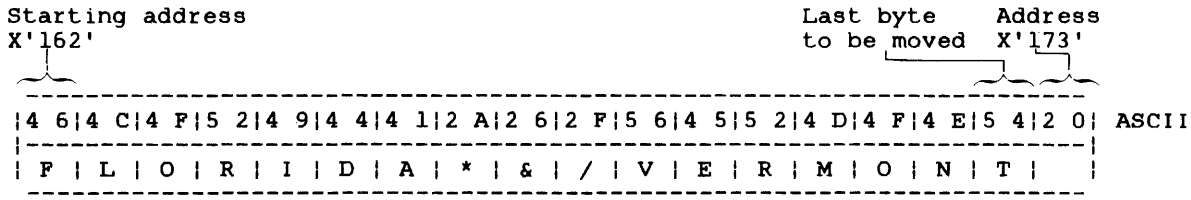
C	V	G	L
0	0	0	0

If option X'80' is specified and the ASCII character string does not contain any characters that match any terminating character, the decimal number specified as m determines the number of bytes to be moved. The CC is set to 4 (V bit set).

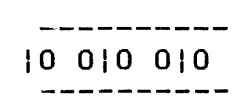
Example:

```
            LA      3,ASTRING
            LA      5,RECBUF
            SVC     2,MOVECHAR
            SVC     2,PAUSE
            .
            .
            .
            ALIGN 4
PAUSE       DB      0,1
            ALIGN 4
MOVECHAR DB         X'80' + 17,18
            DB      3,5
            DC      A(TSTRING)
TSTRING  DB         3,C',$:'
ASTRING  DB         C'FLORIDA*&/VERMONT'
RECBUF   DS         17
```

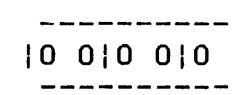
ASCII character string before and after execution of SVC2 code 18:



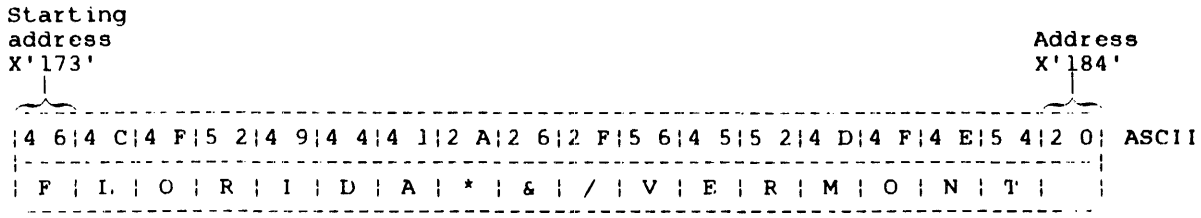
User register 1 before execution of SVC2 code 18:



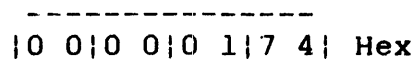
User register 2 before execution of SVC2 code 18:



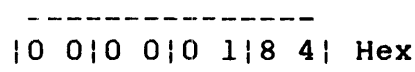
Receiving buffer after execution of SVC2 code 18:



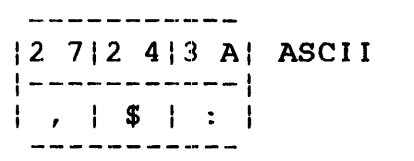
User register 1 after execution of SVC2 code 18:



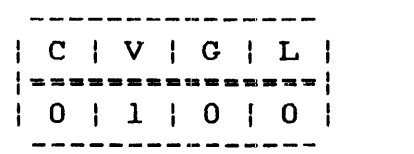
User register 2 after execution of SVC2 code 18:



Terminating character string:



Condition Code:

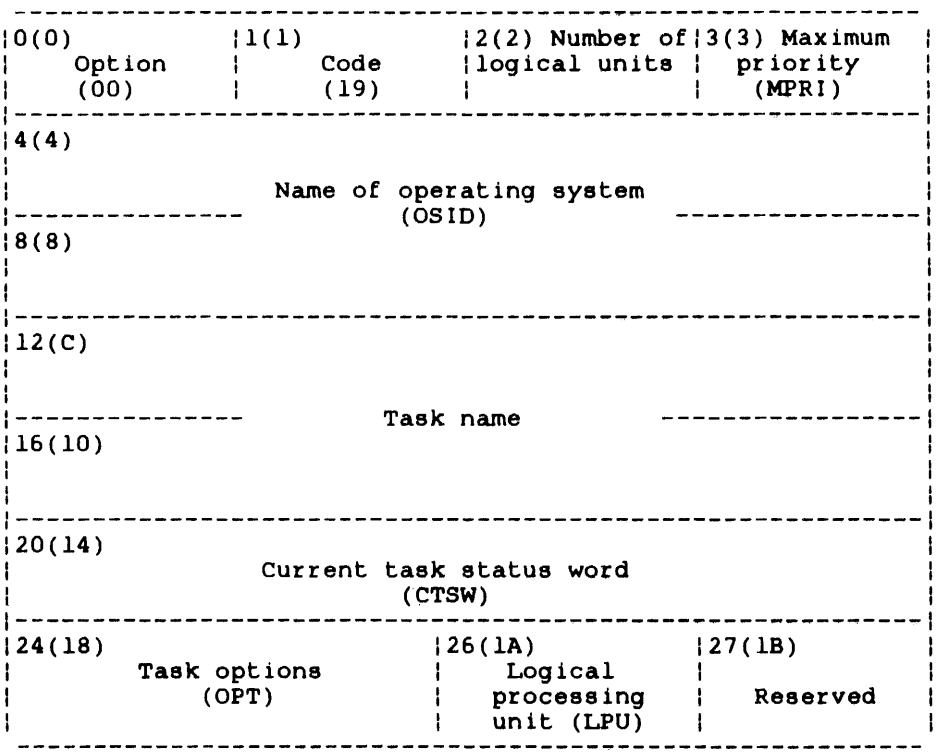


3.19 SVC2 CODE 19: PEEK

SVC2 code 19 provides five parameter block options that can be used to obtain and store task-related information. Each parameter block option obtains a different set of information from the system pointer table (SPT) and the TCB. Figures 3-21 through 3-25 illustrate the five peek parameter block option formats.

3.19.1 Parameter Block for Option X'00'

If SVC2 code 19 is executed with option X'00' specified in the parameter block option field, use the parameter block format in Figure 3-24. This option is used to obtain task information.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'00',19
          DS    25
          DB    0
```

Figure 3-24 SVC2 Code 19 Parameter Block Format and Coding for Option X'00'

This parameter block must be 28 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'00'.
Code	is a 1-byte field that must contain the decimal number 19 to indicate SVC2 code 19.
Number of logical units	is a 1-byte field that receives from the TCB the maximum logical unit number that can be assigned to a task. This hexadecimal number ranges from 0 to 254 (X'FE').
Maximum priority (MPRI)	is a 1-byte field that receives from the TCB the highest priority number at which the assigned task can execute. This hexadecimal number ranges from 10 (X'04') to 249 (X'F9').
Name of operating system (OSID)	is an 8-byte field that receives from the SPT the operating system name in ASCII.
Task name	is an 8-byte field that receives from the TCB the name of the task in ASCII.
Current task status word (CTSW)	is a 4-byte field that receives from the TCB the hexadecimal number representing bits 0 through 31 of the CTSW.
Task options (OPT)	is a 2-byte field that receives the hexadecimal number representing bits 16 through 31 of the option field in the TCB. Bits 0 through 15 are accessible through option X'03' of SVC2 code 19. Table 3-4 lists task options.
Logical processing unit (LPU)	is a 1-byte field that receives the decimal number of the task's current LPU assignment from the TCB. The value of this number ranges from X'00' to X'09'; X'00' indicates the central processing unit (CPU).
Reserved	is a reserved 1-byte field that must contain zeros.

TABLE 3-4 TASK OPTIONS FROM THE TCB

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
0	D-task (Y'8000 0000')	0 = task determined by bit 16 1 = task is a d-task
1	Auxiliary processing unit (APU) only (Y'4000 0000')	0 = task can run on CPU or APU 1 = task cannot run on CPU
2	APU mapping (Y'2000 0000')	0 = no APU mapping allowed 1 = task can perform APU mapping functions
3	APU control (Y'1000 0000')	0 = no APU control allowed 1 = task can perform APU mapping functions
4	Dynamic priority scheduling (Y'0800 0000')	0 = dynamic priority scheduling disabled 1 = dynamic priority scheduling enabled
5	Prompts (Y'0400 0000')	0 = MTM prompts disabled 1 = MTM prompts enabled
6	Vertical forms control (VFC) (Y'0200 0000')	0 = except where specified, all I/O interpreted without VFC 1 = all I/O interpreted with VFC
7	Extended SVC1 parameter block (Y'0100 0000')	0 = SVC1 extended parameter block not used (excludes communications I/O) 1 = extended SVC1 parameter block used
8	Task event service (Y'0080 0000')	0 = new task status word (TSW) for task event service 1 = no new TSW for task event service
9	Task event registers save (Y'0040 0000')	0 = all register contents saved and restored 1 = only contents of registers that contain task event data are saved and restored

TABLE 3-4 TASK OPTIONS FROM THE TCB (Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
10	Task event register save (Y'0020 0000')	0 = task event register not saved 1 = task event register saved
11	System group (Y'0010 0000')	0 = not in system group 1 = in system group
12	Console I/O intercept (Y'0008 0000')	0 = no console I/O interrupt 1 = console I/O interrupt enable (MTM)
13	Universal status report (Y'0004 0000')	0 = universal task status reports not allowed 1 = universal task status reports allowed
14	E-task load (Y'0002 0000')	0 = allow e-task load 1 = prevent e-task load
15	Queued I/O (Y'0001 0000')	0 = queued I/O not purged on error 1 = queued I/O purged on error
16	E-task (Y'0000 8000')	0 = task is a u-task 1 = task is an e-task
17	Arithmetic fault (Y'0000 4000')	0 = task abnormally terminates on arithmetic fault 1 = task continues execution on arithmetic fault
18	Single precision floating point (SPFP) (Y'0000 2000')	0 = task does not support SPFP 1 = task supports SPFP
19	Memory resident (Y'0000 1000')	0 = task is nonresident 1 = task is resident in memory
20	SVC6 control functions (Y'0000 0800')	0 = task can execute all SVC6 control functions 1 = all SVC6 control functions are prevented
21	SVC6 communication functions (Y'0000 0400')	0 = task can execute all SVC6 communication functions 1 = all SVC6 communication functions are prevented

TABLE 3-4 TASK OPTIONS FROM THE TCB (Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
22	Illegal SVC6 (Y'0000 0200')	0 = task abnormally terminates on an illegal SVC6 1 = task continues execution on an illegal SVC6
23	Double precision floating point (DPFP) (Y'0000 0100')	0 = task does not support DPFP 1 = task supports DPFP
24	Rollable (Y'0000 0080')	0 = task is not rollable 1 = task is rollable
25	Overlays (Y'0000 0040')	0 = task does not support the use of overlays 1 = task supports the use of overlays
26	Accounting Facility (Y'0000 0020')	0 = disable Accounting Facility 1 = enable Accounting Facility
27	Intercept calls (Y'0000 0010')	0 = task cannot issue intercept calls 1 = task can issue intercept calls
28	Account number privileges (Y'0000 0008')	0 = task does not have file account number privileges 1 = task has file account number privileges
29	Bare disk I/O privilege (Y'0000 0004')	0 = task cannot directly assign to a disk device 1 = task can directly assign to a disk device for bare disk I/O (see Chapter 8)
30	Universal communications task (Y'0000 0002')	0 = task is not a universal task 1 = task is a universal task
31	E-task keys (Y'0000 0001')	0 = no keys are checked on an assign for an e-task 1 = keys are checked on an assign for an e-task

Example:

```
            SVC2,PEEK
            SVC2,PAUSE
            .
            .
            .
            ALIGN 4
PEEK        DB X'00',19
            DS 25
            DB 0
            ALIGN 4
PAUSE       DB  0,1
```

Parameter block before execution of SVC2 code 19:

00	19	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00

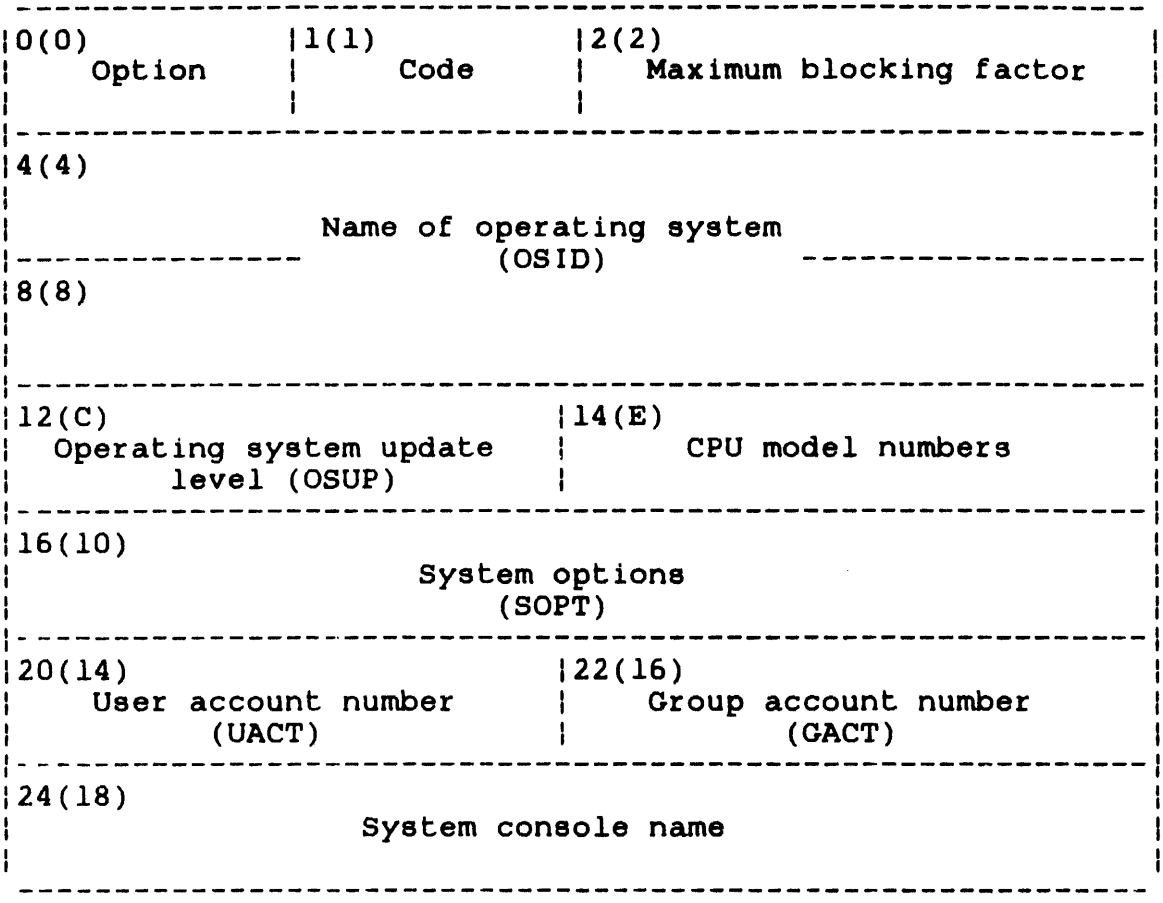
Parameter block after execution of SVC2 code 19:

00	19	0F	81
0	S	3	2
M	T	0	6
M	A	R	
00	00	00	00
0C	84	00	00

0C84 = SVC6 control call prevented.
SVC6 communication call prevented.
Task is rollable.
SVC6 load of e-task prevented.

3.19.2 Parameter Block for Option X'01'

To execute SVC2 code 19 with option X'01' specified in the parameter block option field, use the parameter block format in Figure 3-25.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'01',19
          DS    26
```

Figure 3-25 SVC2 Code 19 Parameter Block Format and Coding for Option X'01'

This parameter block must be 28 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'01'.
Code	is a 1-byte field that must contain the decimal number 19 to indicate SVC2 code 19.
Maximum blocking factor	is a 2-byte field that receives a number ranging from X'01' to X'FF'. This number indicates the maximum number of 256-byte segments that can be specified in an ALLOCATE command or an SVC7 for the data block size of indexed files, and for the indexed block size for indexed nonbuffered indexed and extendable contiguous files. This blocking factor must be set at sysgen. See the System Generation/32 (Sysgen/32) Reference Manual.
Name of operating system (OSID)	is an 8-byte field that receives from the SPT the operating system name in ASCII.
Operating system update level (OSUP)	is a 2-byte field that receives from the SPT the current update level of the operating system in ASCII in the form nn.
CPU model numbers	is a 2-byte field that receives in hexadecimal from the SPT the model numbers of the CPU used in the system. They are:

- A Model 7/32 System has a value of X'0007'.
- A Model 8/32 System has a value of X'0008'.
- A Model 3200MPS System has a value of X'0C80'.
- A Model 3205 System has a value of X'0C85'.
- A Model 3210 System has a value of X'0C8A'.
- A Model 3220 System has a value of X'0C94'.
- A Model 3230 System has a value of X'0C9E'.
- A Model 3240 System has a value of X'0CA8'.
- A Model 3250 System has a value of X'0CB2'.

System options (SOPT)	is a 4-byte field that receives the hexadecimal value of bits 0 through 31 of the options field in the SPT. Table 3-5 lists system options.
User account number (UACT)	is a 2-byte field that receives the user account number from the TCB. This hexadecimal number is right-justified with zeros filling the left-most portion.
Group account number (GACT)	is a 2-byte field that receives the group account number from the TCB. This hexadecimal number is right-justified with zeros filling the left-most portion.
System console name	is a 4-byte field that receives the name of the device that is acting as the system console.

TABLE 3-5 SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
0	Single precision floating point (SPFP) (Y'80000000')	0 = system does not support SPFP 1 = system does support SPFP
1	Form date is displayed (Y'4000 0000')	0 = date is displayed in the form mm/dd/yy 1 = date is displayed in the form dd/mm/yy
2	Time display (Y'2000 0000')	0 = time is displayed on output device specified by the user 1 = time displayed on panel
3	DPFP (Y'1000 0000')	0 = system does not support DPFP 1 = system does support DPFP
4	Writable control store (WCS) (Y'0800 0000')	0 = system does not support WCS 1 = system does support WCS

TABLE 3-5 SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE (Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
5	Address alignment error checking (Y'0400 0000')	0 = hardware does not support address alignment error checking 1 = hardware supports address alignment error checking
6	Direct access (Y'0200 0000')	0 = system does not support direct access 1 = system supports direct access
7	ITAM (Y'0100 0000')	0 = system does not support communications 1 = system supports communications
8	Spool (Y'0080 0000')	0 = system does not support spooling 1 = system supports spooling
9	Roll (Y'0040 0000')	0 = system does not support roll-in, roll-out 1 = system supports roll-in, roll-out
10	Temporary files (Y'0020 0000')	0 = system does not support temporary files 1 = system supports temporary files
11	Multiple register sets (Y'0010 0000')	0 = system does not support multiple register sets 1 = system supports multiple register sets
12	Universal reporting (Y'0008 0000')	0 = intertask reporting between universal tasks off 1 = intertask reporting between universal tasks on
13	General error recording (Y'0004 0000')	0 = general error recording off 1 = general error recording on
14	Memory error recording (Y'0002 0000')	0 = memory error recording off 1 = memory error recording on

TABLE 3-5 SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE (Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
15	Reserved	0 = reserved for future use
16	Load real address (Y'0000 8000')	0 = load real address not supported 1 = load real address supported
17	Memory diagnostics (Y'0000 4000')	0 = memory diagnostics supported 1 = memory diagnostics not supported
18	Processor model (Y'0000 2000')	0 = Model 7/32, 8/32 Processors 1 = Model 3205, 3210, 3220, 3230, 3240, 3250, 3200MPS System Processors
19	Memory address translator (MAT) hardware (Y'0000 1000')	1 = system has MAT hardware 0 = system does not have MAT hardware
20	SPFP traps functions (Y'0000 0800')	1 = SPFP software traps present control functions 0 = SPFP software traps not present
21	DPFP traps (Y'0000 0400')	1 = DPFP software traps present 0 = DPFP software traps not present
31	System debug mode (Y'0000 0001')	0 = normal operation mode 1 = system debug mode

Example:

```
            SVC   2,PEEK
            SVC   2,PAUSE
            .
            .
            .
            ALIGN 4
PEEK        DB    X'01',19
            DS    26
            ALIGN 4
PAUSE       DB    0,1
```

Parameter block before execution of SVC2 code 19:

01	19	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00

Parameter block after execution of SVC2 code 19:

01	19	00	FF
0	S	3	2
M	T	0	6
0	2	00	08
B2	F0	80	00
00	91	00	91
C	O	N	

B2E08000 = SPFP

Time display on hexadecimal display panel

DPFP

Direct access support

Spooler option

Roll option

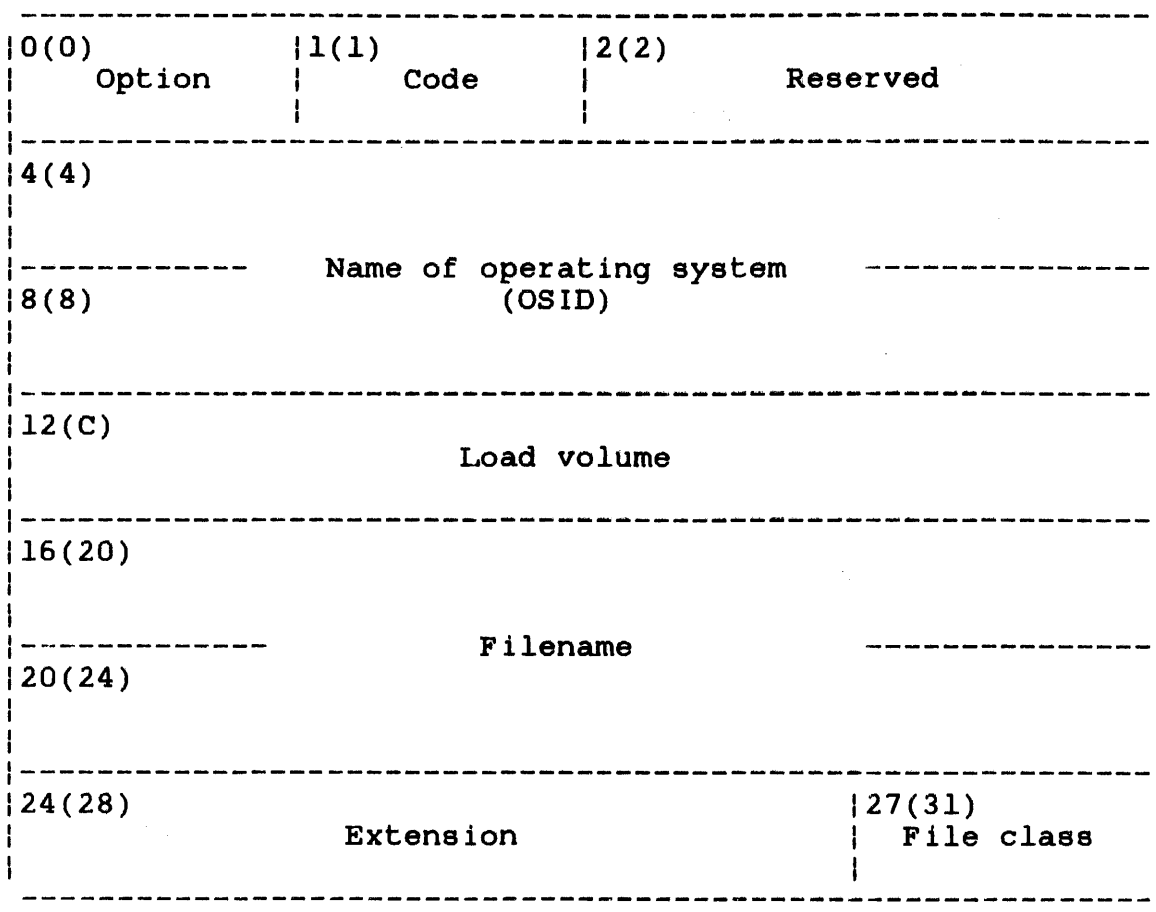
Temporary file support

Multiple register set support

Load read address support

3.19.3 Parameter Block for Option X'02'

If SVC2 code 19 is executed with option X'02' specified in the parameter block option field, use the parameter block format in Figure 3-26.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'02',19
          DC    H'0'
          DS    24
```

Figure 3-26 SVC2 Code 19 Parameter Block Format and Coding for Option X'02'

This parameter block must be 28 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows:

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'02'.
Code	is a 1-byte field that must contain the decimal number 19 to indicate code 19 of SVC2.
Reserved	is a 2-byte field that is reserved and must contain zeros.
Name of operating system (OSID)	is an 8-byte field that receives from the SPT the operating system name in ASCII.
Load volume filename extension file class	represent the fd from which this task was loaded. The fd can be used for subsequent assignments.

Example:

```
          SVC   2,PEEK
          SVC   2,PAUSE
          .
          .
          .
          ALIGN 4
PEEK      DB    X'02',19
          DC    H'0'
          DS    24
          ALIGN 4
PAUSE     DB    0,1
```

Parameter block before execution of SVC2 code 19:

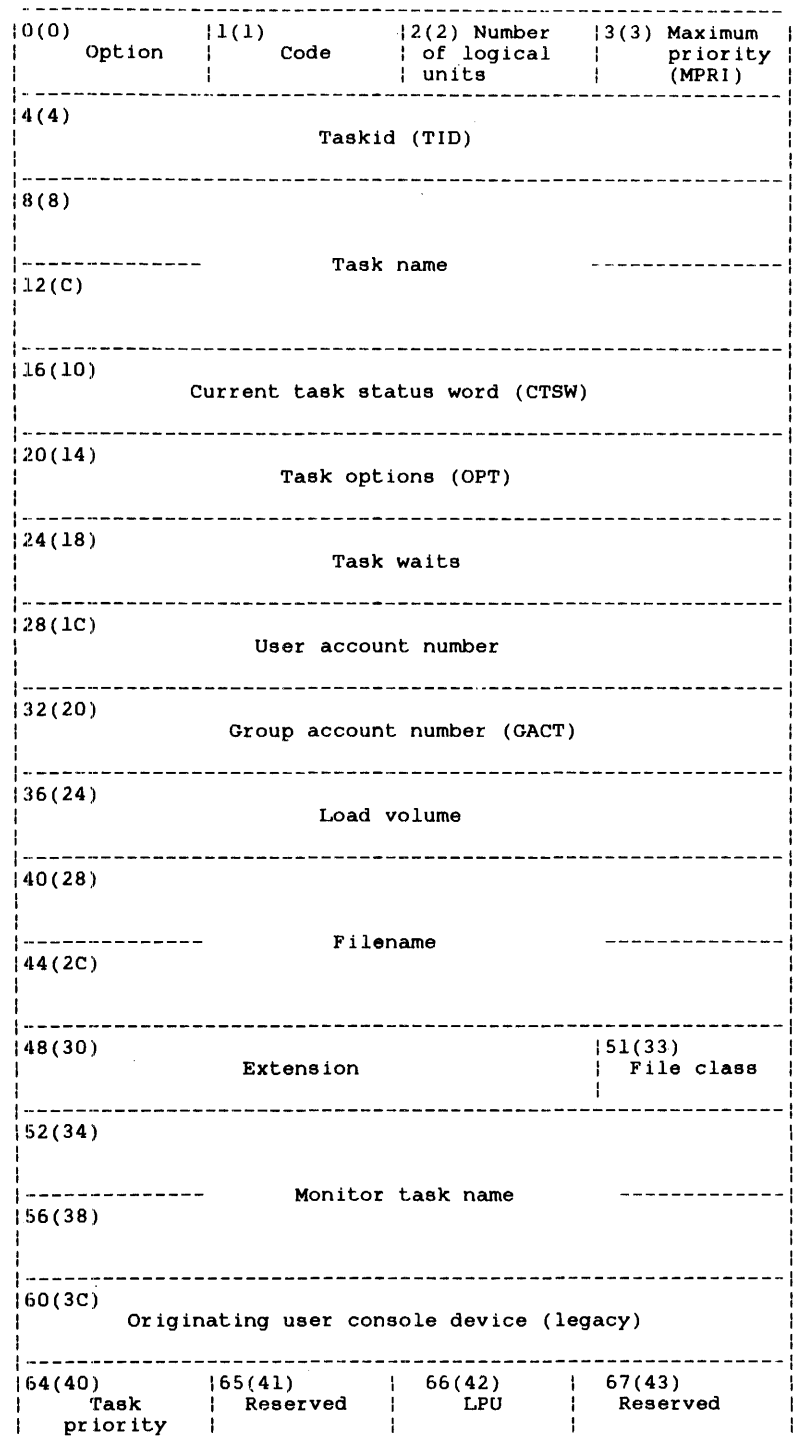
02	19	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00
00	00	00	00

Parameter block after execution of SVC2 code 19:

02	19	00	00
O	S	3	2
M	T	0	6
M	3	0	0
S	U	P	R
V	I	S	R
O	V	Y	S

3.19.4 Parameter Block for Option X'03'

To execute SVC2 code 19 with option X'03' specified in the parameter block option field, use the parameter block format in Figure 3-27. This option is used to obtain extended information on a task.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'03',19
          DS    2
          DC    Y'utask'
          DS    57
          DB    0
          DS    1
          DB    0
```

Figure 3-27 SVC2 Code 19 Parameter Block Format and Coding for Option X'03'

This parameter block must be 68 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'03'.
Code	is a 1-byte field that must contain the decimal number 19 to indicate SVC2 code 19.
Number of logical units (NLU)	is a 1-byte field that receives from the TCB the maximum number of logical units that can be assigned to a task. This hexadecimal number ranges from 0 (X'00') to 254 (X'FE').
Maximum priority (MPRI)	is a 1-byte field that receives from the TCB the highest priority number at which the assigned task can execute. This hexadecimal number ranges from 10 (X'0A') to 249 (X'F9').
Taskid (TID)	is a 4-byte field that contains a hexadecimal number supplied by the user that identifies the task for which the extended task information is being requested. The user obtains this number using the SVC intercept software. See the OS/32 System Level Programmer Reference Manual. The user's own task can be examined by setting the TID field to 0.
Task name	is an 8-byte field that receives from the TCB the name in ASCII of the task for which the extended task information is being requested. If the supplied TID is invalid or the task no longer exists, the task name field is set to binary zeros.
Current task status word (CTSW)	is a 4-byte field that receives from the TCB the hexadecimal number representing bits 0 through 31 of the CTSW.
Task options (OPT)	is a 4-byte field that receives from TCB the hexadecimal number representing bits 0 through 31 of the task option field in the TCB. Table 3-4 lists task options.

Task waits	is a 4-byte field that receives the hexadecimal number representing bits 0 through 31 of the task wait field in the TCB. Table 3-6 lists the wait status bit definitions.
User account number (UACT)	is a 4-byte field that receives the user account number from the TCB. This hexadecimal number is right-justified.
Group account number (GACT)	is a 4-byte field that receives the group account number from the TCB. This hexadecimal number is right-justified.
Load volume	is the fd from which the task was loaded. After the task is loaded, the fd can be assigned to subsequent tasks.
Filename	is the fd from which the task was loaded. After the task is loaded, the fd can be assigned to subsequent tasks.
Extension	is the fd from which the task was loaded. After the task is loaded, the fd can be assigned to subsequent tasks.
File class	is the fd from which the task was loaded. After the task is loaded, the fd can be assigned to subsequent tasks.
Monitor task name	is an 8-byte field that receives the name of the task that is monitoring the specified task.
Originating user console device (legacy)	is a 4-byte field that receives the name of the MTM console from which the specified task was loaded. If the task is not running under MTM, this field contains zeros.
Task priority	is a 1-byte field indicating the priority of the specified task at the time this call is executed.
Reserved	is a 1-byte field that must contain zeros.
Logical processing unit (LPU)	is a 1-byte field that receives the hexadecimal number of the task's current LPU assignment from the TCB. The value of this number ranges from X'00' to X'09'; X'00' indicates the CPU.
Reserved	is a 1-byte reserved field that must contain zeros.

TABLE 3-6 TASK WAIT STATUS BIT DEFINITIONS

BIT POSITION	BIT MASK	MEANING
0-14	Y'0000 0000'	Reserved
15	Y'0001 0000'	Intercept wait
16	Y'0000 8000'	I/O wait
17	Y'0000 4000'	Any IOB wait
18	Y'0000 2000'	Console wait (paused)
19	Y'0000 1000'	Load wait
20	Y'0000 0800'	Dormant
21	Y'0000 0400'	Trap wait
22	Y'0000 0200'	Time of day wait
23	Y'0000 0100'	Suspended
24	Y'0000 0080'	Interval wait
25	Y'0000 0040'	Terminal wait
26	Y'0000 0020'	Roll pending wait
27	Y'0000 0010'	Interrupt initialization (MTM)
28	Y'0000 0008'	Interrupt termination (MTM)
29	Y'0000 0004'	System resource connection wait
30	Y'0000 0002'	Accounting wait
31	Y'0000 0001'	Reserved

NOTE

If bits 0 through 30 are set to 0, the task is active.

Example:

```
             SVC    2,PEEK
             SVC    2,PAUSE
             .
             .
             .
             ALIGN 4
    PEEK     DB     X'03',19
             DS     2
             DC     Y'00'
             DS     57
             DB     0
             DS     1
             DB     0
```

Parameter block before execution of SVC2 code 19:

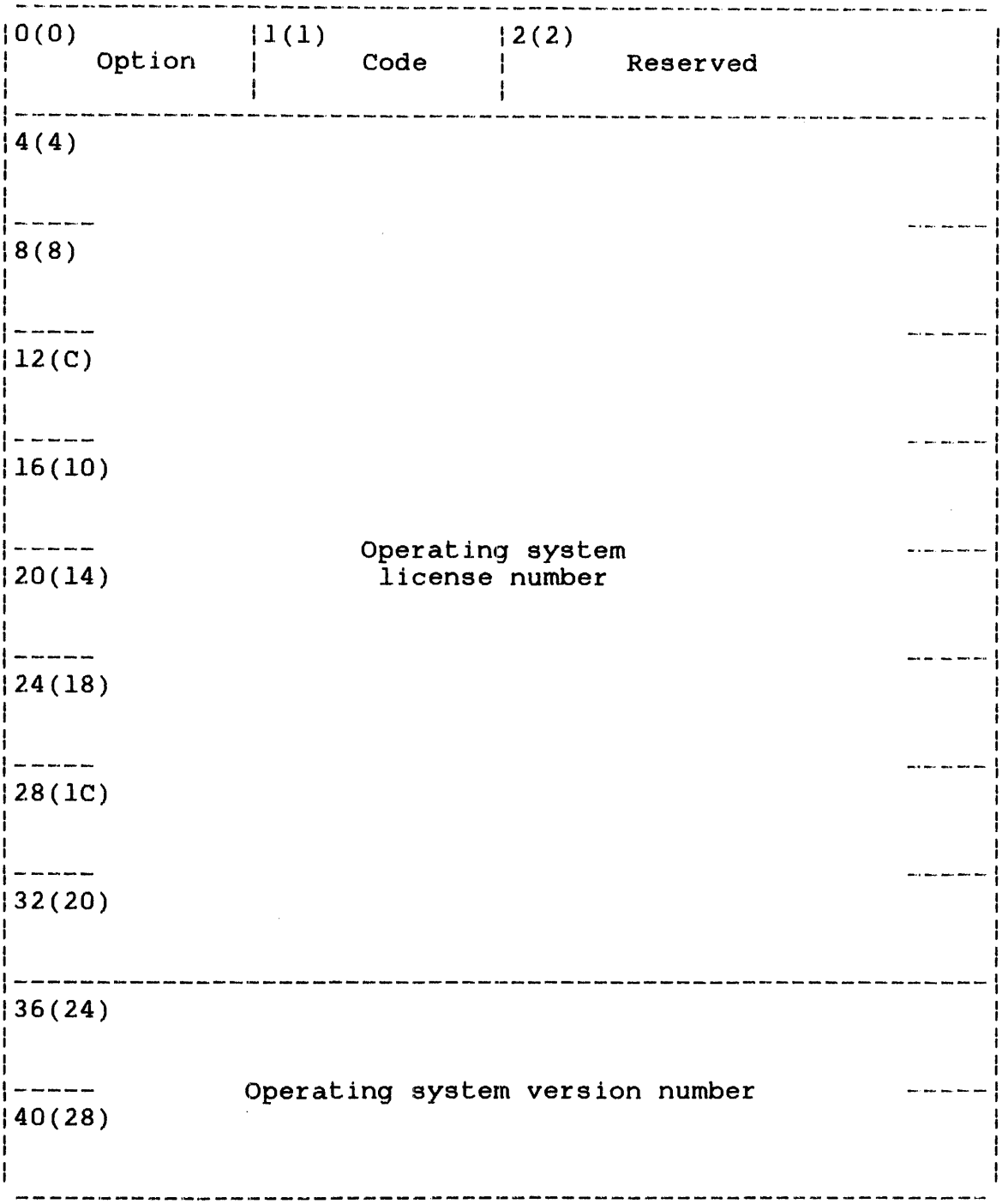
```
 ---------------
|03 |19 |00 |00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|               |
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|               |
|00  00  00  00 |
|---------------|
|00  00  00 |00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00  00  00  00 |
|---------------|
|00 |00 |00 |00 |
 ---------------
```

Parameter block after execution of SVC2 code 19:

03	19	OF	81
00	00	00	00
P	E	E	K
P	E	E	K
00	01	00	01
00	2D	B0	A8
00	00	80	00
00	00	00	91
00	00	00	91
M	3	0	0
S	U	P	R
V	I	S	P
0	V	Y	S
00	00	00	00
00	00	00	00
00	00	00	00
00	00	01	00

3.19.5 Parameter Block for Option X'04'

Option X'04' accesses the license number and current sysgen version of the OS that is currently running on the system. To execute option X'04' of SVC2 code 19, use the parameter block format shown in Figure 3-28.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'04',19
          DC    H'0'
          DS    40
```

Figure 3-28 SVC2 Code 19 Parameter Block Format and Coding Option X'04'

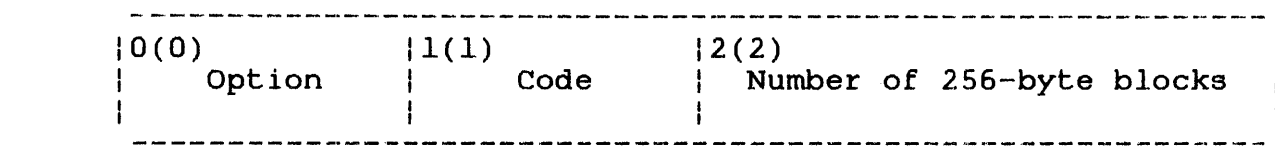
This parameter block must be 44 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that contains the hexadecimal number X'04' indicating option 4 of SVC2 code 19.
Code	is a 1-byte field that contains the decimal number 19 indicating SVC2 code 19.
Reserved	is a 2-byte field that should contain zeros.
OS license number	is a 32-byte (8 fullwords) alphanumeric field that receives the license number of the operating system: e.g., E-0178. Data in this field is left-justified with trailing ASCII blanks (X'20').
OS version number	is an 8-byte (2 fullwords) alphanumeric field that receives the version of the operating system that was specified by the user at sysgen: e.g., 613C.819. Data in this field is left-justified with trailing ASCII blanks (X'20').

3.20 SVC2 CODE 20: EXPAND ALLOCATION

SVC2 code 20 affects only those tasks running under previous 32-bit operating systems and should not be used in a multitasking environment. This SVC provides for compatibility with existing programs; no action is performed. The parameter block for this call is shown in Figure 3-29.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,20
          DC    H'number of 256-byte blocks'
```

Figure 3-29 SVC2 Code 20 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and located in a task-writable segment for option X'80'. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain option X'00' or X'80'.
Code	is a 1-byte field that must contain the decimal number 20 to indicate SVC2 code 20.
Number of 256-byte blocks	is an unused 2-byte field.

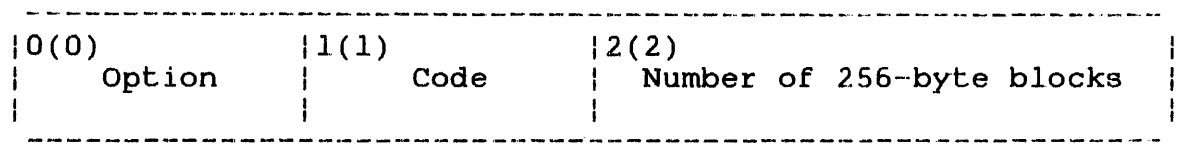
The CC is set after executing SVC2 code 20. Possible CCs follow:

Condition Code:

C	V	G	L	
0	0	0	1	Normal termination with option X'80' specified
0	1	0	0	Normal termination with option X'00' specified

3.21 SVC2 CODE 21: CONTRACT ALLOCATION

SVC2 code 21 affects only those tasks running under previous 32-bit operating systems and should not be used in a multitasking environment. This call provides for compatibility with existing user programs; no action is performed. The parameter block for this call is shown in Figure 3-30.



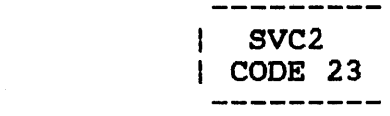

```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,21
          DC    H'number of 256-byte blocks'
```

Figure 3-30 SVC2 Code 21 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain a 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 21 to indicate code SVC2 code 21.
Number of 256-byte blocks	is an unused 2-byte field.



3.22 SVC2 CODE 23: TIMER MANAGEMENT

SVC2 code 23 performs five timer management functions used in coordination with real-time operations:

1. Schedules the addition of a parameter to a task queue when a specified interval has elapsed (option X'00').

2. Waits until completing a specified interval (option X'80').

3. Schedules repetitive additions to a task queue as specified intervals elapse (option X'40').

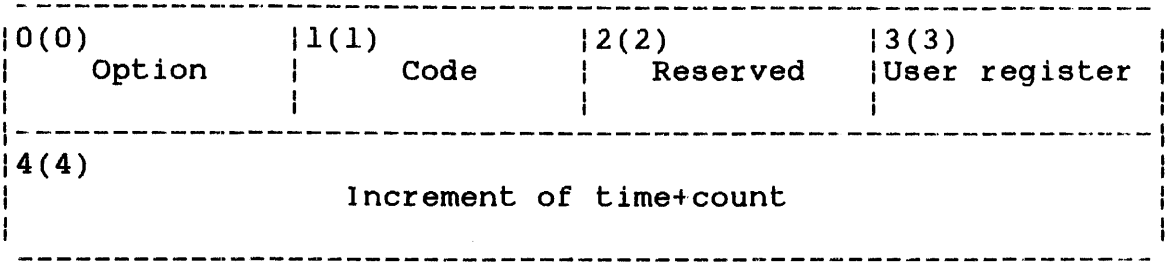
4. Reads time remaining for the specified interval (option X'20').

5. Cancels a previous interval request (option X'10').

Since the five options perform different functions, their parameter block formats and coding differ and are shown as separate parameter blocks. These operations are accomplished through the SVC2 code 23 parameter blocks shown in Figures 3-30 through 3-34.

3.22.1 SVC2 Code 23, Parameter Block for Option X'00'

When specifying option X'00', a timer interval is set up concurrently with the subsequent task executions. Then, an item with a reason code of X'09' is added to the calling task queue when the user-specified interval elapses. This is accomplished through the SVC2 code 23 parameter block for option X'00' shown in Figure 3-31. See the OS/32 Application Level Programmer Reference Manual for information on the task queue.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'00',23,0
          DB    user register
          DC    Y'increment of time'+F'count'
```

Figure 3-31 SVC2 Code 23 Parameter Block Format and Coding for Option X'00'

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'00'.
Code	is a 1-byte field that must contain the decimal number 23 to indicate SVC2 code 23.
Reserved	is a 1-byte field that must contain a zero.
User register	is a 1-byte field that must contain a user-specified register number. Bits 8 through 31 of this register must contain the parameter portion of the item that is added to the task queue when the interval elapses.
Increment of time+count	is a 4-byte field that indicates the number of seconds or milliseconds that must elapse before an item is added to the task queue. The first four bits contain a hexadecimal number indicating how the time period is to be calculated:

- Y'00000000' indicates that the time is calculated in seconds from midnight (time of day).
- Y'10000000' indicates that the time is calculated in milliseconds from the time this call is executed (interval timing).

The remaining bits contain the count or decimal number indicating the number of seconds or milliseconds.

A decimal number greater than 86,399 indicates days in the future. For a detailed explanation of time of day and interval timing, see Sections 3.11 and 3.12.

Before executing this call, prepare the task to handle a task queue trap. See the OS/32 Application Level Programmer Reference Manual.

After the interval is started and the CC is set, the task continues processing or enters a trap wait state. Possible CCs follow:

Condition Code:

C	V	G	L	
0	0	0	0	Interval started; normal termination
0	1	0	0	Insufficient system space available

Example:

```
            LI     3,C'ABC'
            SVC    2,TIMRQ
            SVC    9,TRAPWAIT
            .
            .
            .
            ALIGN  4
TIMRQ       DB     X'00',23,0
            DB     3
            DC     Y'10000000'+F'30000'
            ALIGN  4
TRAPWAIT    DC     Y'88000200'
            DC     Y'0'
```

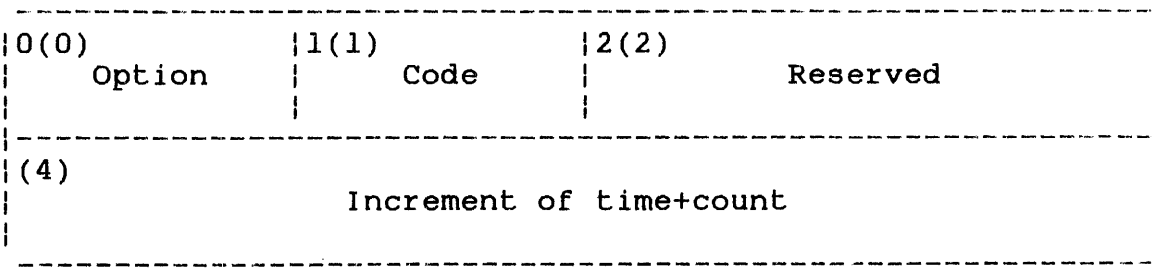
If this call is executed and insufficient system space is available, no time period elapses, no item is added to the task queue and the CC is set to 4 (V bit set). If this call is executed and the task is unprepared to handle this trap, no item is added to the task queue and the task has effectively lost an interrupt.

If queue overflow occurs after the specified interval elapses, the end of task code is set to 1000 and the task terminates abnormally.

If the interval is calculated as time of day and that specified time has already passed, the same time on the following day is assumed.

3.22.2 SVC2 Code 23, Parameter Block for Option X'80'

If option X'80' is specified, the calling task is placed in a timer wait state until a specified interval elapses. Nothing is added to the calling task queue. This is accomplished through the SVC2 code 23 parameter block for option X'80' shown in Figure 3-32.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'80',23
          DC    H'0'
          DC    Y'increment of time'+F'count'
```

Figure 3-32 SVC2 Code 23 Parameter Block Format and Coding for Option X'80'

This parameter block is eight bytes long, fullword boundary-aligned and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal value X'80'.
Code	is a 1-byte field that must contain the decimal number 23 to indicate SVC2 code 23.
Reserved	is a reserved 2-byte field that must contain zeros.
Increment of time+count	is a 4-byte field that indicates the number of seconds or milliseconds that must elapse before the task is released from the wait state. The first four bits contain a hexadecimal number indicating how the time is to be calculated: • Y'00000000' indicates that the time is calculated in seconds from midnight (time of day). • Y'10000000' indicates that the time is calculated in milliseconds from the time this call is executed (interval timing). The remaining bits contain the count or decimal number indicating the number of seconds or milliseconds. A decimal number greater than 86,399 indicates days in the future.

After the specified interval elapses, the task resumes execution with the instruction following SVC2. The possible CCs follow:

Condition Code:

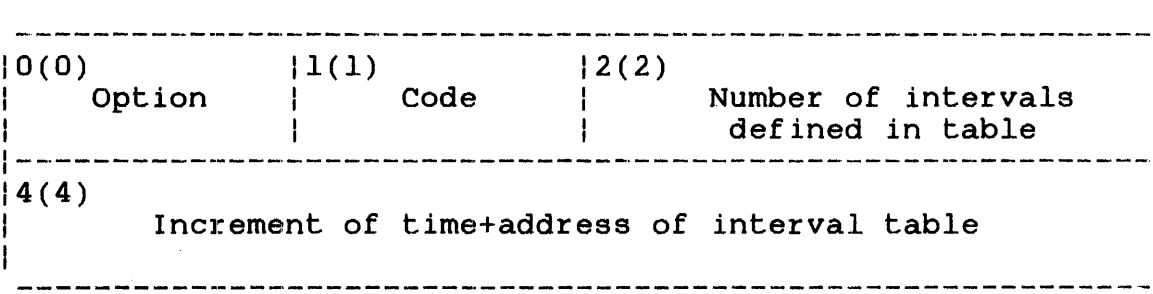
C	V	G	L	
0	0	0	0	Internal started; normal termination
0	1	0	0	Insufficient system space available; no wait occurred

If this call is executed and insufficient system space is available, no interval elapses, no item is added to the task queue and the CC is set to 4 (V bit set).

If the interval is calculated as time of day and that specified time has already passed, the same time on the following day is assumed.

3.22.3 SVC2 Code 23, Parameter Block for Option X'40'

If option X'40' is specified, items with reason code X'09' are repetitively added to the calling task queue at user-defined intervals within a specific time period until the task terminates or cancels the time interval request with SVC2 code 23 option X'10. The user-defined intervals that are within a specific time period must all be specified the same way, either as time of day intervals or as interval-timing intervals. This is accomplished through the SVC2 code 23 parameter block for option X'40' shown in Figure 3-33.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'40',23
          DC    H'number of intervals defined in table'
          DC    Y'increment of time'+A(interval table)
```

Figure 3-33 SVC2 Code 23 Parameter Block Format and Coding for Option X'40'

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'40'.
Code	is a 1-byte field that must contain the decimal number 23 to indicate SVC2 code 23.
Number of intervals defined in table	is a 2-byte field that must contain the decimal number indicating the number of intervals the user defined in the table.
Increment of time+address of interval table	is a 4-byte field that indicates the address of the table containing all the user-defined intervals within a specified time period. The first four bits contain a hexadecimal number indicating how the time designated by the interval table is to be calculated:

- Y'00000000' indicates that the time is calculated in seconds from midnight (time of day).
- Y'10000000' indicates that the time is calculated in milliseconds from the time this call is executed (interval timing).

The remaining bits contain a hexadecimal number indicating the address of the interval table. This table must be fullword boundary-aligned and defined as follows.

Format:

```
table    DC    F'count'            First interval
         DC    F'parameter'
         DC    F'count'            Second interval
         DC    F'parameter'
                .                   .
                .                   .
                .                   .
         DC    F'count'            Last interval
         DC    F'parameter'
```

Parameters:

table	is the user-specified name for the interval table.
DC	is the operation code, define constant, for the instruction.
F	is the type code, fullword, for the instruction.
count	is the decimal number indicating how many seconds or milliseconds must elapse before an item is added to the task queue. The decimal numbers specified for time of day intervals can be any number except 0 and must be specified in ascending order with each count at least one greater than the previous count. The decimal number for interval-timing intervals can be any decimal number except 0. This decimal value occupies bits 4 through 31 of the count field.
parameter	is the item to be added to the task queue when its associated interval elapses. This item occupies bits 8 through 31 of one slot of the task queue. The first byte contains reason code X'09'. See the OS/32 Application Level Programmer Reference Manual.

The time period in which the user-defined intervals occur differs for time of day intervals and interval-timing intervals. The time period for time of day intervals ranges from the day on which the first interval occurs through and including the day on which the last interval occurs. The time period is the sum of days on which the intervals occur. In the following example, the total time period is three days.

Example:

```
          ALIGN 4
INTABLE   DC    F'54000'          1500 hours of current day
          DC    F'1'
          DC    F'140400'         1500 hours of second day
          DC    F'2'
          DC    F'226800'         1500 hours of third day
          DC    F'3'
          DC    F'230400'         1600 hours of third day
          DC    F'4'
```

The time period for interval timing is the sum of all intervals in the table.

Example:

```
          ALIGN 4
INTABLE   DC    F'18000'          first interval
          DC    F'A1'
          DC    F'36000'
          DC    F'A2'             second interval
```

In the above example, the time period is equal to 54000ms. The time period is repetitively executed until the task cancels the time interval request via SVC2 code 23 option X'10' or goes to end of task. Before executing this call, prepare the task to handle this trap as described in the OS/32 Application Level Programmer Reference Manual.

As the specified intervals are elapsing, the task can continue processing. After executing this call, the CC is set to these possible settings:

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	Insufficient system space available; no wait occurred

If this call is executed and insufficient system space is available, no interval elapses, nothing is added to the task queue, and the CC is set to 4 (V bit set).

If this call is executed and the task is not prepared to handle this trap, nothing is added to the task queue. The task has effectively lost an interrupt.

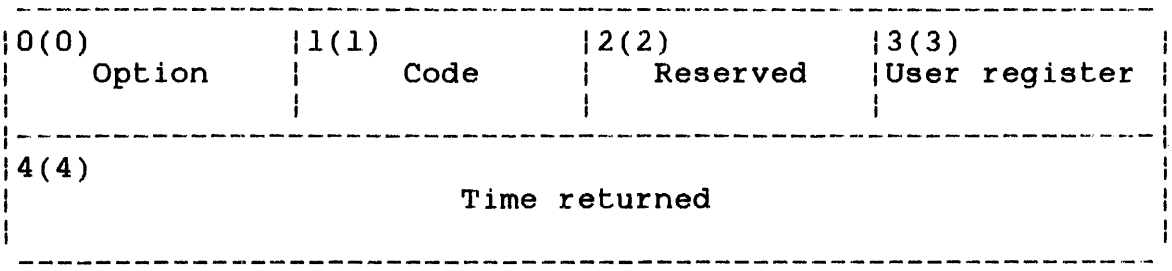
If queue overflow occurs after one of the specified intervals elapses, the end of task code is set to 1000 and the task terminates abnormally.

If the time period is calculated as time of day and the specified time for the first interval has already passed, the same time in the following period is assumed.

If the time period is calculated as time of day and one of the specified intervals in the interval table is 0 or not in ascending order, the task is paused and a message is displayed.

3.22.4 SVC2 Code 23, Parameter Block for Option X'20'

SVC2 code 23 reads the time remaining until the interval previously established with option X'00' or X'40' elapses. This is accomplished through the SVC2 code 23 parameter block for option X'20' shown in Figure 3-34.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'20',23,0
          DB    user register
          DC    Y'increment of time returned'
```

Figure 3-34 SVC2 Code 23 Parameter Block Format and Coding for Option X'20'

This parameter block must be eight bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'20'.
Code	is a 1-byte field that must contain the decimal number 23 to indicate SVC2 code 23.
Reserved	is a 1-byte field that must contain a 0.

User register is a 1-byte field that must contain the user register number. Bits 8 through 31 of this register should contain the parameter associated with the desired starting interval.

Time returned is a 4-byte field that contains a hexadecimal number indicating how the time is returned for the type of interval being read, as follows:

- Y'00000000' indicates the number of seconds from midnight specified for the time of day wait interval in the parameter block for option X'00' of SVC2 code 23.
- Y'10000000' indicates the milliseconds remaining from the time this call is executed to the completion of the time interval specified in the parameter block for option X'40' of SVC2 code 23.

NOTE

If the timer entry that is being read is set for a time of day wait interval (option X'00'), only the value for the time of day interval can be read. An interval timing readout cannot be made for this task. Similarly, if the task is set for interval timing (option X'40'), only an interval readout can be made.

The register in the user register field specifies the parameter associated with the interval to be read. When executed, this call finds the value of the time of day wait interval or the milliseconds remaining for a timing interval by searching for the parameter associated with the interval on the timer queue. The value read is stored in bits 4 through 31 of the time returned field. Bits 0 through 3 remain unchanged. Hence, the final value in the time returned field after execution of the SVC can be represented as follows:

Time returned = increment of time + count

If the interval was started with option X'40' specified and more than one interval in the table has the same parameter associated with it, the current time in the desired interval might not be the one that is read. Each interval must have a unique parameter associated with it.

After executing this call, the CC is set to the possible CCs following.

Condition Code:

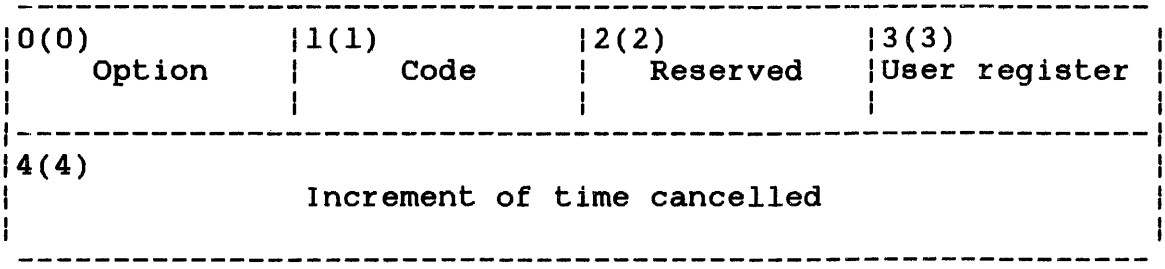
C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	No interval associated with parameter 2 located in user-specified register

Example:

```
TEST1     EQU   1
          LI    3,TEST1
          SVC   2,TIMRQ
          SVC   2,RDTIME
          SVC   9,TRAPWAIT
          .
          .
          .
          ALIGN 4
RDTIME    DB    X'20',23,0,3
          DC    Y'10000000'
          ALIGN 4
TIMRQ     DB    X'00',23,0,3
          DC    Y'10000000'+F'90000'
          ALIGN 4
TRAPWAIT  DC    Y'88000200'
          DC    Y'0'
```

3.22.5 SVC2 Code 23, Parameter Block for Option X'10'

This SVC cancels an interval request that was previously established with option X'00' or X'40'. This is accomplished through the SVC2 parameter block for option X'10' shown in Figure 3-35.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    X'10',23,0
          DB    user register
          DC    Y'increment of time cancelled'
```

Figure 3-35 SVC2 Code 23 Parameter Block Format and Coding for Option X'10'

This parameter block is eight bytes long, fullword boundary-aligned, and does not have to be located in a task-writable segment. A general description of each field in the parameter block follows:

Fields:

Option	is a 1-byte field that must contain the hexadecimal number X'10'.
Code	is a 1-byte field that must contain the decimal number 23 to indicate SVC2 code 23.
Reserved	is a 1-byte field that must contain a 0.
User register	is a 1-byte field that must contain the user register number. Bits 8 through 31 of this user register contain the parameter associated with the interval to be cancelled.

Increment of time cancelled	is a 4-byte field that must contain a hexadecimal number indicating how time is being calculated for the interval to be cancelled. The increments of time are:

- Y'00000000' indicates seconds from midnight (time of day).
- Y'10000000' indicates milliseconds from the time this call is executed (interval timing).

When this call is executed, all previous interval requests that match both the increment of time specified and the parameter located in the user register are cancelled. If the interval to be cancelled is part of a periodic group, the entire time period is cancelled.

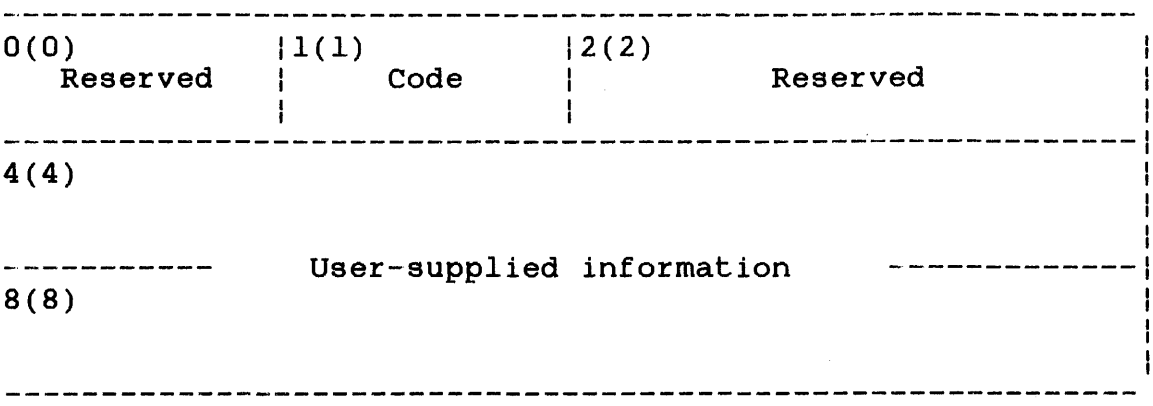
After executing SVC2 code 23, the CC is set to the possible conditions following.

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	No previous interval request exists that matches the parameter provided

3.23 SVC2 CODE 24: SET ACCOUNTING INFORMATION

SVC2 code 24 stores eight bytes of user-supplied information in the ATF task completion or data overflow account records of the ATF. This is accomplished through the SVC2 code 24 parameter block shown in Figure 3-36.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,24
          DC    H'0'
          DC    D'user-supplied information'
```

Figure 3-36 SVC2 Code 24 Parameter Block Format and Coding

This parameter block is 12 bytes long, fullword boundary-aligned, and does not have to be in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Reserved	is a 1-byte field that must contain a 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal number 24 to indicate SVC2 code 24.
Reserved	is a reserved 2-byte field that must contain zeros.
User-supplied information	is an 8-byte field that must contain the user-supplied information to be stored in the ATF task completion or data overflow account records.

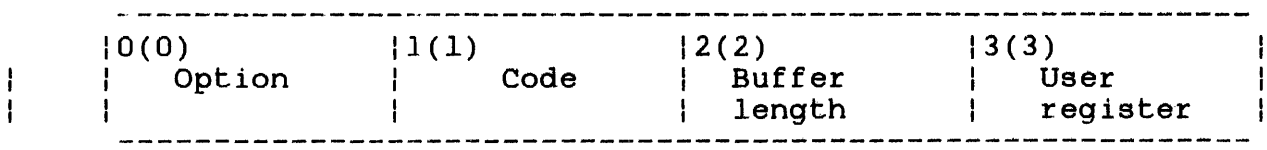
If more than one SVC2 code 24 is executed by a task, the user-supplied information in the last SVC2 code 24 executed is stored in the ATF. The CC is always set to 0.

3.24 SVC2 CODE 25: FETCH ACCOUNTING INFORMATION

SVC2 code 25 fetches task accounting information and stores it into a user-specified fixed or variable buffer. The accounting information accessed is:

- User CPU time
- Operating system CPU time
- APU execution time
- Wait time
- Roll time

This is accomplished through the SVC2 code 25 parameter block shown in Figure 3-37.




```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    'option'
          DB    25
          DB    'buffer length'
          DB    user register
```

Figure 3-37 SVC2 Code 25 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and does not have to be in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Option — is a 1-byte field that must contain a number specifying one of the following options:

- Option 0 means that the fixed-size user buffer is specified.
- Option 1 means that the variable-size user buffer is specified.

Code — is a 1-byte field that must contain the decimal number 25 to indicate SVC2 code 25.

Buffer length — is a 1-byte field that specifies the length of the buffer from 24 to 256 bytes for option 1. Option 0 ignores this field.

User register — is a 1-byte field that must contain a user-specified number of the register that contains the starting address of the area to receive the accounting information. This area is 16-bytes long, fullword boundary-aligned and must be located in a task-writable segment, as shown in Figure 3-38. The CC is always set to 0.

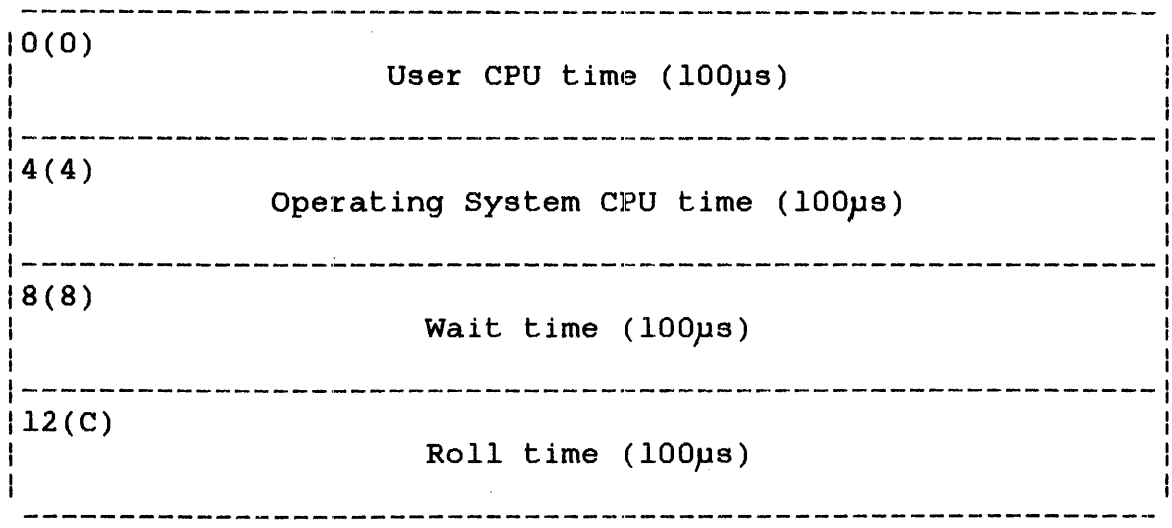


Figure 3-38 Fixed-Size User Buffer Receiving Accounting Information

For option 1, the buffer must be a minimum of 23 bytes long; its contents, upon execution of the SVC, are shown in Figure 3-39.

0(0) Size of buffer	1(1) Buffer used	2(2) Reserved	3(3) Reserved
4(4) User CPU time (100µs)			
8(8) Operating System CPU time (100µs)			
12(C) Wait time (100µs)			
16(10) Roll time (10µs)			
20(14) APU time (100µs)			
Available for future expansion of accounting			

Figure 3-39 Variable-Size User Buffer Receiving Accounting Information

For option 1, the CC is set to 0 if the request is successful. The V flag is set if the buffer is smaller than required.

3.25 SVC2 CODE 26: FETCH DEVICE NAME

SVC2 code 26 searches for a user-supplied volume name in the VMT and returns the name of the device on which that volume is mounted. The format for the SVC2 code 26 parameter block is shown in Figure 3-40.

0(1) Reserved	1(1) Code	2(2) User register 1	3(3) User register 2

```
          SVC   2,parblk
          .
          .
          .
parblk    ALIGN 4
          DB    0,26
          DB    user register number 1
          DB    user register number 2
```

Figure 3-40 SVC2 Code 26 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary-aligned, and does not have to be in a task-writable segment. The fields are described as follows.

Fields:

Reserved	is a 1-byte field that must contain a value of 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal value 26 to indicate SVC2 code 26.
User register 1	is a 1-byte field that must contain a user-specified register number. The specified register contains a pointer to a fullword containing a 4-character volume name.

User register 2	is a 1-byte field that must contain a user-specified register number. The specified register contains the address of the area receiving the device name. This area is four bytes long, fullword boundary-aligned and must be located in a task-writable segment.

NOTE

User registers 1 and 2 can specify the same register number.

Possible CCs occurring after SVC2 code 26 execution follow:

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination
0	1	0	0	Specified volume off-line; no fetch occurred

Example:

```
            .
            .
            .
            LA     R1,MTMVOLN
            LA     R2,MTMDEVN
            SVC    2,FTCHDEVN
            .
            .
            .
            ALIGN  4
FTCHDEVN    DB     0,26
            DB     R1
            DB     R2
```

3.26 SVC2 CODE 27: MEMORY MANAGEMENT

SVC2 code 27 allows a task to access and modify entries (except shared ones) within the PST in its TCB. This SVC can only be called by tasks running on MAT machines. It is used by the virtual task manager (VTM) support routines. The format for the SVC2 code 27 parameter block is shown in Figure 3-41.

0(0) Option (SV227.OP)	1(1) Code (SV227.CD)	2(2) User register 1 (SV227.R1)	3(3) User register 2 (SV227.R2)
4(4) A (destination buffer) (SV227.BF)			

```
          SVC   2,parblk
           .
           .
           .
          ALIGN 4
parblk    DB    option,27,reg1,reg2
          DAC   BUFFADR
```

Figure 3-41 SVC2 Code 27 Parameter Block Format and Coding

This parameter block is eight bytes long, fullword boundary-aligned and located in a task-writable segment. A general description of the parameter block follows.

Fields:

Option (SV227.OP)	is a 1-byte field that contains a decimal number specifying one of the following option codes:

- 0 (SV227.0) indicates the first byte of each PST entry, starting at PSTE 0 and ending with the last private segment table entry (PSTE) of the task's impure segment (PSTE indicated by TCB.CTOP), is moved sequentially to a byte buffer specified by the SV227.BF field. After byte is is moved, the reference (bit 0) of each PSTE is reset.
- 1 (SV227.1) indicates that bits 15 to 31 of PSTE 0 are added to the user register specified by the SV227.R1 field. The result of the addition is stored in the PSTE identified by the number contained in the register specified by the SV227.R2 field.
- 2 (SV227.2) indicates the value in user register 1 is stored in TCB.UTOP and the value in user register 2 is stored in TCB.CTOP.

Code (SV227.CD)	is a 1-byte field that contains the decimal number 27 indicating SVC2 code 27.
User register 1 (SV227.R1)	is a 1-byte field that contains a user-specified register number. If option 0 is specified, this field is unused but must be reserved.
User register 2 (SV227.R2)	is a 1-byte field that contains a user-specified register number. If option 0 is specified, this field is unused but must be reserved.
A (destination buffer) (SV227.BF)	is a 4-byte field that contains the address of the first byte of a user-specified buffer to which the entries in the PST are copied. This buffer should be located in a task-writable writable segment. This field is omitted for options 1 and 2.

SVC2 code 27 sets the CC field in the PSW as follows.

Condition Code:

C	V	G	L	
0	0	0	0	SVC2 code 27 completed. No errors.
0	0	0	1	Size of PSTE exceeds task allocation of memory. Entry not stored in PST.
0	0	1	0	Illegal PSTE number.
0	1	0	0	Shared bit set in PSTE. This entry cannot be modified.
1	0	0	0	Value of UTOP is greater than value of CTOP.

3.27 SVC2 CODE 29: UNPACK FILE DESCRIPTOR

SVC2 code 29 converts a packed fd from the file directory or an SVC7 parameter block to its unpacked format. The format for the SVC2 code 29 parameter block is shown in Figure 3-42.

0(0) Option (UPFD.OPT)	1(1) Code (UPFD.COD)	2(2) Source register (UPFD.SRC)	3(3) Destination register (UPFD.DST)
4(4) Source pointer for option X'01' (UPFD.SAD)			
8(8) Destination pointer for option X'01' (UPFD.DAD)			

```
          SVC   2,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    option,29
          DB    source register
          DB    destination register
          DAC   A(packed fd)
          DAC   BUFFADR
```

Figure 3-42 SVC2 Code 29 Parameter Block Format and Coding

This parameter block is 12 bytes long, fullword boundary-aligned and located in a task-writable segment. A general description of each field on the parameter block follows.

Fields:

Option (UPFD.OPT) is a 1-byte field that contains a hexadecimal number indicating one or more of the following SVC2 code 29 option codes:

- X'80' (UPFO.NNN) forces an account number (nnn) into the unpacked fd even if the file was allocated without account privileges.
- X'40' (UPFO.PGS) forces a /P, /G or /S account designation into the unpacked fd even if the file was allocated with account privileges. If the account number in the packed fd cannot be converted to a P, G or S file class, P is returned to the unpacked fd and the G bit of the CC is set.

NOTE

If neither X'80' nor X'40' is specified, the fd is packed according to the account privileges in effect when the file was allocated.

- X'20' (UPFO.NOV) unpacks the fd in the file directory entry specified in the UPFD.SAD field. When unpacked, the fd does not include a volume name.

NOTE

If X'20' is not specified, SVC2 code 29 unpacks the fd contained in the SVC7 parameter block whose address is specified by the UPFD.SRC or UPFD.SAD field. When unpacked, the fd includes a volume name.

- X'10' (UPFO.WID) indicates the unpacked fd includes any blanks that exist in the packed fd. If option X'10' is not specified, all blanks are suppressed.
- X'08' (UPFO.BLA) indicates the unpacked fd is formatted with blanks. If X'08' is not specified, the unpacked fd is formatted in the standard unpacked fd format including a colon (:), period (.) and slash (/).

- X'01' (UPFO.ADR) indicates the source addressed of the packed fd is specified by the UPFD.SAD field of the SVC2 code 29 parameter block. The unpacked fd is to be stored in the address location specified by the UPFD.DAD field of the parameter block.

NOTE

If X'01' is not specified, the source and destination addresses are to be found in the registers specified by the UPFD.SRC and UPFD.DST fields, respectively.

Code (UPFD.COD)	is a 1-byte field that contains the decimal number 29 indicating SVC2 code 29.
Source register (UPFD.SRC)	is a 1-byte field that specifies the number of the register that contains the address of the file directory entry or SVC7 parameter block that contains the source of the packed fd.
Destination register (UPFD.DST)	is a 1-byte field that specifies the number of the register that contains the address of a 24-byte buffer in a task-writable segment where the unpacked fd is to be stored.

NOTE

If option X'01' has been specified, the source register and destination register fields must be filled with zeros.

Source pointer for option X'01' (UPFD.SAD)	is a 4-byte field that contains the address of the file directory entry or SVC7 parameter block that contains the source of the packed fd. This field is used only if option X'01' has been specified.
Destination pointer for option X'01' (UPFD.DAD)	is a 4-byte field that specifies the address of a 24-byte buffer in a task-writable segment where the unpacked fd is to be stored. This field is used only if option X'01' has been specified.

The following examples demonstrate the use of SVC2 code 29.

Example 1:

```
SVC229    PROG  SVC2,29 EXAMPLE - UNPACK FD
          SVC   2,UFD                 UNPACK FD
          SVC   2,PAUSE

          ALIGN 4
PAUSE     DB    0,1,0,0               PAUSE
UFD       DB    X'A1',29,0,0          NNN, FD, :./, SQUEZ, ADDR
          DAC   SOURCE                PACK FD INPUT
          DAC   DEST                  UNPACK FD OUTPUT

SOURCE    DB    C'TEST   CSS',71      INPUT PACKED FD


DEST      DS    24                    OUTPUT UNPACKED FD
          END
```

Example 2:

```
SVC229A   PROG  SVC2,29 EXAMPLE - UNPACKED FD
          NLSTM
          $SVC7
          LA    1,SVC7PBLK            ADDRESS OF SOURCE
          LA    2,DEST                ADDRESS OF DESTINATION
          SVC   2,UFD                 UNPACK FD
          SVC   2,PAUSE

          ALIGN  4
PAUSE     DB    0,1,0,0               PAUSE
UFD       DB    X'58',29,1,2          PGS, SVC7, BLANKS, BLANKS, REG
          DAC   0
          DAC   0


SVC7PBLK  DS    SVC7.                 INPUT PACKED FD
          ORG   SVC7PBLK+SVC7.VOL
          DC    C'MTM '
          DC    C'TEST     '
          DB    C'CSS'
          DB    C'G'

DEST      DS    24                    OUTPUT UNPACKED FD
          END
```

CHAPTER 4
END OF TASK SUPERVISOR CALL 3 (SVC3)

4.1 INTRODUCTION

The SVC3 instruction terminates task execution.

4.2 SVC3: END OF TASK

The following is an example of an SVC3 instruction.

Format:

```
SVC 3,n
```

Fields:

SVC	is the mnemonic used as an operation code specifying a supervisor call.
3	is a decimal number indicating it is SVC3.
n	is a decimal number ranging from 0 to 255 used as the end of task code when the task terminates. If this number is greater than 255, it is truncated to eight bits. End of task codes greater than 255 are reserved for system use. The end of task code can be used in subsequent command substitution system (CSS) conditional testing. The following standard end of task codes are used: • 0 indicates normal termination. • 255 indicates termination caused by cancellation. • 1000 indicates termination caused by task queue overflow on expiration of time interval. • 1100 indicates a mapping error in an impure segment during roll-in.

- 1101 indicates a mapping error in a pure segment during roll-in.
- 1102 indicates a pure segment was not found during roll-in.
- 1105 indicates an input/output (I/O) error on a roll file for an impure segment.
- 1106 indicates an I/O error on a roll file for a pure segment.
- 1200 indicates termination caused by the expiration of a central processing unit (CPU) time limit.
- 1210 indicates termination caused by the expiration of an I/O transfer limit.

In addition, the end of task code can be stored in a register. For example, to generate a code of 4, use the following sequence:

```
LHI     R8,4
SVC     3,0(R8)
```

If I/O is in progress when an SVC3 is executed, write operations continue until completed and then terminate normally; read operations terminate immediately.

For each logical unit (lu), read operations and SVC15 operations are halted as if an SVC1 halt I/O were issued. In addition, each open lu is checkpointed (for a resident task) or closed (for a nonresident task) as if the corresponding SVC7 had been issued.

The SVC3 may be intercepted by the task's monitor. Normally the operating system generates messages for foreground and background tasks and the multi-terminal monitor (MTM) generates messages for terminal and batch jobs. The system messages contain the end of task code and accounting information for the task.

For more information on using end of task codes in CSS, see the OS/32 Operator Reference Manual.

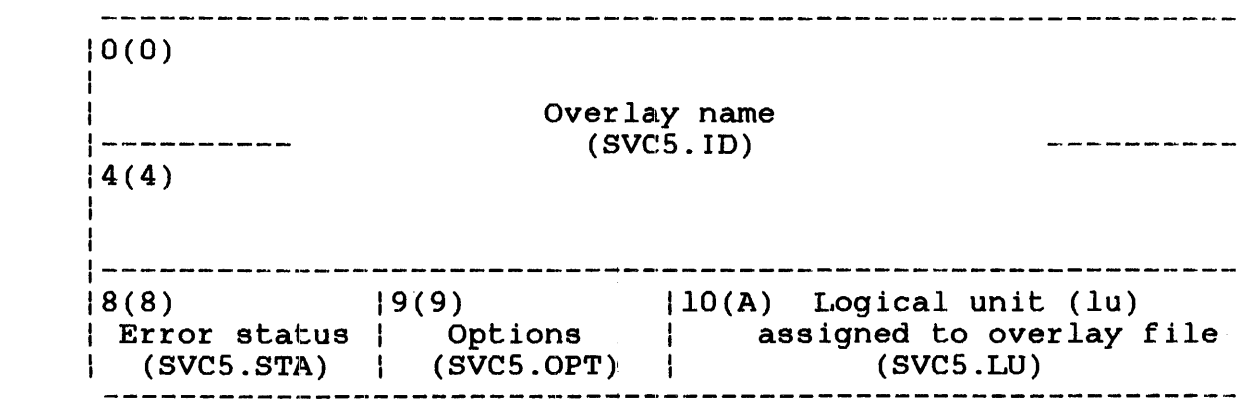
CHAPTER 5
FETCH OVERLAY SUPERVISOR CALL 5 (SVC5)

5.1 INTRODUCTION

SVC5 permits user-controlled loading of overlays generated by Link or TET. Loading of overlays is accomplished through the SVC5 parameter block in Figure 5-1. The SVC5 parameter block is 12 bytes long, fullword boundary-aligned and must be in a task-writable segment.

5.2 SVC5: FETCH OVERLAY

Figure 5-1 is an example of an SVC5 fetch overlay.




```
          SVC   5,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    C'8 character overlay name'
          DS    1
          DB    X'option'
          DC    H'lu'
```

Figure 5-1 SVC5 Parameter Block Format and Coding

Fields:

Overlay name (SVC5.ID) — is an 8-byte field specifying the name of the overlay to be loaded. If the overlay name requires less than eight characters, the data in this field must be left-justified with trailing spaces.

For overlays generated by TET, the overlay name field is matched against the overlay name in the loader information block (LIB) of the overlay file. For overlays generated by Link, this field is matched against the overlay name specified in the OVERLAY command. (From the overlay descriptor table (ODT) of the task image file.) If the overlay name is found, loading of the overlay proceeds as if an automatic overlay load occurred.

Error status (SVC5.STA) — is a 1-byte field that receives the appropriate error code when an error occurs during the execution of SVC5. The status returned is one of the following:

- X'00' indicates overlay loaded successfully.
- X'10' indicates load failed.
- X'20' indicates a mismatch on overlay name.
- X'40' indicates the overlay would not fit in allocated memory. This error code applies to overlays generated by TET only.

Options (SVC5.OPT) — is a 1-byte field that must contain one of the following options:

- Option X'01' indicates load from lu without positioning.
- Option X'04' indicates load from lu after rewind.

The option byte is not required for overlays generated by Link and is ignored.

lu assigned to overlay file (SVC5.LU)	is a 2-byte field containing the device to which the overlay file must be assigned and must be positioned to the first byte of the LIB for the overlay generated by TET. This field is not required for overlays generated by Link and is ignored.

The calling task is placed in a wait state until the overlay is loaded. If the overlay is successfully loaded, the calling program can branch and link to the overlay.

Certain messages might be generated as a result of loading overlays created by Link. These messages are discussed in the OS/32 Link Reference Manual.

Example:

```
          SVC   5,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    C'MARIANNE'
          DB    0                 Initialize status to 0
          DB    1                 Load without positioning
          DC    H'2'              Overlay assigned to lu2
```

CHAPTER 6
INTERTASK COMMUNICATIONS SUPERVISOR CALL 6 (SVC6)

6.1 INTRODUCTION

SVC6 provides a task with the ability to communicate with and control another task. The task that issues an SVC6 is known as the calling task. An SVC6 can be directed to any task within the calling task's execution environment, including the calling task itself. The task to which SVC6 is directed is called the directed task.

Before a calling task can issue an SVC6, that task must be linked with one of the following task options:

- COMMUNICATE - This option allows a calling task to perform SVC6 intertask communications functions (see Section 6.2.1).
- CONTROL - This option allows a calling task to perform SVC6 intertask control functions (see Section 6.2.1).
- NOCOM - This option prevents the calling task from executing SVC6 communications functions.
- NOCON - This option prevents the calling task from executing SVC6 control functions.
- SVCCONTINUE - This option causes an SVC6 executed in a background environment to be ignored.

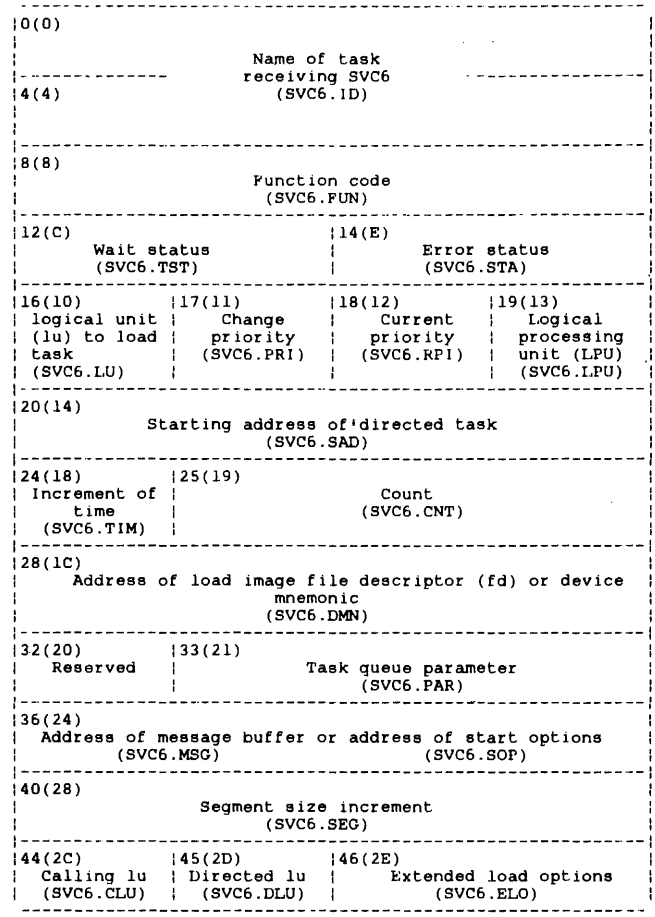
In an OS/32 real-time environment, only foreground tasks can issue an SVC6. If a background task attempts to issue this call, the operating system treats the call as an illegal call or NOP, depending on the SVCPAUSE task option in effect. See the OS/32 Link Reference Manual for more information on the task options that apply to SVC6.

NOTE

SVC6 cannot be executed in a multi-terminal monitor (MTM) environment unless specified as an MTM option at system generation (sysgen). The use of SVC6 in an MTM environment can also be restricted on an account basis via MTM account privileges. See the MTM System Planning and Operator Reference Manual for more information.

6.2 SVC6: INTERTASK COMMUNICATIONS

Communication and control between tasks are accomplished through the SVC6 parameter block shown in Figure 6-1.




```
          SVC   6,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    C'8-byte name of task receiving SVC6'
          DC    Y'function code'
          DS    2 bytes for wait status
          DS    2 bytes for error status
          DB    1 byte for lu to load task
          DB    1 byte for change priority
          DS    1 byte for current priority
          DB    1 byte for LPU
          DC    A(start address of directed task)
          DC    Y'increment of time+count'
          DC    C'4-byte device mnemonic' or A(fd)
          DC    Y'task queue parameter'
          DC    A(message buffer or start options)
          DC    Y'segment size increment'
          DB    1 byte for calling lu number
          DB    1 byte for directed lu number
          DS    2 bytes for extended load options
```

Figure 6-1 SVC6 Parameter Block Format and Coding

This parameter block must be 48 bytes long, fullword boundary-aligned and located in a task-writable segment. For a detailed description of the functions of each field in the parameter block, see the appropriate section in this chapter. A brief description of each field in the parameter block follows.

Fields:

Name of task receiving SVC6 (SVC6.ID)	is an 8-byte field that contains the task name to which SVC6 is directed. If SVC6 is a self-directed call, this field is not required. The name must consist of one to eight alphanumeric characters with the first character always alphabetic. It is left-justified in the field with spaces.
Function code (SVC6.FUN)	is a 4-byte field that contains the hexadecimal number indicating the function to be performed.
Wait status (SVC6.TST)	is a 2-byte field that receives the hexadecimal value of bits 16 through 31 of the directed task's wait status fullword when a SVC6 is executed. If the calling task wants to check the wait status of the directed task at any time, an SVC6 can be issued with the function code set to Y'80000000' or Y'C0000000'.
Error status (SVC6.STA)	is a 2-byte field that receives the appropriate error code when an error occurs during execution of the SVC6. If no error occurs, a value of 0 is stored in this field.
lu to load task (SVC6.LU)	is a 1-byte field used only when a load operation is requested. This field specifies the logical unit (lu) currently assigned to the directed task that is to be loaded.
Change priority (SVC6.PRI)	is a 1-byte field used only when a change priority operation is requested. This field must contain a user-specified hexadecimal number indicating the new priority to which the task is to be changed. The hexadecimal number must have a decimal value ranging from 10 to 249.

Current priority (SVC6.RPI)	is a 1-byte priority field that receives a hexadecimal number indicating the priority at which the task is executing when an SVC6 is executed. If the calling task wants to check the current priority of the directed task at any time, an SVC6 can be issued with bits 0 and 1 of the function code set to 10 or 11, and the remaining bits set to 0.
Logical processing unit (LPU) (SVC6.LPU)	is a 1-byte field used only when an LPU assignment operation is requested. It contains a user-specified hexadecimal number indicating the LPU assigned to the task (0...max LPU).
Starting address of directed task (SVC6.SAD)	is a 4-byte field used only when a start operation is requested. This field must contain a user-specified hexadecimal number indicating the address where the directed task is to start execution.
Increment of time (SVC6.TIM)	is a 1-byte field used in conjunction with the count field only when the delay-start operation is requested. This field must contain a user-specified hexadecimal number indicating how the time is to be calculated. These hexadecimal numbers are: • X'00' indicates seconds from midnight (time of day). • X'10' indicates milliseconds from the time this call is executed (interval timing).
Count (SVC6.CNT)	is a 3-byte field used in conjunction with the increment-time field only when a delay-start operation is requested. This field must contain a user-specified decimal number indicating how many seconds or milliseconds must elapse before the directed task starts execution.
Address of load image fd or device mnemonic (SVC6.DMN)	is a 4-byte field that contains a user-specified device mnemonic of a trap generating device when the connect, thaw, sint, freeze or unconnect operations are requested. If a task is to be loaded with bit 3 (load and proceed) of the extended load option field set, this field should contain the address of the file descriptor (fd) of the file containing the task to be loaded.

Reserved	is a reserved 1-byte field that must contain a 0.
Task queue parameter (SVC6.PAR)	is a 3-byte field used only when the add to task queue or connect to trap generating device operations are requested. This field must contain the user-specified parameter that is to be added to the task queue of the directed task.
Address of message buffer (SVC6.MSG) or address of start options (SVC6.SOP)	is a 4-byte field used only when a send message operation or start operation is requested. For the send message operation, this field must contain a user-specified hexadecimal number indicating the address of the buffer containing the message to be sent to the directed task. For the start operation, this field must contain the address of the start options to be included at run-time.
Segment size increment (SVC6.SEG)	is a 4-byte field used only when a load operation is requested and must contain the user-specified hexadecimal number indicating the number of bytes used to expand the task's allocated memory.
Calling lu (SVC6.CLU)	is a 1-byte field that must contain the user-specified hexadecimal number representing the logical unit of the calling task.
Directed lu (SVC6.DLU)	is a 1-byte field that must contain the user-specified hexadecimal number representing the logical unit of the directed task.
Extended load options (SVC6.ELO)	is a 2-byte field used only when the extended load options are requested. This field must contain a user-specified hexadecimal number indicating one or more of the options listed in Table 6-3.

6.2.1 Function Code (SVC6.FUN)

SVC6.FUN has 21 functions for intertask communications and control. These functions are listed in Table 6-1.

TABLE 6-1 SVC6.FUN FUNCTIONS

COMMUNICATION FUNCTIONS	CONTROL FUNCTIONS	
Send data	Direction	Freeze
Send message	End task	Unconnect
Add to task queue	Load task	Assign LPU
	Resident task	Transfer to LPU
	Suspend	Transfer to CPU
	Change priority	Release
	Send lu	Nonresident
	Receive lu	Rollable
	Connect	Nonrollable
	Thaw	Start
	Sint	

These functions are specified by setting the appropriate bits in the function code field shown in Figure 6-2. Each bit setting and its corresponding function are listed in Table 6-2.

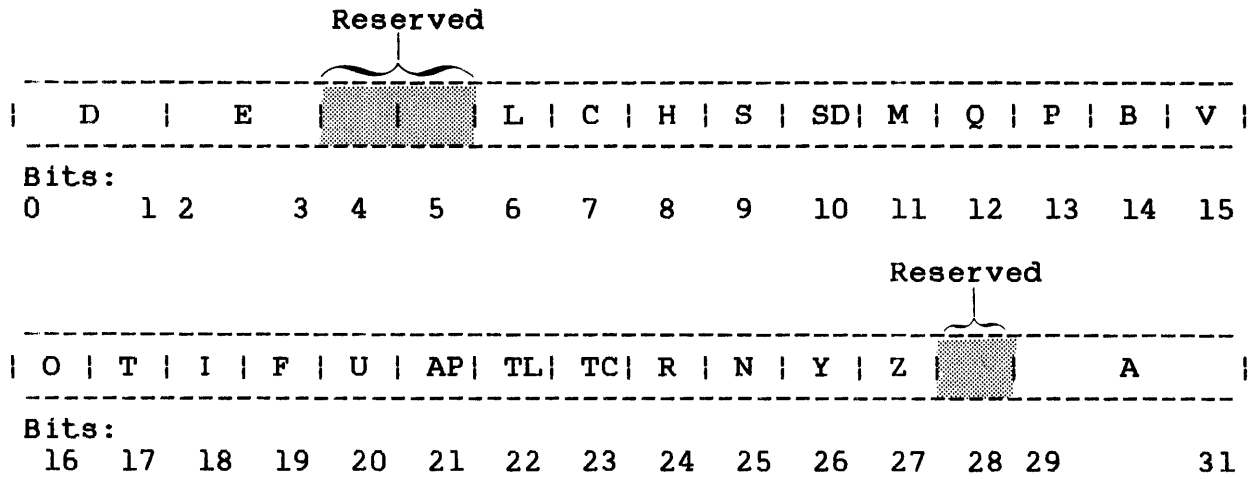


Figure 6-2 SVC6 Function Code Field

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC6 CALLS

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
0 (D) 1	Direction (SFUN.DOM=10) (SFUN.DSM=11)	The task to which SVC6 is directed	00 = illegal 01 = illegal 10 = other task 11 = self-directed
2 (E) 3	End task (SFUN.ECM=01) (SFUN.EDM=10 or =11)	End or terminate task execution.	00 = no function requested 01 = cancel 10 = delete directed task 11 = delete directed task
4 5	N/A	Reserved	00 = reserved
6 (L) 7 (C)	Load task (SFUN.LM=10 SFUN.LXM=01)	Load the directed task.	00 = no function requested 01 = illegal 10 = load task 11 = load task (10) with extended load options (01)
8 (H)	Resident task (SFUN.HM)	Make the directed task resident in memory.	0 = no function requested 1 = make task resident
9 (S)	Suspend (SFUN.SM)	Put the directed task into a wait state.	0 = no function requested 1 = put task into wait state
10 (SD)	Send data (SFUN.DM)	The calling task sends a variable length message to the directed task.	0 = no function requested 1 = send message
11 (M)	Send message (SFUN.MM)	The calling task sends a 64-byte message to the directed task.	0 = no function requested 1 = send message

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC6 CALLS (Continued)

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
12 (Q)	Add to task queue (SFUN.QM)	Add parameter to the directed task's queue.	0 = no function requested 1 = add to task queue
13 (P)	Change priority (SFUN.PM)	Change the priority of the directed task.	0 = no function requested 1 = change the priority
14 (B)	Send lu (SFUN.XSM)	Calling task's lu is assigned to the directed task.	0 = no function requested 1 = send lu
15 (V)	Receive lu (SFUN.XRM)	Directed task's lu is assigned to the calling task.	0 = no function requested 1 = receive lu
16 (O)	Connect (SFUN.OM)	A trap generating device is connected to the directed task.	0 = no function requested 1 = connect device to task
17 (T)	Thaw (SFUN.TM)	Enable interrupts on a trap generating device connected to the directed task.	0 = no function requested 1 = enable interrupts
18 (I)	Sint (SFUN.IM)	Simulate interrupt on a trap generating device to the directed task.	0 = no function requested 1 = simulate interrupt
19 (F)	Freeze (SFUN.FM)	Disable interrupts on a trap generating device connected to the directed task.	0 = no function requested 1 = disable interrupts

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC6 CALLS (Continued)

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
20 (U)	Unconnect (SFUN.UM)	Disconnect the specified trap generating device from the directed task.	0 = no function requested 1 = disconnect device from task
21 (AP)	Assign LPU (SFUN.LPM)	Assign an LPU to the directed task.	0 = no function requested 1 = assign LPU
22 (TL)	Transfer to LPU (SFUN.XLM)	Make the task LPU-directed.	0 = no function 1 = set LPU-directed
23 (TC)	Transfer to CPU (SFUN.XCM)	Make the task CPU-directed.	0 = no function requested 1 = reset LPU-directed
24 (R)	Release (SFUN.RM)	Remove the directed task from a wait state.	0 = no function requested 1 = remove task from state
25 (N)	Nonresident (SFUN.NM)	Make the directed task nonresident.	0 = no function requested 1 = make task nonresident
26 (Y)	Rollable (SFUN.RLM)	Make the directed task rollable.	0 = no function requested 1 = make task rollable
27 (Z)	Nonrollable (SFUN.NRM)	Make the directed task nonrollable.	0 = no function requested 1 = make task nonrollable
28	N/A	Reserved	0 = reserved

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC6 CALLS (Continued)

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
29 30 (A) 31	Start (SFUN.SIM=010) (SFUN.SOM=011) (SFUN.SDM=100) or (SFUN.SDM=110)	Start execution of the directed task.	000 = no function requested 001 = illegal 010 = start 011 = start with start option 100 = delay start 101 = delay start with start option 110 = delay start

6.2.2 Direction (SFUN.DOM, SFUN.DSM) Function

The direction function identifies the task to be affected by the SVC6 call. The name of this task is located in the task name field. The required parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0 and 1 of the function code field (SVC6.FUN)

If the bit setting equals 10 (SFUN.DOM), the call is directed to the task whose name is specified in the task name field. If the bit setting equals 11 (SFUN.DSM), the call is self-directed (directed to the task initiating the call). A self-directed call does not require a name in the task name field. A call can also be self-directed by setting the bits to 10 and specifying the calling task name in the task name field of the parameter block. Other bit settings for bit positions 0 and 1 are illegal and cause an error code to be stored into the error status field of the parameter block.

6.2.3 End Task (SFUN.ECM, SFUN.EDM) Function

The end task function abnormally terminates (cancels) execution of the directed task. The required parameter block fields are the task name field and bit positions 0, 1, 2 and 3 of the function code field. When the bit setting equals 01 (SFUN.ECM) and the directed task is resident, these operations occur:

- Task execution is cancelled (end of task code = 255).
- The task remains in memory.
- All of the task's assigned files and devices are checkpointed, not closed.

When the bit setting equals 01 (SFUN.ECM) and the directed task is nonresident, these operations occur:

- Task execution is cancelled (end of task code = 255).
- The task is removed from memory.
- All of the task's assigned files and devices are closed.

When the bit setting equals 10 or 11 (SFUN.EDM), these operations occur:

- Task execution is cancelled (end of task code = 255).
- The task is made nonresident (if it was resident).
- The task is removed from memory.
- All of the task's assigned files and devices are closed.

If this call is self-directed, SVC6 is immediately terminated. After the call is executed, an end of task code 255 indicating abnormal termination is returned to the user.

6.2.4 Load Task Functions

The load task function loads the directed task into memory. Options are provided for the calling task to wait until the load is completed or to continue execution and receive a trap when the load is completed.

When a task is loaded, the operating system reads the loader information block (LIB) of the task to see if any needed shared segments are already in memory. If they are not in memory, the auto loader feature automatically loads them, provided sufficient memory exists. See the OS/32 Operator Reference Manual. When all shared segments named in the LIB are memory resident, the operating system builds linkages to them.

6.2.4.1 Load Task (SFUN.LM) Function

The required parameter block fields for bit setting 10 in bits 6 and 7 (SFUN.LM) are:

- Task name field (SVC6.ID)
- Bits 0, 1, 6 and 7 of the function code field
- lu to load task field (SVC6.LU)

Before executing this call, the lu specified in the parameter block must be assigned to the file or device containing the directed task image. This call is processed as a load wait.

The lu must be positioned to the first byte of the task LIB. When this call is executed, the directed task is loaded from the specified lu into a memory area large enough to hold the task. If such an area does not exist and the roll option is specified, the directed task is rolled out to a file on the roll volume and placed in a wait state. While the directed task is being loaded, the calling task is placed in a wait state. When the directed task is loaded, its task name becomes the name specified in the task name field of the parameter block. The calling task is released from the wait state, and the lu is positioned to the record following the loaded task. If the same task is to be reloaded from other than a direct access file with the same lu assigned, the lu must be rewound by using SVC1 prior to each subsequent load. For direct access files, the load task function automatically rewinds the file and initializes the start address to zero.

If the following error conditions occur, SVC6 is rejected, and an error code is stored in the error status field of the parameter block:

- The receiving task is already loaded into memory.
- The task name specified in the parameter block is invalid.
- The call is self-directed.
- The system does not have a memory area large enough to hold the receiving task and does not support the roll option.
- The requested memory size specified in the segment size increment field is larger than the total system memory space.
- The directed task is a background task. (Background tasks can be loaded only from the system console.)
- The lu is not positioned to the LIB or the LIB is invalid.

6.2.4.2 Load Task with Extended Load Options (SFUN.LXM) Function

The extended load options can be specified at load time and are located in the extended load options field of the parameter block (see Figure 6-3).

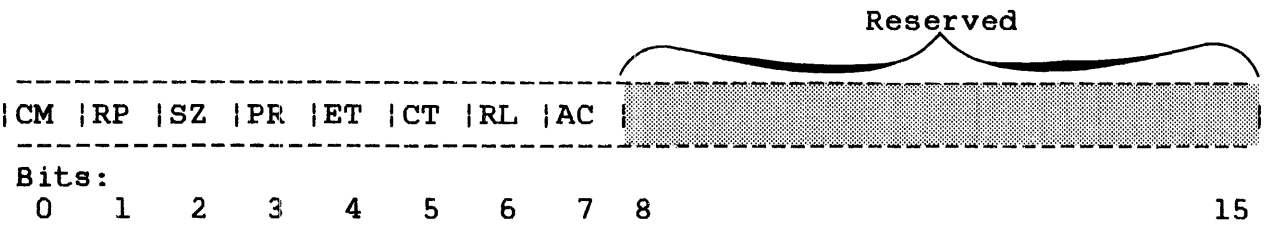


Figure 6-3 Extended Load Options Field

The required parameter block fields for bit setting 11 in bits 6 and 7 (SFUN.LXM) are:

- Task name field (SVC6.ID)
- Bits 0, 1, 6 and 7 of the function code field
- lu to load task field (SVC6.LU) (required when the load wait extended option is specified)
- Address of load image fd (SVC6.DMN) (required when load and proceed extended option is specified)
- Extended load options field (SVC6.ELO)
- Segment size increment field (SVC6.SEG) (required only when the extended load option SELO.SZM is set).

When a task is loaded with the SFUN.LXM enabled, any options specified in the extended load options field are in effect during the loading of the directed task. See Table 6-3 for a list of the available options.

When the extended load and proceed option is requested, the calling task continues executing while the directed task is loaded. The directed task is loaded from the file indirectly specified by the device mnemonic field in the SVC6 parameter block. This field should contain the address of the fd of the task image file to be loaded. If the roll option had been specified when the directed task was link-edited, the private image segment of the task is rolled out to disk if sufficient memory space is not available.

When bit 3 (load and proceed) of the extended load options field is not set, execution of the calling task is suspended during loading of the directed task. This is called a load wait operation. After a load wait operation is completed, the calling task is released from suspension and the lu assigned to the directed task image file is positioned at the record following the last byte of the task image. If the task is again loaded from the same lu, an SVC1 rewind operation should be performed on the task image file prior to that load.

TABLE 6-3 EXTENDED LOAD OPTIONS FIELD BIT DEFINITIONS

BIT POSITION	OPTION AND MASK NAME	MEANING
0 (CM)	Intertask communication (SELO.CMM)	If bit 0 equals 1, the directed task that was loaded into memory can execute the SVC6 communication functions. If bit 0 equals 0 and the loaded receiving task issues an SVC6 communication function, the call is rejected, and an error code is stored in the error status field of the parameter block.
1 (RP)	Subtask reporting (SELO.RPM)	If bit 1 equals 1, the calling task becomes a monitor task and the directed task becomes a subtask. This causes the subtask to report all status changes during execution to the monitor task through task traps. If bit 1 equals 0, the directed task is not a subtask. No subtask status changes are reported.
2 (SZ)	Segment size increment (SELO.SZM)	If bit 2 equals 1, the size of the task workspace is increased by adding a user-specified number of bytes. This hexadecimal number must be located in the parameter block segment size increment field. If bit 2 equals 0, the workspace set by the WORK= parameter of the LINK OPTION command is used.

TABLE 6-3 EXTENDED LOAD OPTIONS FIELD BIT DEFINITIONS (Continued)

BIT POSITION	OPTION AND MASK NAME	MEANING
3 (PR)	Load and Proceed (SELO.PRM)	If bit 3 equals 1, the calling task continues executing while the directed task is being loaded from the file specified by the device mnemonic field of the SVC6 parameter block. A trap to the calling task occurs if bit 20 of its task status word (TSW) equals 1 (load and proceed complete). When the trap occurs, the reason code is 7 and the SVC6 parameter block address is added to the task queue.* If bit 3 equals 0, the calling task is suspended while the directed task is loaded from the specified lu. This lu must be assigned to the file or device from which the task is to be loaded.
4 (ET)	Prevent executive task (e-task) or diagnostic task (d-task) (SELO.ETM)	If bit 4 equals 1, any directed task that is an e-task or d-task is not loaded. If the calling task issues an SVC6 call to load an e-task or d-task with this bit set, the call is rejected, and an error code is stored in the parameter block error status field. If the loading task has bit 4 set and the directed task was linked with the ACPRIVILEGE, DISC or INTERCEPT task option, the task will be loaded but these options will be disregarded.
5 (CT)	Intertask Control (SELO.CTM)	If bit 5 equals 1, the directed task that is loaded can execute the control functions of SVC6. If bit 5 equals 0 and the directed task that is loaded issues an SVC6 control function, the call is rejected, and an error code is stored into the error status field of the parameter block.**

TABLE 6-3 EXTENDED LOAD OPTIONS FIELD BIT DEFINITIONS
(Continued)

BIT POSITION	OPTION AND MASK NAME	MEANING
6 (RL)	Roll (SELO.RLM)	If bit 6 equals 1, the directed task is forced to be a rollable task regardless of the roll option established by Link. If bit 6 equals 0, the directed task uses the roll option established by Link.
7 (AC)	Accounting (SELO.AEM)	If bit 7 equals 1, the directed task that is loaded is given the accounting option. This setting overrides the NOACCOUNT option established by Link. If bit 7 equals 0, the directed task uses the accounting option established by Link.

*When bit 3 equals 1, all other SVC6 functions are ignored except the Start function and send start options. If the calling task terminates while the directed task is being loaded, the load continues, no trap occurs, and no status is stored in the parameter block error status field.

**Self-directed task generating device functions can be executed if bit 5 equals 1.

6.2.5 Task Resident (SFUN.HM) Function

The task resident function makes the directed task memory resident regardless of what options were specified by Link.

At end of task, the open logical units of a resident task are checkpointed and the task remains in memory. A resident task can be rollable. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 8 of the function code field

6.2.6 Suspend (SFUN.SM) Function

The suspend function places the directed task into a wait state. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 9 of the function code

The directed task remains in the wait state until an SVC6 releasing the suspended task (bit 24 of the function code) is executed. If this call is self-directed, it causes the calling task to suspend itself. To release the calling task from the wait state, another task must be available to subsequently release it.

This function can be used to suspend execution of auxiliary processing unit (APU) active or ready tasks. See the OS/32 System Level Programmer Reference Manual for more information on using SVC6 in a Perkin-Elmer Model 3200MPS System.

6.2.7 Send Data (SFUN.DM) Function

Blocks of data that are communicated from one task to another are called messages. The send data function allows a task to send variable length messages to another task.

6.2.7.1 Send Data Message Buffer for Sending Task

To pass a message from one task to another via the send data function, certain data structures are required. The most important of these structures is the send data message buffer. The structure of this message buffer allows the directed task to receive a variable length message in the format in which it was sent. The maximum length of a message that can be sent is determined by the size and number of the message buffers set up by the directed task to receive the message. However, the actual length of the message is determined by the number and size of message buffers set up by the task issuing the SVC6.

Hence, two data structures are required by the calling task: the SVC6 parameter block and the send data message buffer. The required SVC6 parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 10 of the function code field
- Address of the buffer containing the message to be sent (SVCG.MSG) (if a chain of buffers is to be sent, only the address of the first buffer in the chain is required)

The format of the send data buffer for the calling task is shown in Figure 6-4.

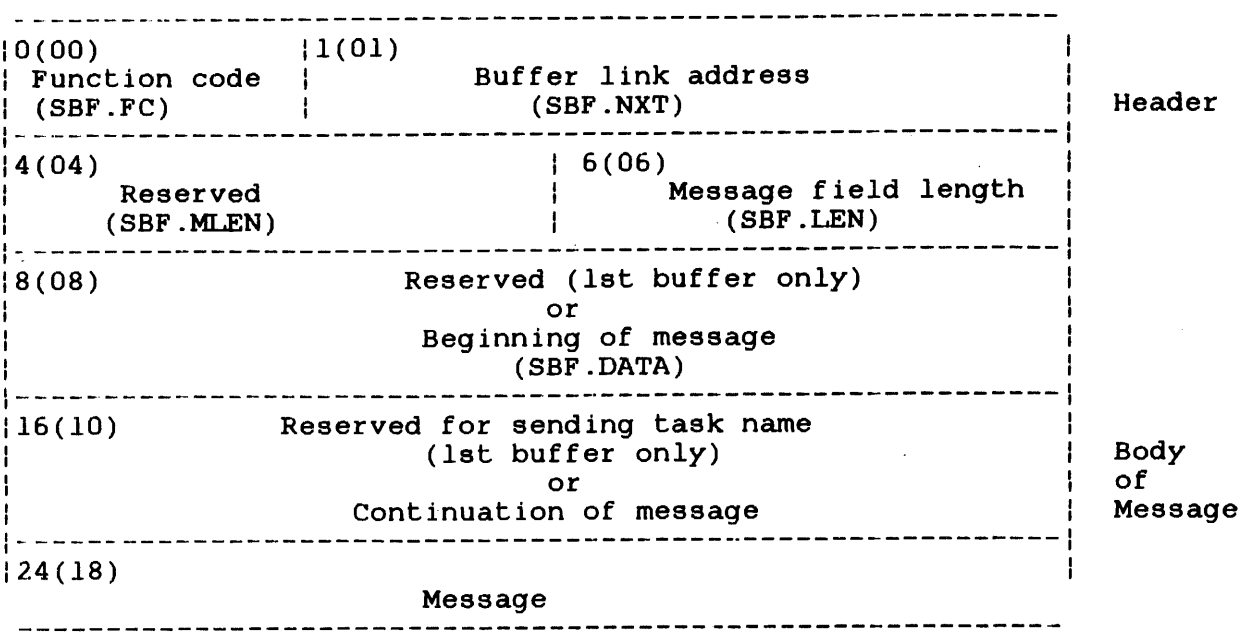


Figure 6-4 Send Data Message Buffer Format for Calling Task

Each send data message buffer can vary in length provided that the buffer is aligned on a fullword boundary and its total length is equal to an integral number of fullwords. Note that the message buffer consists of two parts: the header and the body of the message. The message body holds the data that is to be sent. Because the send data function allows the size of a message to be variable, the length of the body is determined only by the quantity of data that is to be sent by SVC6.

Note that if the buffer is the only buffer containing the message to be sent (or the first buffer in a chain), the first 16 bytes of the message body are reserved and filled with zeros. When the message is transferred to the directed task buffer, the first eight bytes of the message body of the directed task buffer are filled with zeros, the next eight bytes are filled with the sending task's name (left-justified and padded with blanks). All remaining buffers in the chain use these first 16 bytes of the message body to hold data.

A description of the fields in the message header follows:

Function code (SBF.FC)	is a 1-byte field indicating whether the buffer is the only buffer to be sent or is a member of a message buffer chain. The function codes are: • X'00' indicates that the buffer is an intermediate buffer in a chain. • X'10' indicates that the buffer is the last buffer in a chain. • X'20' indicates that the buffer is the first buffer in a chain. • X'30' indicates that the buffer is the only buffer to be sent.
Buffer link address (SBF.NXT)	is a 3-byte field specifying the address of the next buffer in the chain. The operating system ignores this field in buffers with a function code of X'10' or X'30'.
Reserved (SBF.MLEN)	is a 2-byte field reserved for use by the directed task.
Message field length (SBF.LEN)	is a 2-byte field specifying the length in bytes of the message body for that buffer. This length must be a multiple of 4.

6.2.7.2 Free Send Data Message Buffers for Receiving Task

Before a directed task can receive a message, the following structures must be contained within the task address space:

- Free send data message buffers
- Free buffer list queue
- Task queue
- User-dedicated location (UDL) containing the address of the task queue, free buffer list queue and TSW with address of send data trap service routine
- TSW initialized to enable send data traps

The total length of the send data message buffers should be sufficient to hold the entire message transferred to those buffers by the calling task. The format of the send data message buffers is shown in Figure 6-5.

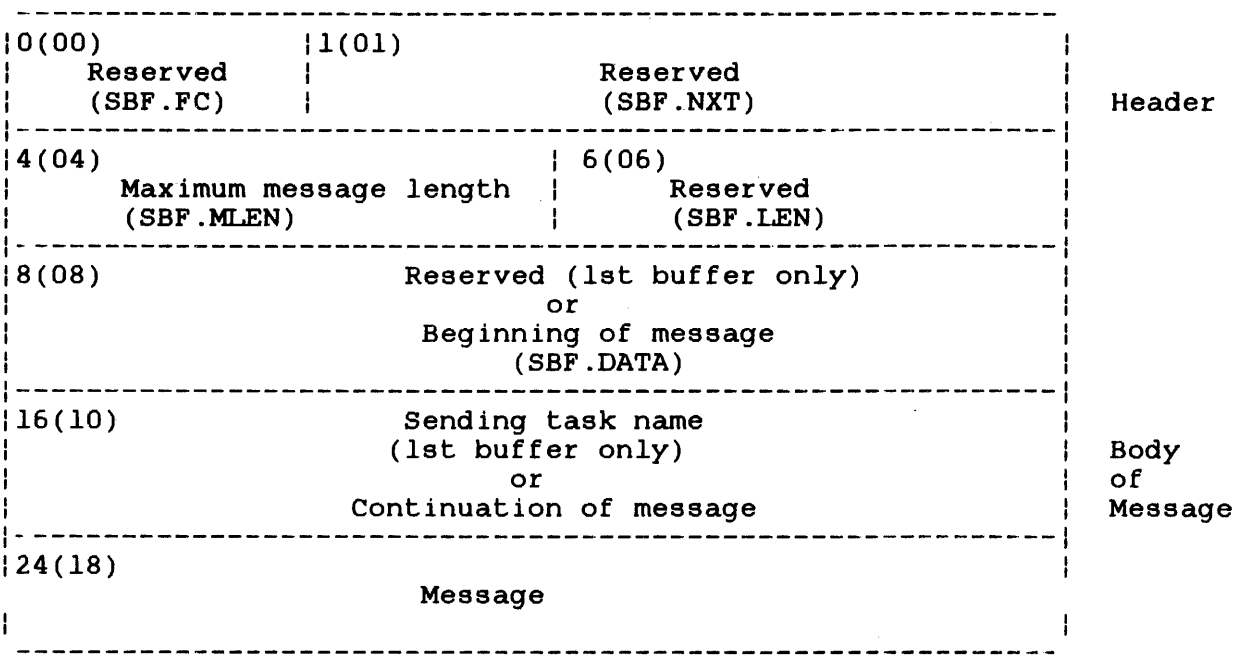


Figure 6-5 Send Data Message Buffer Format for Directed Task

Like the calling task's message buffers, each free message buffer can vary in length as long as the buffer is fullword boundary-aligned and the total length in bytes is an integral number of fullwords.

When initialized, the maximum message length field contains the number of bytes that are available for the body of the message. The remaining fields of the message header are reserved.

The address of each of the free message buffers is placed on a standard Perkin-Elmer circular list established in the task address space. This list is known as the free buffer list queue. The address of the queue is placed in the UDL.SDQ field of the UDL.

When a calling task issues an SVC6 to send a message, the operating system takes the address of the free buffer list queue from the UDL and then takes a free buffer address off the queue. Once the free buffer is found, the operating system sets the reserved field of the message body to blanks and enters the name of the calling task in the sending task name field (left-justified and padded with blanks). After the buffer is filled with the data from the calling task's message buffer, the operating system places the number of bytes of the message body (including the reserved and sending task name fields) into SBF.LEN.

If the entire message has not been transferred, the operating system fetches the address of another free message buffer, places the address of this buffer in the SBF.NXT field and sets the function code. It then begins transferring message data to the free message buffer, the address of which is now specified by SBF.NXT. However, this time the message data begins at the first fullword following the SBF.LEN field. After this buffer is filled, the length of the message body is placed in the SBF.LEN field. The value in this field can never be larger than the maximum message length field.

The operating system continues to fetch and fill the directed task's free buffers until the entire message is tranferred or until no buffers are left on the queue. If the directed task runs out of buffers to hold the message data, the entire message is returned to the calling task buffers. The addresses of the directed task buffers are returned to the queue. The operating system outputs an error message indicating no message was sent.

If the entire message is successfully transferred, the operating system places reason code X'04' and the address of the first filled message buffer on the task queue of the directed task. If the directed task has been properly initialized to receive a task queue trap, the task then branches to a trap-handling routine to process the message. It is good practice to have the trap-handling routine return each message buffer address to the free buffer list queue after the data in that buffer is processed. See the OS/32 Application Level Programmer Reference Manual for more information on preparing directed tasks to handle send data traps.

If the directed task trap structures have not been properly initialized (e.g., no task queue has been established), the message is returned to the calling task.

6.2.7.3 Sample Programs Using SVC6 Send Data Function

The following sample programs demonstrate the data structures used to send a message via the SVC6 send data function.

Sample send data application: Sending Task

```
          MLIBS 8,9
          $SVC6
SEND      DS    SVC6.                START THE PCB
SENDE     EQU   *                    END OF PCB
          ORG   SEND+SVC6.ID         GO INTO THE I.D. FIELD
          DB    C'RECDATA'           STORE THE TASKID
          ORG   SEND+SVC6.FUN        GO TO THE FUNCTION FIELD
          DC    SFUN.DOM!SFUN.DM     SEND DATA:OTHER TASK
          ORG   SEND+SVC6.MSG        GO TO ADDR OF DATA FIELD
          DC    A(MES1)              STORE THE ADDR OF THE 1ST BUFFER
          ORG   SENDE                GO TO THE END OF THE PCB

MES1      EQU   *                    ADDR OF 1ST BUFFER
          DC    Y'20000000'+A(MES2)  1ST BUFF+ADDR OF 2ND
          DC    H'0'                 NOT USED BY CALLER
          DC    H'80'                # OF BYTES WE ARE SENDING
          DS    8                    RESERVED FIELD FOR 1ST BUFFER
          DS    8                    SENDING TASK NAME FOR 1ST BUFFER
          DC    C'THIS IS A MESSAGE FROM ANOTHER TASK.'
          DC    C'WITH THE SEND'
          DC    C'                   ' TOTAL 80 BYTES
          ALIGN 4
MES2      EQU   *                    ADDR OF 2ND BUFFER
          DC    Y'0'+A(MES3)         MIDDLE BUFFER + ADDR OF NEXT BUFF
          DC    H'0'                 NOT USED BY CALLER
          DC    H'80'                # OF BYTES WE ARE SENDING
          DC    C' DATA FUNCTION WE CAN SEND '
          DC    C'VARIABLE LENGTH MESSAGES TO TASKS.  '
          DC    C'THIS EXAMPLE'
          DC    C'    '              TOTAL 80 BYTE BUFFER
          ALIGN 4
MES3      EQU   *                    ADDR OF 3RD BUFFER
          DC    Y'10000000'          LAST BUFFER IN CHAIN CODE
          DC    H'0'                 NOT USED BY CALLER TASK
          DC    H'80'                # OF BYTES WE ARE SENDING
          DC    C' SENT 3 BUFFERS AS ONE MESSAGE '
          DC    C'FROM ONE TASK TO ANOTHER '
          DC    C'AS ONE MESSAGE'
          DC    C'        '
START     EQU   *                    LET'S GO
          SVC   6,SEND               SEND THE DATA
          LH    1,SEND+SVC6.STA      GET THE STATUS
          BNZ   ERROR                AND BRANCH IF AN ERROR OCCURRED
          SVC   3,0                  EOT
ERROR     EQU   *
          SVC   3,1                  RETURN CODE OF 1 ON ERROR
          END   START                TRANSFER ADDR
```

Sample send data application: Receiving Task

```
          MLIBS 8,9
          $VOL
          $SVC1
          $TSW
MBF       STRUC                    STRUCTURE FOR THE MESSAGE BUFFER FORMAT
SBF.FC    EQU   *                  FUNCTION CODE FIELD
SBF.NXT   DS    4                  ADDR OF NEXT BUFFER
SBF.MLEN  DS    2                  MAX LENGTH OF BUFFER
SBF.LEN   DS    2                  LENGTH OF DATA TRANSFER
SBF.DATA  EQU   *                  START OF DATA AREA
          ENDS
MYUDL     DS    256                START OF UDL
MYUDLE    EQU   *                  END OF UDL
          ORG   MYUDL+UDL.TSKQ     GO TO TASK Q ADDR
          DC    A(TRAPQ)           STORE ADDR OF TASK QUEUE
          ORG   MYUDL+UDL.SDQ      GO TO ADDR OF FREE BUFFER LIST
          DC    A(QUEUE)           STORE ADDR OF FREE BUFFER LIST
          ORG   MYUDL+UDL.TSKN     GO TO NEW TSW AREA FOR Q SERVICE
          DC    0                  STATUS OF NEW TSW
          DC    A(QSERVICE)        LOCATION COUNTER OF NEW TSW
          ORG   MYUDLE             GO TO END OF THE UDL
START     EQU   *                  LET'S GO
          LA    1,BUFF1            GET THE ADDR OF BUFF1
          ABL   1,QUEUE            ADD TO BOTTOM OF FREE LIST
          LA    1,BUFF2            GET THE ADDR OF BUFF2
          ABL   1,QUEUE            ADD TO BOTTOM OF FREE LIST
          LA    1,BUFF3            GET THE ADDR OF BUFF3
          ABL   1,QUEUE            ADD TO FREE LIST
          SVC   9,TSW              ENTER TRAP WAIT
*
QSERVICE  EQU   *                  TRAP ROUTINE
          RBL   3,TRAPQ            GET THE REASON CODE
          LR    2,3                STORE IT IN 2
          NI    2,Y'FF000000'      CLEAR THE FIELD
          CI    2,TRC.SDTA         IS IT A SEND DATA REASON CODE
          BNE   ERROR              BRANCH IF NOT
DATA      EQU   *
          L     2,0(3)             GET THE FUNCTION CODE
          NI    2,Y'FF000000'      STRIP OFF THE ADDR
          CI    2,Y'20000000'      IS IT THE FIRST BUFFER
          BE    FIRST
          CI    2,Y'10000000'      IS IT THE LAST BUFFER
          BE    LAST
          CI    2,Y'30000000'      IS IT THE ONLY BUFFER
          BE    ONLY
```

```
NEXT    EQU   *
        LA    8,MBF(3)              GET THE STARTING ADDR OF BUFF
        LHL   9,SBF.LEN(3)          GET THE LENGTH OF DATA TRANSFER
        AR    9,8                   ADD STARTING ADDR
        SIS   9,1                   SUBTRACT ONE FROM ENDING ADDR
        ST    8,WRITE+SVC1.SAD      STORE THE STARTING ADDR
        ST    9,WRITE+SVC1.EAD      ENDING ADDR
        SVC   1,WRITE               WRITE THE NEXT BUFFER
        L     2,0(3)                GET THE ADDR OF THIS BUFFER
        LR    3,2                   STORE IN THREE
        B     DATA                  CONTINUE
FIRST   EQU   *
        LA    8,MBF+16(3)           GET THE STARTING ADDR OF DATA
        LHL   9,SBF.LEN(3)          GET THE LENGTH OF DATA TRANSFER
        AR    9,8                   ADD THE STARTING ADDR
        SIS   9,1                   SUBTRACT ONE FROM ENDING ADDR
        ST    8,WRITE+SVC1.SAD      STORE THE STARTING ADDR
        ST    9,WRITE+SVC1.EAD      STORE THE ENDING ADDR
        SVC   1,WRITE               WRITE THE FIRST BUFFER
        L     2,0(3)                GET ADDR OF FIRST BUFFER
        LR    3,2                   SAVE IN THREE
        B     DATA                  CONTINUE
LAST    EQU   *
        LA    8,MBF(3)              GET THE STARTING ADDR
        LHL   9,SBF.LEN(3)          GET THE # OF BYTES TRANS
        AR    9,8                   ADD TO MAKE ENDING ADDR
        SIS   9,1                   SUBTRACT ONE FROM END
        ST    8,WRITE+SVC1.SAD      STARTING ADDR
        ST    9,WRITE+SVC1.EAD      ENDING ADDR
        SVC   1,WRITE               WRITE OUT THE LAST BUFFER
ONLY    EQU   *
        LA    8,MBF+16(3)           GET THE STARTING ADDR
        LHL   9,SBF.LEN(3)          GET THE # OF BYTES TRANS.
        AR    9,8                   GET AN ENDING ADDR
        SIS   9,1                   SUBTRACT ONE
        ST    8,WRITE+SVC1.SAD      STARTING ADDR
        ST    9,WRITE+SVC1.EAD      ENDING ADDR
        SVC   1,WRITE               WRITE THE ONLY BUFFER
FINI    EQU   *
*       THE OS REMOVES FROM THE TOP OF THE FREE LIST
        LA    1,BUFF1               ADDR OF 1ST BUFF
        ABL   1,QUEUE               ADD TO FREE LIST
        LA    1,BUFF2               ADDR OF 2ND BUFF
        ABL   1,QUEUE               ADD TO FREE LIST
        LA    1,BUFF3               ADDR OF 3RD BUFF
        ABL   1,QUEUE               ADD TO FREE LIST
        LIS   1,0                   GET A ZERO
        ST    1,UDL.TSKO            ZERO THE STATUS
        SVC   9,UDL.TSKO            LOAD A TSW
ERROR   EQU   *
        SVC   3,2                   RETURN CODE OF 2
*
        ALIGN 4
WRITE   DS    SVC1.                 START OF PCB
```

```
WRITEE   EQU   *                      END OF PCB
         ORG   WRITE+SVC1.FC          GO TO THE FUNCTION CODE FIELD
         DB    SV1.WRIT!SV1.WAIT      WRITE AND WAIT
         ORG   WRITE+SVC1.LU          GO TO THE LU FIELD
         DB    2                      LU 2 FOR A WRITE
         ORG   WRITEE                 GO TO END OF PCB
         ALIGN 4
TSW      EQU   *                      NEW TSW
         DC     TSW.WTM!TSW.TSKM!TSW.SDM WAIT,Q TRAP,SEND DTA
         DC    0                      LOCATION COUNTER
         ALIGN 4
QUEUE    DLIST 3                      FREE LIST SIZE
TRAPQ    DLIST 3                      TASK QUEUE SIZE
BUFF1    EQU   *                      1ST BUFF
         DS    4                      FUNCTION CODE AND LINK ADDR
         DC    H'80'                  MAX SIZE OF THIS BUFFER
         DC    H'0'                   # OF BYTES TRANS. SET BY OS
         DS    16                     RESERVED FIELD FOR 1ST BUFFER
         DS    80                     # OF BYTES WE CAN ACCEPT IN BUFF
BUFF2    EQU   *
         DS    4                      FUNCTION CODE AND LINK ADDR
         DC    H'80'                  MAX SIZE OF THIS BUFFER
         DC    H'0'                   # OF BYTES TRANS SET BY OS
         DS    80                     # OF BYTES WE CAN ACCEPT IN BUFF
BUFF3    EQU   *                      3RD BUFF
         DS    4                      FUNCTION CODE AND LINK ADDR
         DC    H'80'                  MAX SIZE OF THIS BUFFER
         DC    H'0'                   # OF BYTES TRANS SET BY OS
         DS    80                     # OF BYTES WE CAN ACCEPT IN BUFF
         END   START                  TRANSFER ADDR
```

6.2.8 Send Message (SFUN.MM) Function

The send message function allows the calling task to send a 64-byte message to the directed task. SVC6 appends the calling task name to the message, finds the address of the receiving task buffer in the UDL of the directed task, fills the receiving buffer, and places the address of that buffer onto the directed task queue.

The required SVC6 parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 11 of the function code field
- Address of message buffer field (SVC6.MSG)

To prepare a directed task to accept the message sent by the SVC6 send message function:

- Allocate message buffers to receive the message. (Use the message buffer format described in Section 6.2.8.1.)
- Write a routine to service task queue traps as described in the OS/32 Application Level Programmer Reference Manual.
- Store the address of the receiving message buffer in the address of the message buffer ring field in the UDL of the directed task.

6.2.8.1 Message Buffers

When allocating receiving message buffers for the send message function, use the buffer format shown in Figure 6-6.

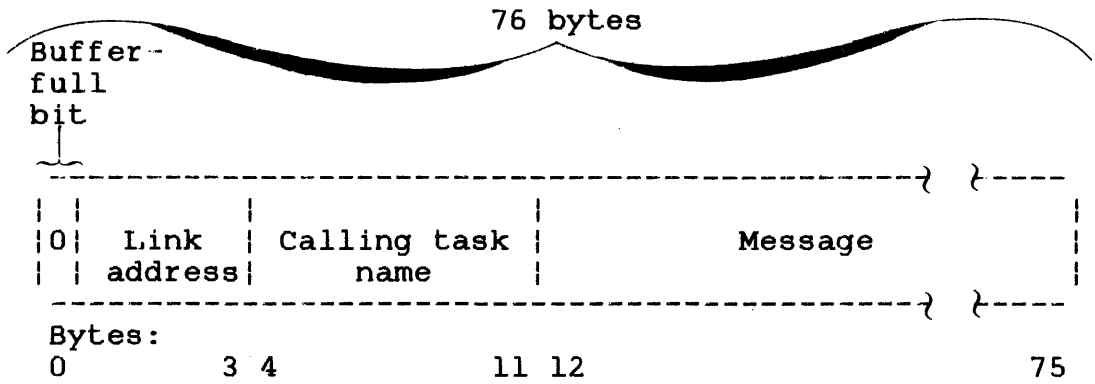


Figure 6-6 Message Buffer Format for Directed Task

This message buffer must be 76 bytes long and aligned on a fullword boundary-aligned. A description of each field in the message buffer format follows.

Fields:

Buffer-full bit	is a 1-bit field indicating whether or not the buffer can receive the message being sent from the calling task.

If bit 0 equals 0, the buffer is available to receive the message.

If bit 0 equals 1, the buffer is full and the message is rejected. After the message sent by the calling task is stored in the message buffer, the system sets the buffer-full bit to 1 to indicate the message buffer is full. After the directed task processes the message, the user must reset the buffer-full bit to 0 to indicate that it is available to receive the next message.

Link address — is a 4-byte field containing the address of the subsequent message buffer to receive the next message sent by the calling task. If this field contains an invalid address, the call is rejected.

Calling task name — is an 8-byte field receiving the calling task's name from the system.

Message — is a 64-byte field receiving the message sent by the calling task.

Using the Link address field, the user can construct the following structures from the basic message format:

- Single buffer ring
- Single buffer chain
- Multiple buffer ring
- Multiple buffer chain

A single buffer ring consists of one buffer as shown in Figure 6-7. The buffer-full bit initially should be set to 0, and the link address field should contain the buffer's own starting address (points to itself). When a message is sent to a single buffer ring, the system sets the buffer-full bit to 1. All subsequent messages are rejected until the user resets the buffer-full bit to 0.

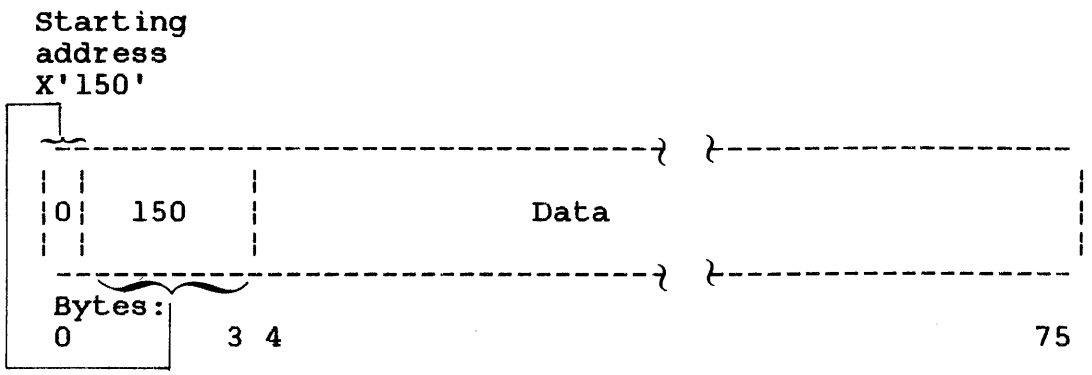


Figure 6-7 Single Buffer Ring

A single buffer chain consists of one buffer as shown in Figure 6-8. The buffer-full bit should initially be set to 0, and the link address field should contain zeros (terminating the chain). When a message is sent to a single buffer chain, the system sets the buffer-full bit to 1 and stores the link address field contents into the address of the message ring field of the UDL of the directed task. All subsequent messages are rejected until the user stores the empty buffer address into the UDL address of the message ring field and resets the buffer-full bit of the empty message buffer to 0.

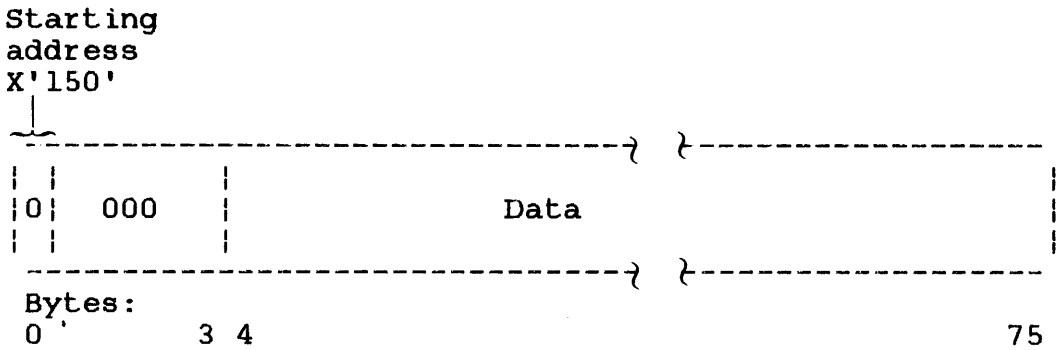


Figure 6-8 Single Buffer Chain

A multiple buffer ring consists of a variable, unlimited number of buffers. Each buffer-full bit should initially be set to 0, and each link address field should contain the address of a subsequent buffer. The last buffer's link address field should contain the first buffer's address (forming a ring). When a message is sent to a multiple buffer ring, the first buffer, pointed to by the address stored in the UDL address of the message ring field, receives the message if the buffer-full bit is 0.

The system then stores the contents of the first buffer link address field into the UDL address of the message ring field. That UDL field now points to the second buffer in the ring. If the calling task sends another message, the second buffer receives the message if the buffer-full bit is 0.

The system stores the contents of the second buffer link address field into the UDL address of the message ring field, which now points to the third buffer in the ring. When the last buffer in the ring receives a message and the contents of the link address field are stored into the UDL, that UDL field points to the first message buffer in the ring. If the calling task sends another message, the first buffer receives the message if the buffer-full bit is reset to 0; otherwise, the message is lost (see Figure 6-9).

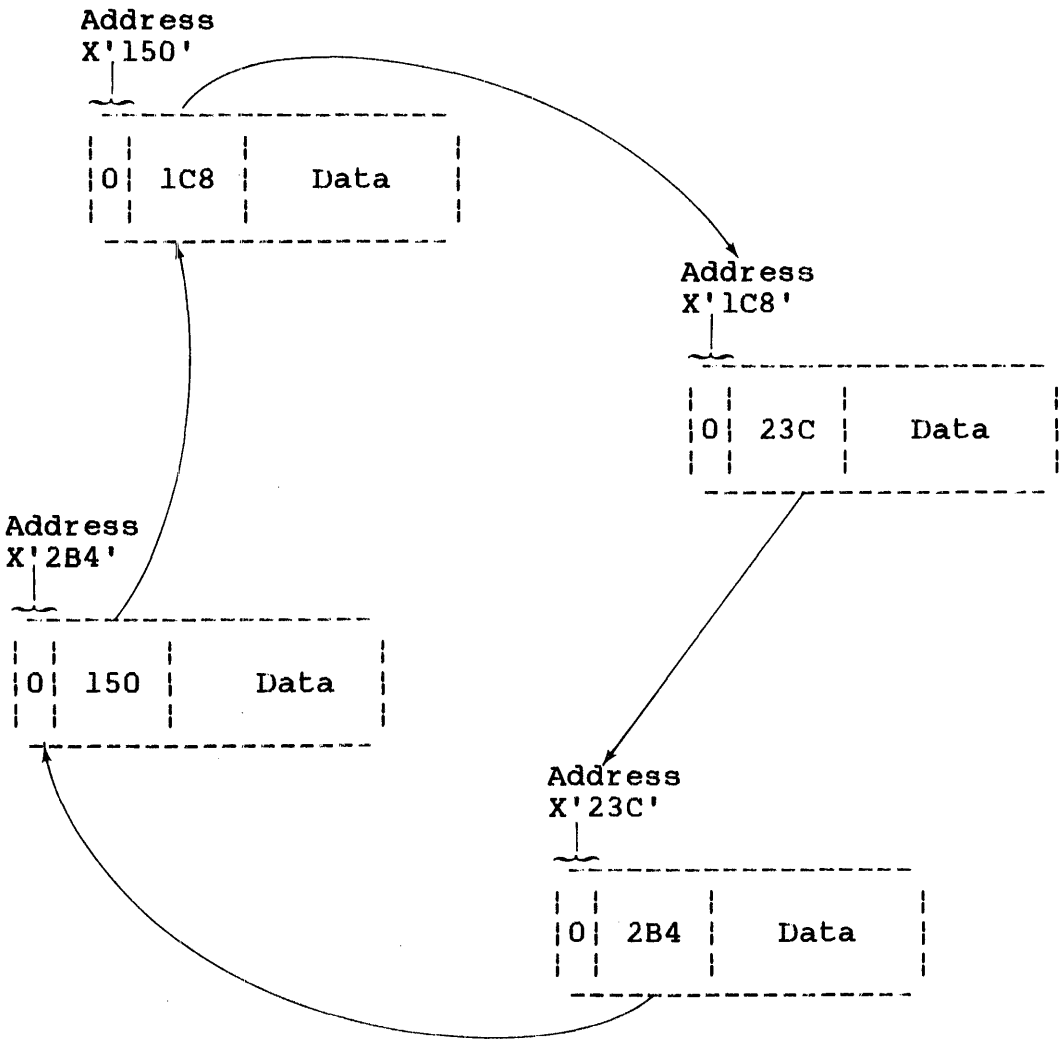


Figure 6-9 Multiple Buffer Ring

A multiple buffer chain consists of a variable, unlimited number of buffers. Each buffer-full bit should initially be set to 0, and each link address field should contain a subsequent buffer address. The last buffer link address field should contain zeros (terminating the chain). When a message is sent to a multiple buffer chain, the first buffer, pointed to by the UDL address of the message ring field, receives the message if the buffer-full bit is 0.

The system then stores the contents of the first buffer link address field into the UDL address of the message ring field. That UDL field now points to the second buffer in the chain. If the calling task sends another message, the second buffer receives the message if the buffer-full bit is 0.

The system then stores the contents of the second buffer link address field into the UDL address of the message ring field. That UDL field points to the third buffer in the chain. When the last buffer in the chain receives a message and the system stores the contents of the link address field into the UDL, all subsequent messages are rejected until the user stores an empty buffer address into the UDL address of message ring field and resets the buffer-full bit of that message buffer (see Figure 6-10).

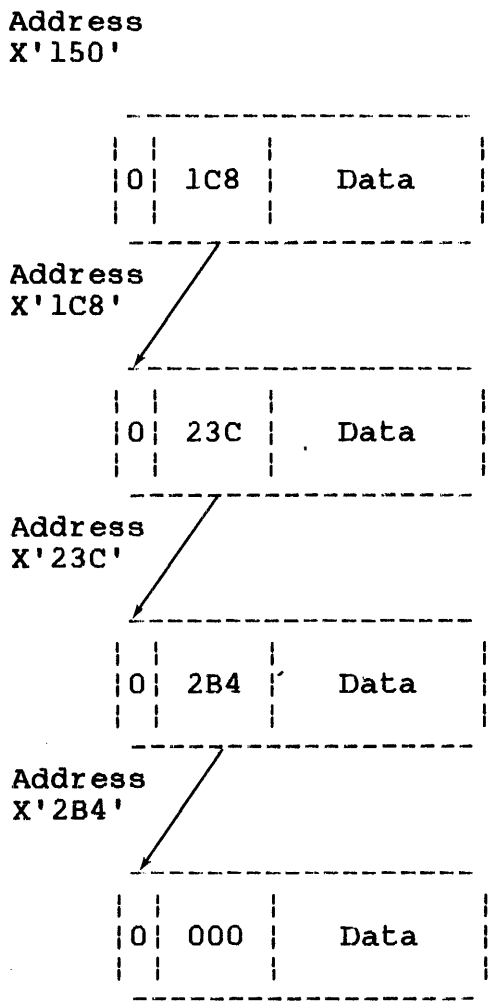


Figure 6-10 Multiple Buffer Chain

The following sample programs demonstrate the data structures used to send a message via the SVC6 send message function.

Sample send message application: Sending task

```
SEND     PROG  SVC6 EXAMPLE - SEND MESSAGE
*
*  This task sends a message to task RECEIVE.
*
*
         NLSTM
         NLSTU
         $SVC6
         $REG$

SEND     EQU   *
         SVC   6,SVC6                SEND THE MESSAGE
         LB    R15,SVC6+SVC6.STA+1 GET ERROR STATUS
         SVC   3,0(R15)              END OF TASK


         ALIGN 4
SVC6     DS    SVC6.                 RESERVE SPACE FOR SVC 6 PBLK
         ORG   SVC6+SVC6.ID          NAME OF TASK MESSAGE IS SENT TO
         DC    C'RECEIVE '
         ORG   SVC6+SVC6.FUN         SEND MESSAGE TO ANOTHER TASK
         DC    SFUN.DOM!SFUN.MM
         ORG   SVC6+SVC6.MSG         ADDRESS OF MESSAGE TO BE SENT
         DC    A(MESSAGE)


MESSAGE  DC    C'Message from SEND to RECEIVE '
         DC    C'                             '
         DC    C'    '
         END   SEND
```

Sample send message application: Receiving task

```
RECEIVE  PROG  SVC6 EXAMPLE - RECEIVE MESSAGE
*
*
*  This task receives a message sent to it by another
*  task.  The message is logged to the console device.
*
*
         NLSTM
         NLSTU
         $UDL$
         $REG$
```

```
*
*
*  Set up UDL and link message buffers into a message ring.
*
*
RECEIVE  EQU   *
         SVC   2,FETCHPTR          GET ADDRESS OF UDL IN R01
         LI    R14,TSW.PMM         ALLOW MESSAGES TO BE QUEUED
         LA    R15,QSERVICE        ADDRESS OF QUEUE SERVICE ROUTINE
         STM   R14,UDL.TSKN(R1)    SAVE TASK QUEUE NEW TSW
         LA    R15,TSKQ            ADDRESS OF TASK QUEUE
         ST    R15,UDL.TSKQ(R1)
         LA    R15,MESSQ           ADDRESS OF MESSAGE BUFFER RING
         ST    R15,UDL.MSGR(R1)

         LHI   R15,NMESS           NUMBER OF MESSAGES IN RING
         LA    R14,MESSQ           HEAD OF MESSAGE RING
LINKRING EQU   *
         SIS   R15,1
         BNP   LINKDONE
         LA    R13,76(R14)         GET ADDRESS OF NEXT BUFFER
         ST    R13,0(R14)          LINK NEXT TO CURRENT
         LR    R14,R13             CURRENT IS NEXT
         B     LINKRING            CONTINUE LINKING OF RING


LINKDONE EQU   *
         LA    R13,MESSQ           ADDRESS OF FIRST MESSAGE
         ST    R13,0(R14)          LINK FIRST TO LAST

         SVC   9,TRAPENA           ENABLE RECEIVE OF MESSAGES
*
*
*  Service task queue traps
*
*
QSERVICE EQU   *
         REL   R2,TSK              AN ITEM ON THE TASK QUEUE?
         BO    QEMPTY              NO - ENTER TRAP WAIT
         LA    R15,0(R2)           GET PARAMETER
         SRL   R2,24               ISOLATE REASON CODE
         CLHI  R2,6                MESSAGE RECEIVED?
         BNE   QSERVICE            NO - IGNORE IT
         LA    R14,4(R15)          SKIP OVER MESSAGE LINK
         ST    14,LOGMESS+4        ADDRESS OF MESSAGE TO BE LOGGED
         SVC   2,LOGMESS           LOG SENDER ID AND MESSAGE
         L     R0,0(R15)           RESET MESSAGE ACTIVE FLAG
         NI    R0,Y'7FFFFFFF'
         ST    R0,0(R15)
         L     R0,12(R15)          GET FIRST FOUR BYTES OF MESSAGE
         CLI   R0,C'STOP'          IS IT "STOP"?
         BNE   QSERVICE            NO - CHECK FOR MORE MESSAGES
         SVC   3,0                 YES - STOP TASK
```

```
QEMPTY   EQU   *
         SVC   9,TRAPWAIT          ENTER TRAP WAIT


         ALIGN 4
TRAPENA  DC    TSW.PMM,0


TRAPWAIT DC    TSW.WTM!TSW.TSKM!TSW.PMM,0

FETCHPTR DB    0,5,0,R01           FETCH UDL POINTERS

LOGMESS  DB    X'40',7
         DCX   72
         DC    0

TSKQ     DLIST 5                   TASK QUEUE

NMESS    EQU   3                   NUMBER OF MESSAGE BUFFERS
MESSQ    DS    76*NMESS            MESSAGE BUFFERS
         END   RECEIVE
```

6.2.9 Queue Parameter (SFUN.QM) Function

The queue parameter function adds the user-specified parameter, located in the task queue parameter field of the parameter block, to the directed task's queue. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 12 of function code field
- Task queue parameter field (SVC6.PAR)

Before the directed task can receive the parameter sent from the calling task, the directed task must prepare to service traps as described in the OS/32 Application Level Programmer Reference Manual.

If the directed task's queue is full, the call is rejected, the parameter is lost, and an error code is stored into the error status field of the caller's parameter block.

6.2.10 Change Priority (SFUN.PM) Function

The change priority function changes the directed task's current priority to the user-specified priority located in the parameter block change priority field. It then stores the previous priority value of the directed task into the current priority field of the parameter block. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 13 of the function code field
- Change priority field (SVC6.PRI)

If the change priority value specified in the parameter block is greater than the maximum priority value established by Link, that maximum priority is used. However, if the change priority value the user specified is outside the range of 10 through 249, the call is rejected, and an error code is stored in the parameter block error status field (see Table 6-5).

6.2.11 Send Logical Unit (lu) (SFUN.XSM) Function

The send lu function assigns to the specified lu of the directed task the device or file currently assigned to the specified lu of the calling task and then closes the lu assigned to the calling task. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 14 of the function code field
- Calling lu field (SVC6.CLU)
- Receiving lu field (SVC6.DLU)

Before the directed task accepts the lu that the calling task sends:

- the directed task's lu must not be assigned, and
- the directed task must be in either a dormant or paused wait state or be suspended by an SVC6.

6.2.12 Receive Logical Unit (lu) (SFUN.XRM) Function

The receive logical unit function assigns to the specified lu of the calling task the device or file currently assigned to the specified lu of the directed task, and then closes the lu assigned to the directed task. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 15 of the function code field
- Calling lu field (SVC6.CLU)
- Receiving lu field (SVC6.DLU)

Before the calling task accepts the lu, the directed task sends:

- the calling task's lu must not be assigned, and
- the directed task must be in either a dormant or paused wait state or be suspended by an SVC6.

6.2.13 Connect (SFUN.OM) Function

The connect function connects the trap generating device specified in the device mnemonic field of the parameter block to the directed task. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 16 of the function code field
- Device mnemonic field (SVC6.DMN)
- Task queue parameter field (SVC6.PAR)

Before the connection is made:

- The user-specified device must be a trap generating device.
- The device must not be currently connected to the directed task or any other task; it can be connected to only one task at a time. However, a task can be connected to more than one trap generating device at the same time.
- The directed task must be prepared as described in the OS/32 Application Level Programmer Reference Manual if traps are to be serviced as they occur.

When the connection is made and the thaw function is specified, an interrupt occurs, and the user-specified parameter located in the task queue parameter field of the parameter block is placed on the directed task queue with a reason code of 0. The connect function does not enable interrupts. The operating system provides capabilities to connect APUs as a trap generating device to a task. The actual APU signals are defined in the OS/32 System Level Programmer Reference Manual.

6.2.14 Thaw (SFUN.TM) Function

The thaw function enables interrupts from the specified trap generating device connected to the directed task. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 17 of the function code field
- Device mnemonic field (SVC6.DMN)

Before this function is effected, the task should be prepared to handle traps as described in the OS/32 Application Level Programmer Reference Manual.

When the thaw function is executed, the system first ensures that the trap generating device is connected to the directed task specified in the parameter block; it then enables interrupts. Interrupts are disabled when the directed task terminates or if an unconnect or freeze function is specified. If a thaw function is executed when interrupts are already enabled, this call has no effect. Note that an APU can be treated as a pseudo trap generating device.

6.2.15 Sint (SFUN.IM) Function

The sint function simulates an interrupt from the specified trap generating device connected to the directed task only if the thaw function was specified. If interrupts are disabled, this call has no effect. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 18 of the function code field
- Device mnemonic field (SVC6.DMN)

When the simulate interrupt function is executed, the system first ensures that the trap generating device is connected to the directed task specified in the parameter block; it then simulates an interrupt from the specified device. Note that an APU can be treated as a pseudo trap generating device.

6.2.16 Freeze (SFUN.FM) Function

The freeze function disables interrupts from the specified trap-generating device connected to the directed task. The required parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 19 of the function code field
- Device mnemonic field (SVC6.DMN)

When the freeze function is executed, the system first ensures that the trap-generating device is connected to the directed task specified in the parameter block; it then disables interrupts from the specified device. When the freeze function disables interrupts, the trap-generating device and directed task remain connected, but all generated interrupts are lost. If interrupts are already disabled, this call has no effect. Note that an APU can be treated as a pseudo trap-generating device.

6.2.17 Unconnect (SFUN.UM) Function

The unconnect function disconnects the specified trap-generating device from the directed task. The required parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 20 of the function code field
- Device mnemonic field (SVC6.DMN)

When the unconnect function is executed, the system first ensures that the trap-generating device is connected to the directed task specified in the parameter block; it then disables all interrupts and disconnects the specified device from the directed task. The device can now be connected to another task. Note that an APU can be treated as a pseudo trap-generating device.

6.2.18 Assign Logical Processing Unit (LPU) Function (SFUN.LPU) Function

The assign LPU function assigns the directed task to a new LPU number. This assignment has no effect until the directed task is transferred by the task dispatcher. The required parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 21 of the function code field
- LPU assignment field (SVC6.LPU)

If task mapping to a waiting APU has been changed, the wait condition is removed and the APU is restarted.

6.2.19 Transfer to Logical Processing Unit (LPU) (SFUN.TL) Function

The transfer to LPU function makes the directed task LPU-directed and transfers the directed task to its assigned LPU the next time the task is dispatched, provided that all requirements for transfer are met. See the OS/32 System Level Programmer Reference Manual for more information on the task dispatcher. If this function is self-directed, it duplicates the function of the RSCH 2 instruction. If a calling task executing on the CPU directs this function to itself, the task is made LPU-directed, and the calling task is dispatched to its assigned LPU. If the LPU is mapped to the CPU, no transfer occurs. If the LPU is mapped to an APU, a transfer to the APU occurs.

The required parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 22 of the function code field.

NOTE

If both bits 22 (SFUN.TL) and 23 (SFUN.TC) are set in the same SVC6 parameter block, only bit 23 is recognized when the call is complete.

6.2.20 Transfer to Central Processing Unit (CPU) (SFUN.TC) Function

The transfer to CPU function makes the directed task CPU-directed. As a result, the next time the task is dispatched from the CPU ready queue, it will execute on the CPU regardless of its LPU assignment and eligibility. If the calling task directs this function to a task that is active or ready on an APU, the LPU-directed status is reset after the task is returned to the CPU for any reason.

If a calling task executing on the CPU directs this function to itself, it duplicates the function of the RSCH 0 instruction with the following exception.

If the task's TCB specified a trap block, which has a nonzero wait bit in the SVC new program status word (PSW),the APU halts and waits for the task to resume execution on the APU. The APU remains halted while the task executes on the CPU until:

- The task is transferred back to the LPU.
- The task is assigned to a different LPU.
- The task goes to end of task or is cancelled.
- LPU mapping for the task's LPU is changed.

If a calling task operating on the APU directs this function to itself, the calling task is transferred to the CPU. Normally, APU processing continues after the task is transferred unless the wait bit in the SVC new PSW field of the APU trap block has been set. In this case, APU processing is explicitly suspended while the task executes on the CPU until:

- The task is explicitly transferred back to the APU via SVC6.
- The task is assigned to a different APU through an LPU assignment.
- The task is cancelled or goes to end of task.
- LPU mapping for the task's LPU is changed.

See the OS/32 Application Level Reference Manual for more information on setting the trap block to suspend APU processing. The required parameter block fields for this function are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 23 of the function code field

NOTE

If both bits 22 (SFUN.TL) and 23 (SFUN.TC) are set in the same SVC6 parameter block, only bit 23 is recognized when the call is completed.

6.2.21 Release (SFUN.RM) Function

The release function releases a directed task currently suspended by a previous SVC6 by taking it out of a wait state. Once released, the task continues executing with the instruction following the instruction executed before the task was suspended if the task is not in another wait state at this time. The required parameter block fields for this function are:

- Task name field (SVC6-ID)
- Bits 0, 1 and 24 of the function code field

The SVC6 release function can be used by a system task (.CMDP, .CSL, .MTM and .SPL) to remove another task from a suspended state. After the task is released, it continues execution at the location specified in the SVC6.SAD field of the SVC6 parameter block. If this SVC is used by other than a system task, the continuation address (SVC6.SAD) is ignored. Figure 3-6 shows the parameter block format and coding for the SVC6 release function for system tasks.

6.2.22 Nonresident (SFUN.NM) Function

The nonresident function makes the directed task nonresident regardless of the Link options specified. When a nonresident task goes to end of task, it is removed from the system. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 25 of the function code field

6.2.23 Rollable (SFUN.RLM) Function

The rollable function makes the directed task rollable. The directed task must have been linked as a rollable task. If this function is directed to a task linked as nonrollable, an error status is returned. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 26 of the function code field

6.2.24 Nonrollable (SFUN.NRM) Function

The nonrollable function prevents the directed task from being rolled. The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1 and 27 of the function code field

If both the rollable and nonrollable functions are specified, only the nonrollable function is recognized.

6.2.25 Start (Bit Positions 29, 30, 31) Function

The start function starts execution of the directed task. This call is rejected if it is self-directed. Four methods of starting are:

- Start (bit setting equals 010)
- Start with start options (bit setting equals 011)
- Delay start (bit setting equals 100 or 110)
- Delay start with start options (bit setting equals 101)

The required parameter block fields are:

- Task name field (SVC6.ID)
- Bits 0, 1, 29, 30 and 31 of the function code field
- Address of start options field (SVC6.SOP) (only required when start with start options or delay start with start options is specified in the function code)
- Increment of time field (SVC6.TIM) (only required when delay start or delay start with start options is specified)
- Count field (SVC6.CNT) (only required when delay start or delay start with start options is specified)
- Starting address of directed task field (SVC6.SAD)

Before the start function is executed, the directed task must be:

- loaded or present in memory, and
- in a dormant or console wait state.

6.2.26 Start Function for SVC6 (SFUN.SIM) Function

When this function is specified, execution of the directed task is started at the address in the parameter block starting address of the directed task field. However, if the user-specified starting address is 0, the directed task is started at the default start address specified by Link. If the user-specified starting address is outside the established task boundaries, this call is rejected, and an error code is stored in the parameter block error status field.

6.2.27 Start Function with Start Options for SVC6 (SFUN.SOM) Function

When this function is specified, the start options, optionally specified in certain language and utility programs at execution time, are also included as run-time information when the directed task starts execution. When the start function is executed, the start options located at the address specified in the parameter block are stored into the directed task UTOP area. If sufficient memory is not available between UTOP and CTOP, this call is rejected and an error code is stored in the parameter block error status field. The task should then be reloaded into a larger segment using the extended load option segment size increment field.

The user-specified start options must be located on a fullword boundary. The maximum length of the start options are defined at sysgen through the CMDLENGTH option. If the length of the start options is greater than that specified at sysgen or a carriage return (CR) is present within the start options, only those characters up to the maximum number or the CR are stored in the task UTOP area.

Since the address of the start options field is also the address of the message buffer field in the parameter block, this field's contents are always assumed to be the start option address when the start function is specified.

6.2.28 Delay Start Function for SVC6 (SFUN.SDM) Function

When this function is specified, the directed task starts execution after a user-specified interval located in the parameter block increment of time and count fields elapses. This can be specified as a time-of-day or interval-timing interval.

When this start function is executed for the directed task, bytes 192 through 251 of the directed task's UDL are used by the operating system for SVC6 delay start function use.

When this start function is executed, the directed task is immediately placed into a time wait state. When the user-specified interval elapses, the directed task starts execution.

6.2.29 Delay Start Function with Start Options for SVC6 (SFUN.SDM, SFUN.SOM)

When this function is specified, the directed task starts execution after a user-specified interval located in the parameter block increment of time and count fields elapses. The interval can be specified as a time-of-day or interval-timing interval.

When this start function is executed for the directed task, bytes 192 through 251 of the UDL are used by the operating system for SVC6 delay start function use.

When the start function is executed, the start options located at the address specified in the parameter block are stored into the directed task UTOP area, and the directed task is immediately placed into a time wait state. If sufficient memory is not available between UTOP and CTOP, this call is rejected, and an error code is stored in the parameter block error status field. The task should then be reloaded into a larger segment using the extended load option segment size increment field (see Section 6.2.4.2).

The user-specified start options must be located on a fullword boundary. The maximum length of the start options is defined at sysgen through the CMDLENGTH option. If the length of the start options is greater than that specified at sysgen or a CR is present within the start options, only those characters up to the maximum number or the CR are stored in the task UTOP area. Since the address of the start options field is also the address of the message buffer field in the parameter block, this field's contents are always assumed to be the address of the start options when the start function is specified. When the user-specified interval elapses, the directed task starts execution.

6.2.30 Wait Status Field (SVC6.TST)

The wait status is sent to the wait status field in the parameter block each time an SVC6 is executed.

If the calling task wants to check the wait status of the directed task, an SVC6 should be executed with bits 0 and 1 of the function code set to 10 and the remaining bits set to 0. This operation also causes the current priority field of the directed task to be returned to the current priority field in the parameter block. Table 6-4 lists the wait status bit definitions.

TABLE 6-4 WAIT STATUS BIT DEFINITIONS

BIT POSITION	WAIT STATUS FIELD MASK	MEANING
0 (IO)	X'8000'	I/O queue wait
1 (CN)	X'4000'	Connection wait
2 (CW)	X'2000'	Console wait (task paused)
3 (LW)	X'1000'	Load wait; calling task waiting for receiving task to be loaded
4 (DM)	X'0800'	Dormant; task not started or at end of task
5 (TW)	X'0400'	Trap wait
6 (TD)	X'0200'	Time-of-day wait
7 (TK)	X'0100'	Task suspended
8 (TM)	X'0080'	Interval wait
9 (TR)	X'0040'	Terminal wait
10 (RO)	X'0020'	Roll pending wait
11 (II)	X'0010'	Intercept initialization
12 (IT)	X'0008'	Intercept termination
13 (CO)	X'0004'	Connection wait
14 (AC)	X'0002'	Accounting wait
15	X'0001'	Reserved for future use

6.2.31 Error Codes (SVC6.STA)

If an error occurs, execution of the current SVC6 function stops, and any other functions specified in the function code to the right of the current function are not executed. The position of the function code bit, which indicates the function being executed when the error occurred, is stored in bits 0 through 7 of the parameter block error status field. The bit position value ranges from 0 to 31. The error code indicating the error type is stored in bits 8 through 15 of the parameter block error status field shown in Figure 6-11. Table 6-5 lists SVC6 error codes.

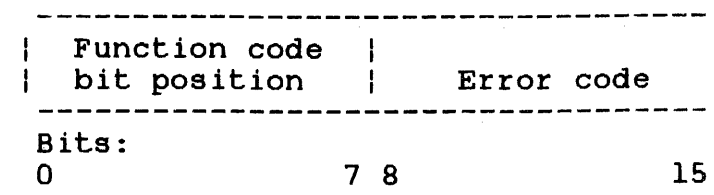


Figure 6-11 Error Status Field

TABLE 6-5 SVC6 ERROR CODES

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
0	All	No error occurred. All requested functions terminated normally.
1	All	Syntax error present in parameter block task name field. This error does not include self-directed calls.
2	All	Illegal function code
3	6 (L)	Directed task is already loaded into memory.
4	All except 6 (L)	The specified directed task is not present in the calling task environment.
	6 (L)	The directed task is not present, but the calling task has the SVC6 intertask control function disabled.
5	13 (P)	The specified priority is outside of the range of 10 through 249.
6	6 (L)	The directed task requires floating point facilities that are not sysgened into the system.

TABLE 6-5 SVC6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
7	9 (S) 14 (B) 15 (V) 29 30 (A) 31	The specified directed task is dormant, paused or suspended. The specified directed task is not dormant, paused or suspended.
8	11 (M) 29 30 (A) 31	The message is not fullword boundary-aligned, or an invalid starting address was specified for a directed task.
9	All	The calling task cannot execute SVC6 control or communication functions.
A (10)	29 30 (A) 31	The values specified for the increment of time and count fields are invalid.
B (11)	10 (SD) 11 (M)	The calling task message was not sent to the directed task.
C (12)	10 (SD) 11 (M) 12 (Q)	Task queue service in the directed task TSW is disabled. The directed task queue is full. The directed task has no queue.
D (13)	16 (O) 17 (T) 18 (I) 19 (F) 20 (U)	The device mnemonic specified in the parameter block does not exist in the system.
E (14)	16 (O) 17 (T) 18 (I) 19 (F) 20 (U)	The device mnemonic specified in the parameter block is not a connectable device.
F (15)	16 (O)	The device mnemonic specified in the parameter block is busy and cannot be connected.

TABLE 6-5 SVC6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
10 (16)	17 (T) 18 (I) 19 (F) 20 (U)	The device mnemonic specified in the parameter block is not connected to the specified directed task.
11 (17)	6 (L)	The lu specified in the lu to load task field of the parameter is invalid.
12 (18)	14 (B) 15 (V)	The lu the calling task sends or receives is greater than the maximum allowed value.
13 (19)	14 (B) 15 (V)	The directed task is currently assigned to an lu during a send lu operation.
14 (20)	14 (B) 15 (V)	The calling task is currently assigned to an lu during a receive lu operation.
16 (22)	29 30 (A)	The specified directed task to be started is currently rolled out.
17 (23)	26 (Y)	The directed task did not specify the roll option by Link and, therefore, cannot be rolled out.
18 (24)	29 30 (A) 31	There is insufficient room between the task UTOP and CTOP to store the task-specified start options.
19 (25)	18 (I)	An interrupt cannot be simulated on the specified device.
1B (27)	6 (L)	Loading the direct task exceeds the maximum number of sysgen-established tasks that can be present in the system at one time.
21 (23)	6 (L)	An error occurred while loading a pure segment.
42 (66)	6 (L)	The run-time library (RTL) or a TCOM required by the directed task is not present at load time.

TABLE 6-5 SVC6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
43 (67)	6 (L)	The calling task specified load options and the directed task specified Link options are not the same.
44 (68)	6 (L)	The LIB format is invalid.
45 (69)	6 (L) 14 (B) 15 (V) 29 30 (A) 31	Insufficient system space exists to load or start the directed task. There is insufficient system space in the directed task to accept the lu of the calling task being sent.
46 (70)	6 (L)	Attempt was made to load tree-structured overlays from a device that does not support random access.
47 (71)	6 (L)	System does not support loading of tree-structured overlays.
48 (72)	6 (L)	Data in the ODT of a three-structured overlay is invalid.
49 (73)	6 (L)	Memory does not have a large enough area into which the directed task can be loaded. The roll option was not specified as a Link option.
4A (74)	6 (L)	An error occurred while mapping a shared segment. Previously mapped or shared segment table was full.
50 (80)	6 (L)	The allocation of or assignment to the specified roll file is invalid, and the task cannot be loaded.
51 (81)	6 (L)	An I/O error occurred when the directed task was rolled out (written) to the roll volume; it cannot be loaded back into memory.

TABLE 6-5 SVC6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
52 (82)	6 (L)	The physical size of a sharable segment was smaller than the minimum size required.
53 (83)	6 (L)	The access privileges of a sharable segment were incompatible with those requested by the task.
54 (84)	21 (AP)	The LPU number is outside the range specified by the MAXLPU parameter at sysgen.
55 (85)	21 (AP) 23 (IC)	The directed task is an APU-only task and cannot be transferred to the CPU.
80-FF (128-255)	6 (L)	An I/O error occurred when the directed task was being loaded (read) into memory. An SVC1 error occurred.

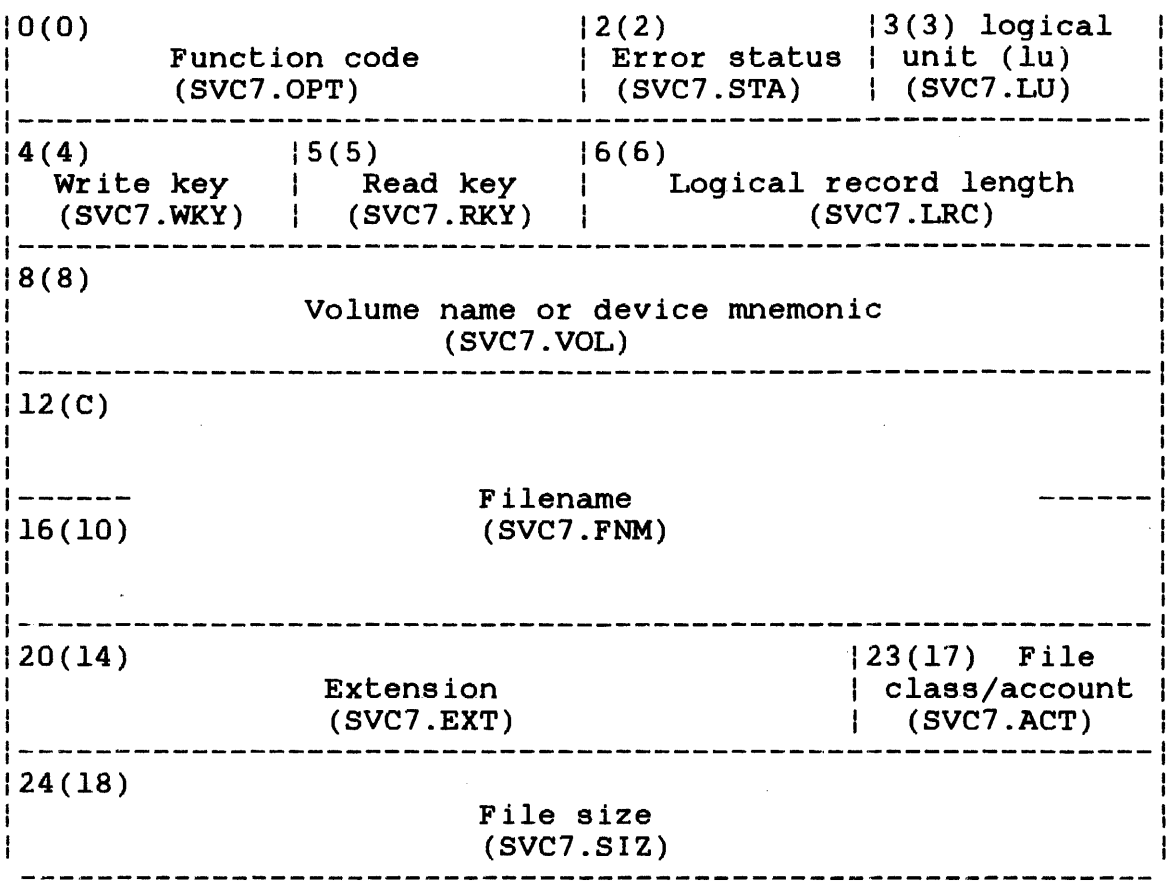
The calling task can check the parameter block for functions the directed task executed before the error occurred and for functions that were not executed.

CHAPTER 7
FILE HANDLING SERVICES SUPERVISOR CALL 7 (SVC7)

7.1 INTRODUCTION

SVC7 provides file and device handling functions supported by the file manager and the data communications subsystem. These functions are accomplished through the SVC7 parameter block shown in Figure 7-1. For a description of the OS/32 file management services, see the OS/32 Application Level Programmer Reference Manual and the OS/32 Basic Data Communications Reference Manual.

7.2 SVC7: FILE HANDLING SERVICES




```
          SVC   7,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    X'function code'
          DS    1
          DB    lu
          DB    'write key'
          DB    'read key'
          DC    H'record length'
          DC    C'4-character volume name or device
                mnemonic'
          DC    C'8-character filename'
          DC    C'3-character extension'
          DB    C'file class'
          DC    F'file size'
```

Figure 7-1 SVC7 Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary-aligned and located in a task-writable segment. A description of each field in the parameter block follows:

Fields:

Function code (SVC7.OPT)	is a 2-byte field that contains the hexadecimal number indicating the function to be performed.
Error status (SVC7.STA)	is a 1-byte field that receives the appropriate error code when an error occurs while executing SVC7. If no error occurs, a value of 0 is stored in this field.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit used for all SVC7 functions (except the allocate and delete functions).
Write key (SVC7.WKY)	is a 1-byte field that contains a hexadecimal number indicating the write protection keys for direct access and data communications files and devices when the allocate, assign, reprotect and delete functions are executed.
Read key (SVC7.RKY)	is a 1-byte field that contains a hexadecimal number indicating the read protection keys for direct access and data communications files and devices when the allocate, assign, reprotect and delete functions are executed. When executing the SVC7 fetch attributes function, the device and file attributes are stored in the write and read key fields of the parameter block.
Logical record length (SVC7.LRC)	is a 2-byte field that contains a decimal number indicating the logical record length for indexed files, nonbuffered indexed files, or buffered logical terminal manager (communications). When executing a fetch attributes function, this field receives a device physical record length.

Volume name or device mnemonic (SVC7.VOL)

is a 4-byte field that contains the ASCII code indicating the volume name of a direct access device, the device mnemonic of a nondirect access device or the name of the data communications access line, when the allocate, assign, delete and fetch attributes functions are executed.

Filename (SVC7.FNM)

is an 8-byte field that must contain the ASCII code indicating:

- A filename on a direct access device when the allocate, assign, rename and delete functions are executed; a filename is not required for nondirect access devices.
- The buffered logical terminal described by the line control block (LCB) that is being allocated or assigned

When executing a fetch attributes function, this field receives the filename from the direct access or data communications device currently assigned to the lu specified in the parameter block. If it is a nondirect access device, this field is blank.

Extension (SVC7.EXT)

is a 3-byte field that contains the ASCII code indicating further identification of the filename or the file type (.CAL, .OBJ, .TSK, .CSS) on direct access devices.

File class/ account (SVC7.ACT)

is an optional 1-byte field that contains the account number or class to which the file is allocated. If SVC7 is issued by an executive task (e-task) or a user task (u-task) that was link-edited with the ACPRIVILEGE option, an account number can be specified in this field.

NOTE

To specify a file with an account number, the file descriptor (fd) must be packed into the SVC7 parameter block using SVC2 code 16 (see Section 3.14.9). The account number can range from 0 to 65,535.

If SVC7 is issued by a u-task that was link-edited with the NACPRIVILEGE option, the file class is specified as follows:

- /P indicates the file is allocated under a private account.
- /G indicates the file is allocated under a group account.
- /S indicates the file is allocated under a system account.

See the OS/32 Link Reference Manual for more information on the account privileges task option.

File size (SVC7.SIZ) is a 4-byte field that contains a hexadecimal number indicating the file size established when a file is allocated to a direct access device.

7.2.1 Function Code Field (SVC7.OPT)

SVC7 has nine functions specified by the first byte of the function code, called the command byte, and has four modifier fields specified by the second byte of the function code, called the modifier byte. The modifier fields are:

- Access privileges for the allocate function and change access privilege function
- Access method (data communications only) for the assign function
- File types for the allocate function
- Density selection for the assign function (magnetic tape drives)

There are no modifier fields for the rename, reprotect, close, delete, checkpoint and fetch attributes functions.

These functions and modifier fields are specified through different function code bit settings shown in Figure 7-2. The functions specified in the function code are executed from left-to-right.

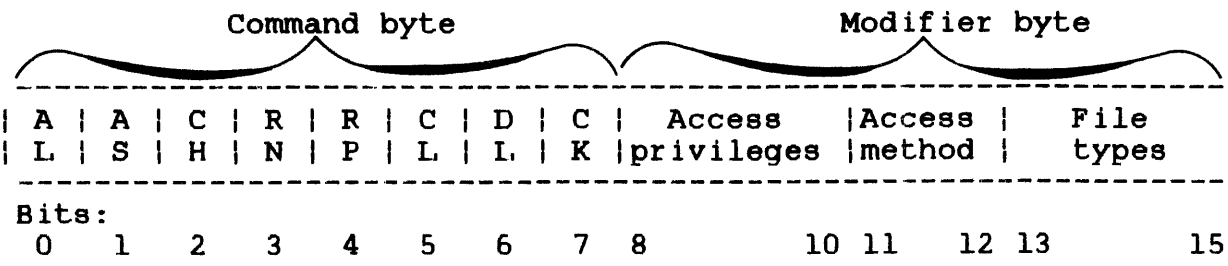


Figure 7-2 SVC7 Function Code Field

The function of each bit setting in the SVC7 function code field is explained in Table 7-1.

TABLE 7-1 SVC7 FUNCTION CODE BIT DEFINITIONS

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
0 (AL)	Allocate	Reserves space on a direct access device.	Reserves an LCB for a buffered terminal manager.	0 = no function requested 1 = reserve space
1 (AS)	Assign	Assigns an lu to a device or file.	Assigns an lu to line driver (SVC15) and terminal managers (SVC1).	0 = no function requested 1 = assign an lu
2 (CH)	Change access privilege	Changes the user's current access privilege to a new access privilege.	Changes the communications user current access privilege to a new access privilege.	0 = no function requested 1 = change access privilege

TABLE 7-1 SVC7 FUNCTION CODE BIT DEFINITIONS (Continued)

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
3 (RN)	Rename	Changes the current file-name to a new user-specified filename.	Changes the name of the communica-tions line (SVC15) or terminal (SVC1.)	0 = no function requested 1 = change filename
4 (RP)	Repro-tect	Changes the files current read/write protection keys to new protection keys.	Changes the read/write protection keys of the communi-cations line (SVC15) or terminal (SVC1) to new protection keys.	0 = no function requested 1 = change protection keys
5 (CL)	Close	Closes an lu assignment for a particular device or file.	Closes an lu assignment for a partic-ular line driver or terminal	0 = no function requested 1 = close an lu
6 (DL)	Delete	Releases re-served space on a direct access device.	Releases a reserved LCB.	0 = no function requested 1 = release reserved space
7 (CK)	Check-point	Copies buf-fered file data to a direct access device.	Copies buff-ered file data to a logical terminal.	0 = no function requested 1 = copy buffered file data

TABLE 7-1 SVC7 FUNCTION CODE BIT DEFINITIONS (Continued)

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
8 9 10	Access privileges	Specifies a file's reading and writing restrictions.	Specifies the terminal's reading and writing restrictions.	000 = shared read-only (SRO) 001 = exclusive read-only (ERO) 010 = shared write-only (SWO) 011 = exclusive write-only (EWO) 100 = shared read/write (SRW) 101 = shared read, exclusive write (SREW) 110 = exclusive read, shared write (ERSW) 111 = exclusive read/write (ERW)
11 12	Access method	Specifies vertical forms control (VFC) for devices that support VFC.	Indicates file access method for data communications.	00 = terminal level (SVC1) access 01 = terminal level (SVC1) access with VFC 10 = reserved 11 = line level (SVC15) access

TABLE 7-1 SVC7 FUNCTION CODE BIT DEFINITIONS (Continued)

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
13 14 15	File types or software density selection	Indicates file type or mag tape density being used.	Indicates if buffered terminal or line access is being used.	000 = contiguous files or enable manual density selection on Telex mag tape drives; no action on other mag tape drives 001 = extendable contigous files 010 = indexed files 011 = nonbuffered indexed files 110 = long record files 100 = select 800 bpi nonreturn to zero inverted (NRZI) density (STC and Telex drives only) 101 = Select 1600 bpi PE density (STC and Telex drives only)

TABLE 7-1 SVC7 FUNCTION CODE BIT DEFINITIONS (Continued)

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
13 14 15 (Continued)				110 = select 6250 bpi group coded recording (GCR) density (STC and Telex drives only) 111 = communications buffered terminal manager
0-15	Fetch attributes	Returns the physical attributes of a file or device to the parameter block.		X'0000' = fetch attributes
0-15	VFC	Turns VFC on or off for devices that support VFC.		X'FF20' = VFC on X'FF21' = VFC off
0-15	Fetch time and date attributes from disk Directory	Returns the time and date that the file was created and last written to.		X'FF00 = returns time and date in system generation (sysgen) format X'FF01' = returns time and date in mm/dd/yy; hr:min:sec format X'FF02' = returns time and date in dd/mm/yy; hr:min:sec format

TABLE 7-1 SVC7 FUNCTION CODE BIT DEFINITIONS (Continued)

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
0-15 (Continued)				X'FF03' = returns time and date in Julian format X'FF04' = returns time and date in directory format
0-15	Fetch logical attributes of open file	Returns current total logical records, current logical record position, index blocksize and data blocksize.		X'FF0A'

7.2.1.1 Allocate Function

The allocate function makes a directory entry and reserves space on a direct access device for the file type specified in the modifier byte. The required parameter block fields for this function are:

- Bits 0 and 13 through 15 of the function code
- Write key field
- Read key field
- Logical record length field
- Volume name field
- Filename field
- Extension field
- File class field
- File size field

When a contiguous file is allocated, the file sectors are reserved, and the filename, sector starting address, read/write keys, file type and dates created and written are entered into the directory. A contiguous file is not buffered. When an indexed file is allocated, the filename, number of logical records, read/write keys, file type and dates created and written are entered into the directory.

When an extendable contiguous file or nonbuffered indexed file is allocated, the file directory is set up as for an indexed file.

When doing an allocation using a data communications terminal manager, two data buffers, each equal to the device physical block size, are reserved in memory for the LCB. The buffered terminal filename, logical record length, and read/write keys are entered into the LCB. See the OS/32 Basic Data Communications Reference Manual.

7.2.1.2 Assign Function

The assign function uses an lu to establish a logical connection between the task issuing the SVC7 and a file or device, and the communications line and buffered terminal. The required fields in the parameter block are:

- Bits 1 and 8 through 12 (and 13 through 15 for magnetic tape drives) of the function code
- lu
- Write key field*
- Read key field*
- Volume name field
- Filename field*
- Extension field*
- File class field

* Used for direct access devices

When assigning to disk devices, the user-specified read/write keys corresponding to the specified access privileges are compared to the read/write keys in the file directory entry. If there is a match, the file is assigned according to the specified access privileges. If the access privileges are SWO or EWO and the user executes an assign function, the file is positioned at its logical end (append mode); otherwise, the file is positioned at the beginning. The access method '01' specifies the use of VFC.

When assigning to nondirect access devices, only the access privileges are examined. If the file is an indexed file, two data buffers and one indexed buffer are allocated in system space when the file is assigned. Each data buffer equals the file data block size; the index buffer equals the file index block size. If the file is an extendable contiguous file or nonbuffered indexed file, one index buffer is allocated in system space when the file is assigned.

7.2.1.2.1 Temporary File Allocation and Assignment Function

The allocation and assignment functions can also reserve space temporarily on a direct access device for the file type specified in the modifier byte. Such a file is temporary because it exists only while the file is assigned to an lu and is deleted when the file is closed. The required parameter block fields for this function are:

- Bits 0, 1, 8 through 10, and 13 through 15 of the function code field
- lu field
- Logical record length field
- File size field

To allocate a temporary file, specify an allocate or assign function and an ampersand (&) as the first character of the filename. When the temporary file is allocated, a directory entry is made for the filename and the file is placed by default on the temporary volume. The temporary file is then assigned to the lu specified in the parameter block. A temporary file also can be allocated and assigned from the system console through the operator TEMPFILE command. See the OS/32 Operator Reference Manual.

7.2.1.3 Change Access Privileges Function

The change access privileges function changes the current access privileges of an assigned file or device to the access privileges specified in the parameter block. The new access privileges must be compatible with the existing ones; otherwise, the change in access privileges would result in increased access. For example, to change SRO to SRW, the appropriate protection key must be supplied. The required parameter block fields for this function are:

- Bits 2 and 8 through 10 of the function code field
- lu field

- Write key field
- Read key field

7.2.1.4 Rename Function

The rename function causes the filename and extension identifiers currently in effect to be changed to the filename and extension identifiers specified in the parameter block. The file must be currently assigned to the specified lu with ERW access privileges and exist on a direct access storage device (DASD). The required parameter block fields for this function are:

- Bit 3 of the function code field
- lu field
- Filename field
- Extension field
- File class/account field

When executing the rename function, the parameter block volume name is ignored, and the specified filename and extension replace the current filename and extension in the device directory.

NOTE

An e-task is allowed to rename a device. See the OS/32 System Level Programmer Reference Manual for more information.

7.2.1.5 Reprotect Function

The reprotect function changes the read/write protection keys of a currently assigned file to the contents of the read and write key fields. The file must be on a direct access device and assigned to the specified lu with access privileges. The required parameter block fields are:

- Bit 4 of the function code field
- lu field
- Write key field
- Read key field

When executing the reprotect function, the specified read/write keys replace the current read/write keys of a specified file in the device directory.

NOTE

An e-task is allowed to reprotect a device. See the OS/32 System Level Programmer Reference Manual for more information.

7.2.1.6 Close Function

The close function breaks the logical connection between the task and file or between the device or a data communication line and terminal by closing the currently assigned lu. The parameter block's required fields are:

- Bit 5 of the function code field
- lu field

When the lu is closed, all data in system buffers or terminal buffers are copied to the user file.

7.2.1.7 Delete Function

The delete function removes the file directory entry and releases the reserved space of a currently unassigned file on a direct access device. When deleting through the communications buffered terminal manager, a currently unassigned LCB is removed from memory. The required parameter block fields are:

- Bit 6 of the function code field
- Write key field
- Read key field
- Volume name field
- Filename field
- Extension field
- File class/account field

If the contents of the parameter block volume name, filename, extension and read/write keys fields match the fields in the file directory entry, the file is deleted. If the logical terminal name matches the name in the LCB, the LCB is deleted.

7.2.1.8 Checkpoint Function

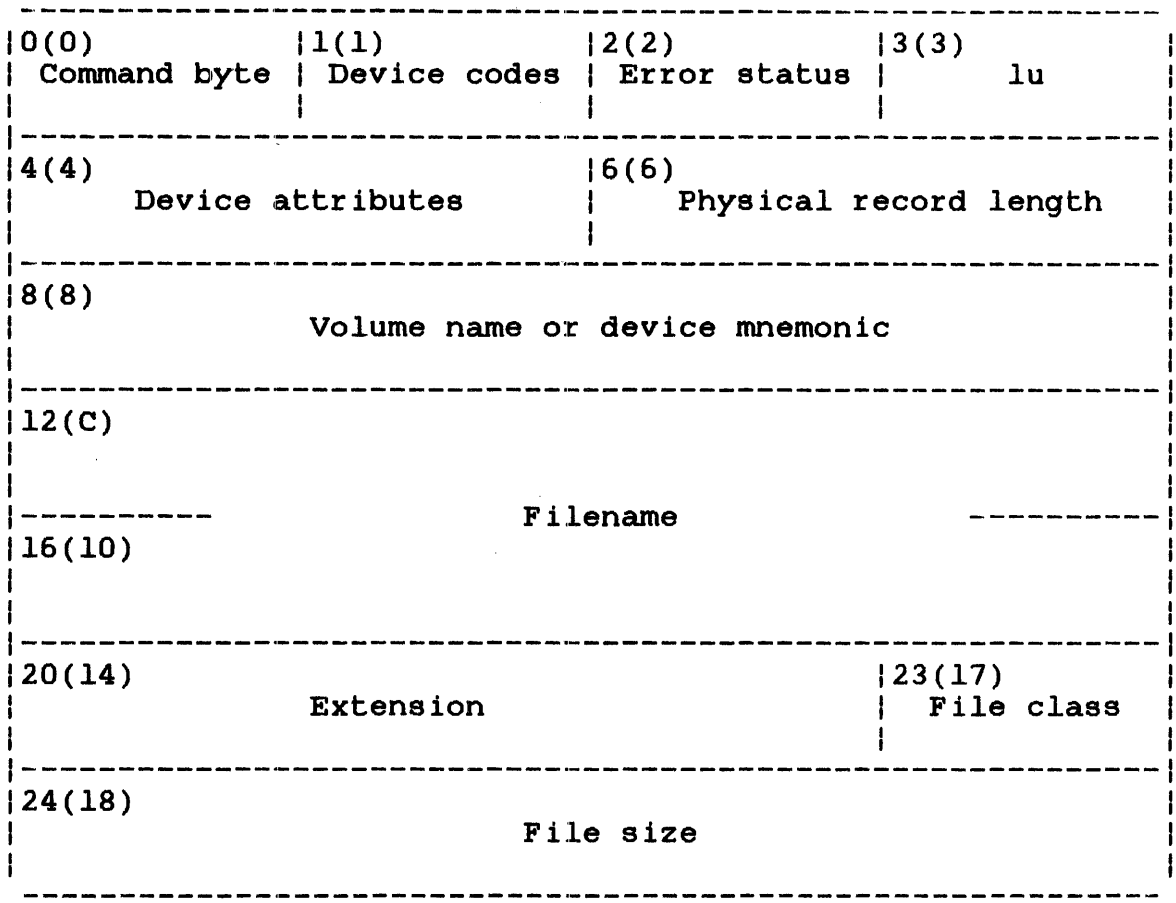
The checkpoint function copies the buffered file data to the indexed file or the buffered terminal data to the terminal and updates the directory entries. Executing a checkpoint function on a nonbuffered indexed file or extendable contiguous file updates the directory entries for the file. Executing a checkpoint function on a contiguous nondirect access device or unbuffered file has the same effect as an SVC1 wait-only call. The required parameter block fields for this function are:

- Bit 7 of the function code field (bits 11 and 12 for data communications)
- lu field

After executing a checkpoint function, the file pointer is not repositioned to the beginning of the file as in a close function. If a system failure occurs and data exists in the file buffers, all data up to the last close or checkpoint function is recoverable; any data appended after the last close or checkpoint function is lost. Therefore, to prevent loss of data, use the checkpoint function frequently, especially after a large amount of data or any important data has been written to a buffered file.

7.2.1.9 Fetch Attributes Function

The fetch attributes function sends to the SVC7 parameter block the physical attributes of the file or device currently assigned to the specified lu. These attributes include the device mnemonic or volume name, filename, extension, file class and file size, which are sent to their respective fields in the SVC7 parameter block. Device codes are sent to the modifier byte of the function code field. Device attributes are stored in the write and read key fields. The logical record length field can receive either a file logical record length or a device physical record length. These field differences for the fetch attributes function are illustrated in Figure 7-3.




```
          SVC   7,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,0
          DS    1
          DB    lu
          DS    24 bytes for device attributes
```

Figure 7-3 SVC7 Parameter Block Format and Coding for a Fetch Attributes Function

When executing this function, the device codes field receives a hexadecimal number indicating the file or device type. The OS/32 System Generation 132 (Sysgen/32) Reference Manual lists all the devices and their device codes. The command byte, error status and lu fields are the same as those defined in Section 7.2.

The device attributes field receives a hexadecimal number indicating certain file or device attributes. Table 7-2 lists all supported attributes and corresponding masks.

TABLE 7-2 DESCRIPTION AND MASK VALUES OF THE DEVICE ATTRIBUTES FIELD

BIT POSITION	MASK	ATTRIBUTES
0	X'8000'	Interactive device
1	X'4000'	Supports read*
2	X'2000'	Supports write*
3	X'1000'	Supports binary
4	X'0800'	Supports wait I/O
5	X'0400'	Supports random access
6	X'0200'	Supports unconditional proceed
7	X'0100'	Supports image mode and extended options
8	X'0080'	Supports halt I/O
9	X'0040'	Supports rewind
10	X'0020'	Supports backspace record
11	X'0010'	Supports forwardspace record
12	X'0008'	Supports write filemark
13	X'0004'	Supports forward space filemark
14	X'0002'	Supports backspace filemark
15	X'0001'	Device-dependent function

* Indicates the current access privilege

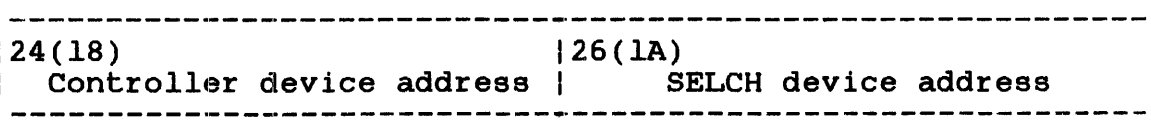
The physical record length field receives the logical record length of the file or physical record length of the device assigned to the specified lu; e.g., 80-byte record for card readers and 120- or 132-byte record for line printers. If the device has variable length records, a value of 0 is returned to this field; e.g., magnetic tape. However, variable length record devices are normally used as fixed record length devices.

For direct access devices, contiguous and extendable contiguous files may be treated as having either a sector-length record size (256 bytes) or a variable length record. Indexed and nonbuffered indexed files have a fixed record length, which is the file's logical record length established at allocation time.

The direct access device volume name, filename, extension and file class are sent to their corresponding fields in the parameter block. For a nondirect access device, the device mnemonic is sent to the volume name field, and the filename, extension and file class fields of the parameter block are filled with blanks.

For direct access devices, the file size field receives an unsigned hexadecimal number indicating the current size of a direct access file. For indexed and nonbuffered indexed files, this field contains the number of logical records in the file. For contiguous or extendable contiguous files, this field contains the number of sectors in the file.

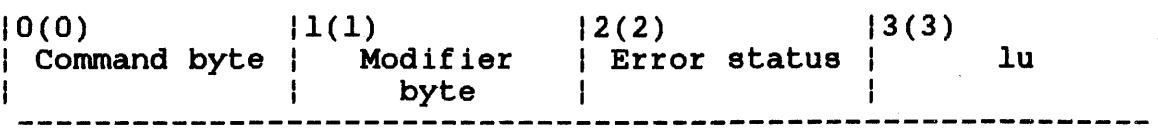
For bare disk devices, the first two bytes of the file size field contain the controller device address if the bare disk is attached to a controller. If the bare disk is not attached to a controller, the first two bytes equal zero. The remaining two bytes of this field contain the selector channel (SELCH) device address if the disk is accessed via a SELCH device; otherwise, this 2-byte field contains a zero.



After executing a fetch attributes call, the file size field receives the current size of a file on a direct access device. The file size field is not used for nondirect access devices.

7.2.1.10 Vertical Forms Control (VFC) Function

The VFC option turns the VFC function on or off for a particular device. To execute this function, only the first four bytes of the SVC7 parameter block are required as shown in Figure 7-4.




```
          SVC   7,parblk
          .
          .
          .
parblk    ALIGN 4
          DB    X'FF'
          DB    X'20' or '21'
          DS    1
          DB    lu
```

Figure 7-4 SVC7 Parameter Block Format and Coding for VFC Function

This parameter block must be fullword boundary-aligned and located in a task-writable segment.

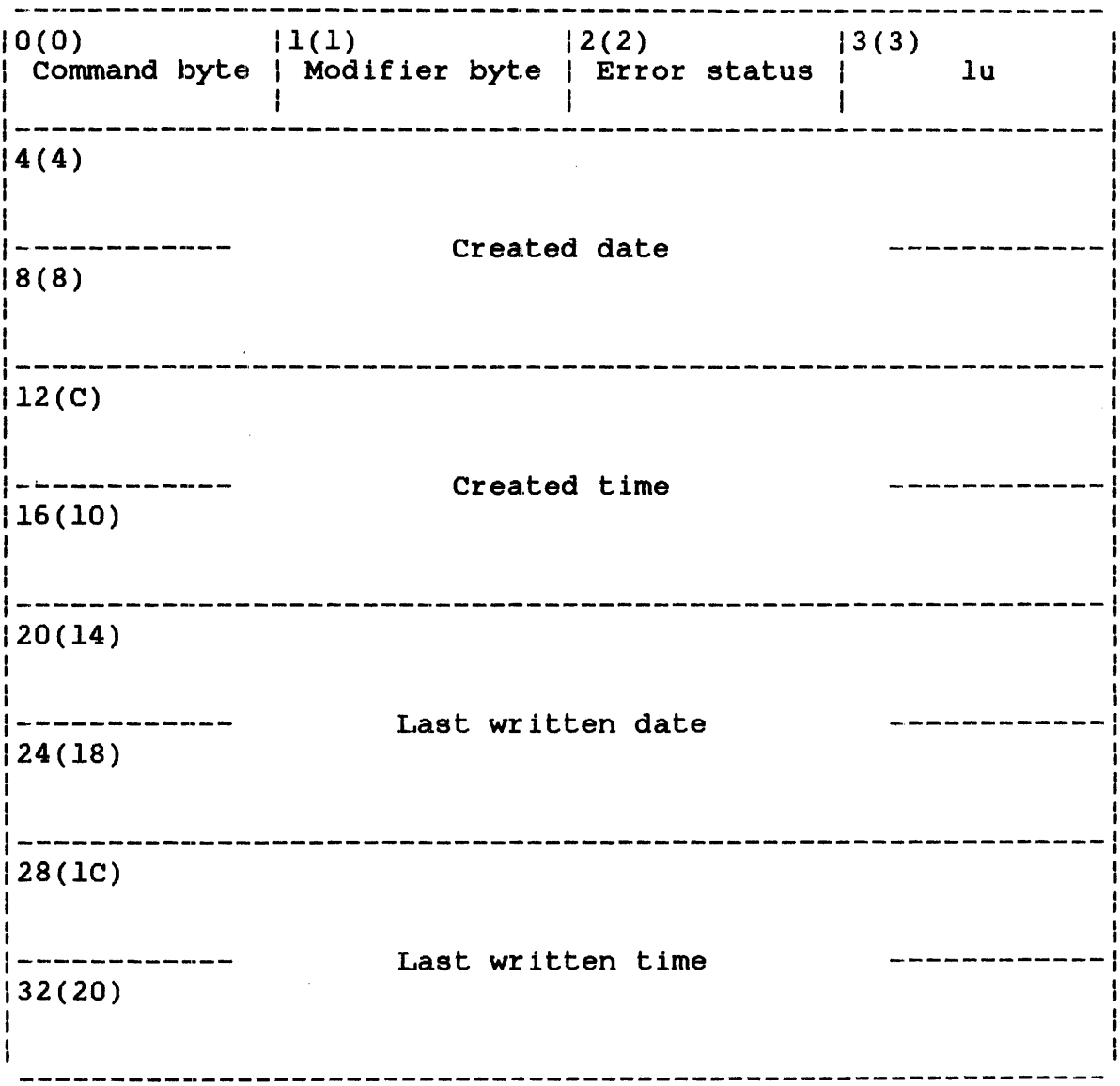
To turn on the use of the VFC function for a particular device, set the modifier byte to X'20'. To turn the function off, set the modifier byte to X'21'. The error status and lu fields are the same as for all SVC7 services.

7.2.1.11 Fetch Time and Date Attributes from Disk Directory Function

The fetch time and date attributes function returns to the SVC7 parameter block the date and time the disk file was created and last written to.

FUNCTION CODE	FORMAT
X'FF00'	Defined during sysgen
X'FF01'	Month/day/year hours:minutes:seconds
X'FF02	Day/month/year hours:minutes:seconds
X'FF03'	Julian
X'FF04'	Directory

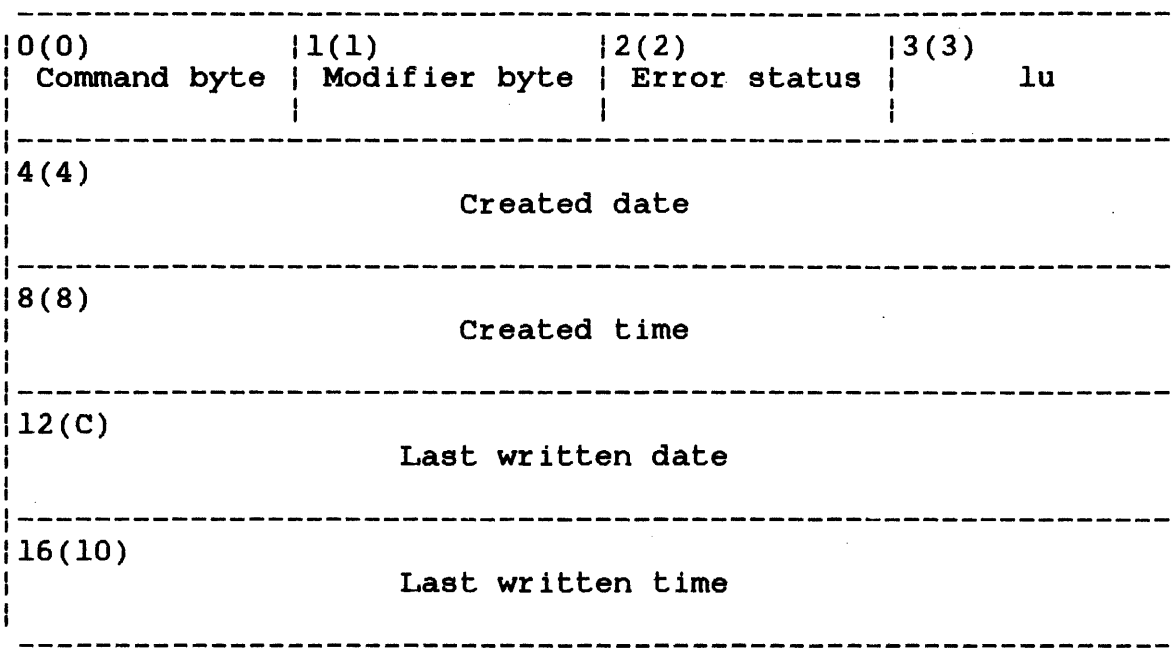
The parameter block fields for receiving the first three options are shown in Figure 7-5. Sysgen can define either format designated by function code X'FF01' or X'FF02'.




```
          SVC   7,parblk
          .
          .
          .
parblk    DB    X'FF'
          DB    X'0n'       n=0,1, or 2
          DS    1
          DB    lu
          DS    8 bytes for created date
          DS    8 bytes for created time
          DS    8 bytes for last written date
          DS    8 bytes for last written time
```

Figure 7-5 SVC7 X'FF00', X'FF01' or X'FF02' Parameter Block Format and Coding for Fetch Time and Date Attributes Function

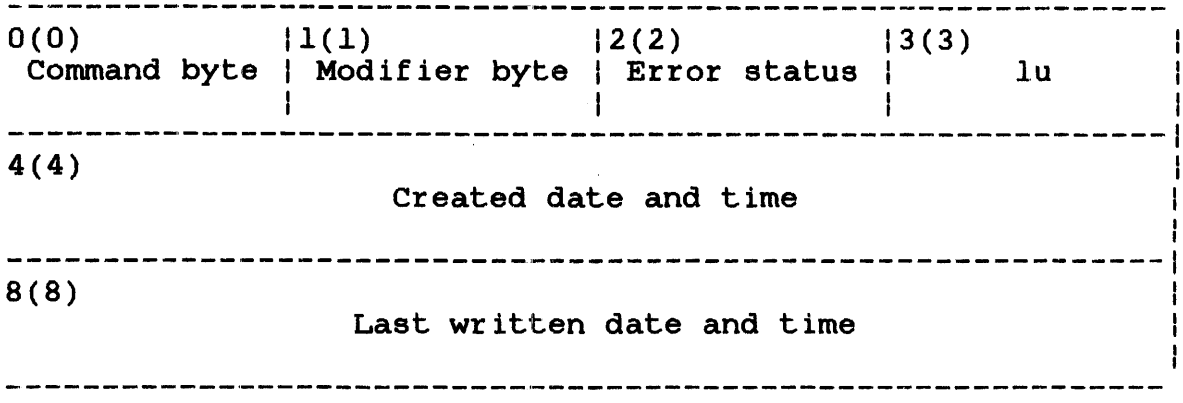
The SVC7 parameter block fields for receiving the Julian format are shown in Figure 7-6. The date is represented by a five-digit number. The first two digits indicate the year; the last three digits indicate the number of days since January 1. The time is the number of minutes since midnight. Both the date and time are returned as binary numbers.




```
          SVC   7,parblk
          .
          .
          .
parblk    DB    X'FF'
          DB    X'03'
          DS    1
          DB    lu
          DS    4 bytes for created date
          DS    4 bytes for created time
          DS    4 bytes for last written date
          DS    4 bytes for last written time
```

Figure 7-6 SVC7 X'FF03' Parameter Block Format and Coding for Fetch Time and Date Attributes Function

The SVC7 parameter block fields for receiving the dates and times exactly as they appear in the directory are shown in Figure 7-7.




```
          SVC   7,parblk
          .
          .
          .
parblk    DB    X'FF'
          DB    X'04'
          DS    1
          DB    lu
          DS    4 bytes for created date and time
          DS    4 bytes for last written date and time
```

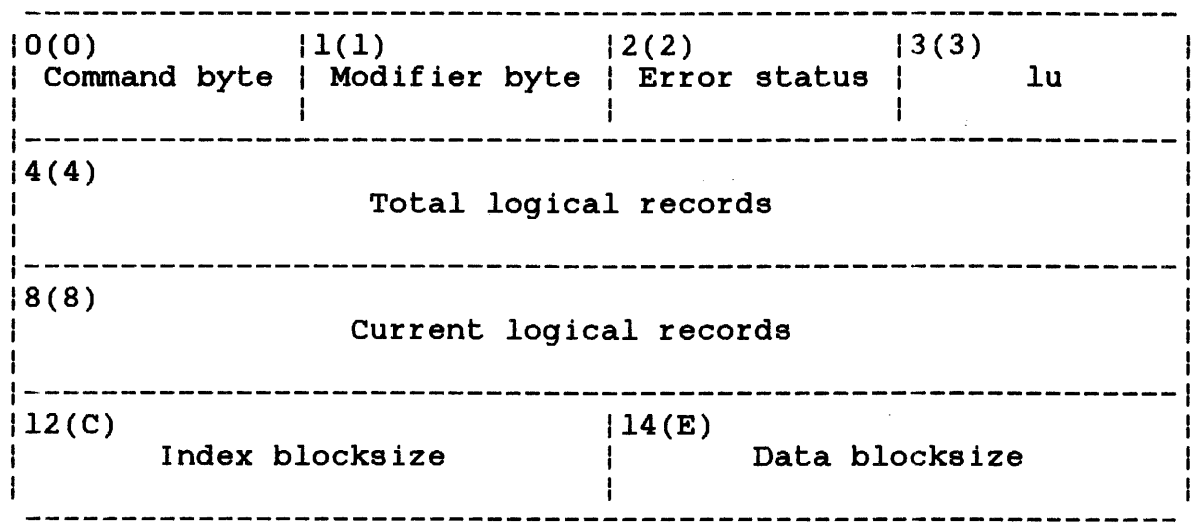
Figure 7-7 SVC7 X'FF04' Parameter Block Format and Coding for Fetch Time and Date Attributes Function

7.2.1.12 Fetch Logical Attributes of Open File Function

The fetch logical attributes of open file function returns the following attributes for an open file to the SVC7 parameter block:

- Total logical records currently in the file
- Current logical record position if the file is accessed sequentially
- Index blocksize of indexed nonbuffered indexed and extendable contiguous open files
- Data blocksize of indexed nonbuffered indexed and extendable contiguous open files

The function code for this SVC7 is X'FF0A. Figure 7-8 shows the SVC7 parameter block fields for receiving the logical attributes of an open file.




```
          SVC   7,parblk
          .
          .
          .
parblk    DB    X'FF'
          DB    X'0A'
          DS    1
          DB    lu
          DS    4 bytes for total records
          DS    4 bytes for current logical record
          DS    2 bytes for index blocksize
          DS    2 bytes for data blocksize
```

Figure 7-8 SVC7 X'FF0A' Parameter Block Format and Coding for the Fetch Logical Attributes of Open File Function

7.2.1.13 Spoolfile Assign to Pseudo Device Function

The spoolfile assign to pseudo device function will enable the user to specify such things as preprinted forms, number of copies to be output, output priority, image or formatted I/O, multiple assignments to the same spoolfile and specification of data-index block sizes for the spoolfile. Additionally, the printing of a spoolfile can be delayed until its output is requested.

This spoolfile assign is intercepted by SPL/32, which is solely responsible for its handling. If the SPL/32 fails to intercept this SVC7, an error will result (error code X 'D': SPL/32 inactive).

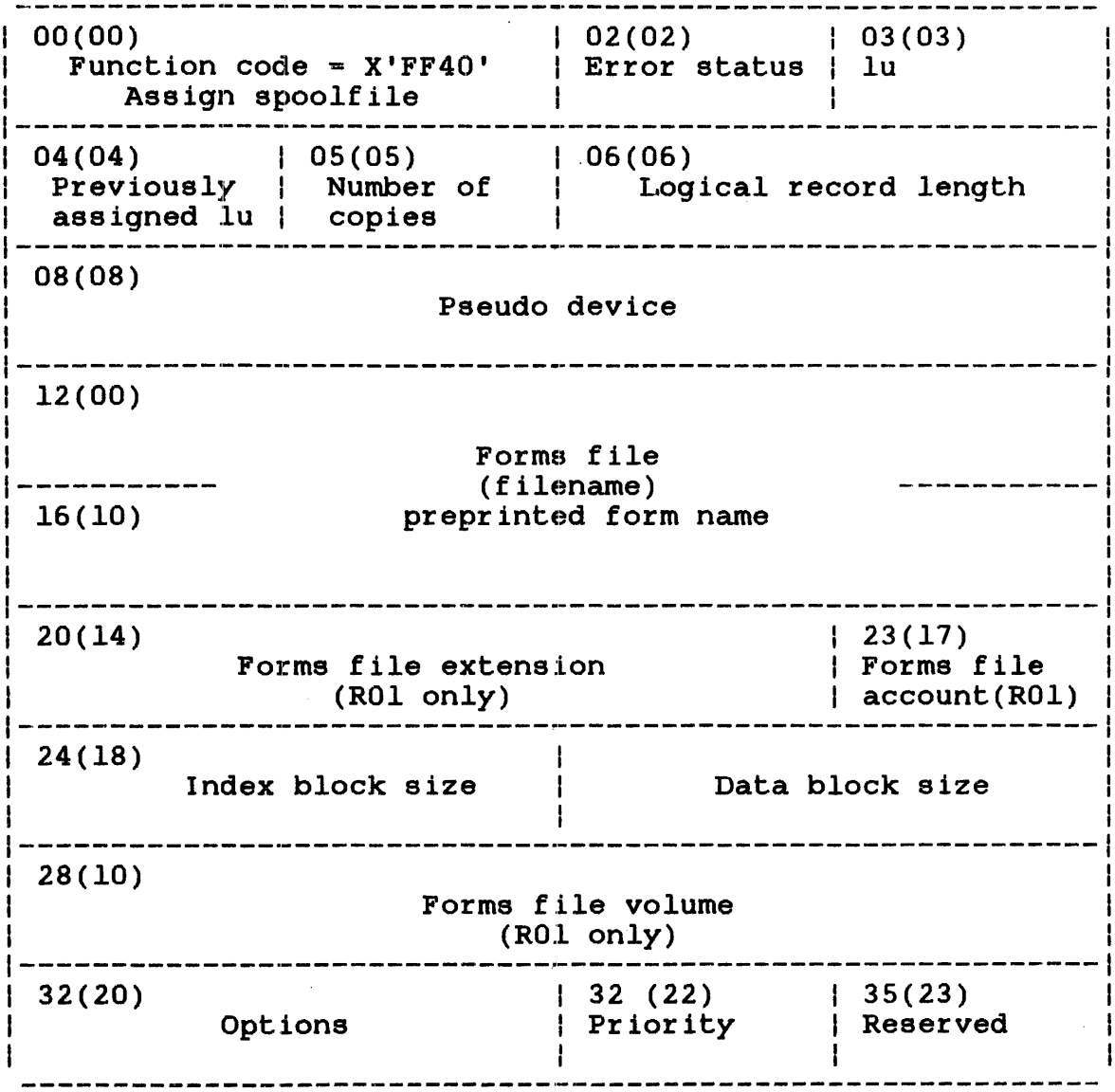


Figure 7-9 SVC7 Spoolfile Assign to Pseudo Device Parameter Block

The 1-byte error status field receives the appropriate error code from SPL/32 when an error occurs while servicing an SVC7. If no error occurs, the error status field contains zeros. For error codes returned by SPL/32, see Table 7-6.

7.2.1.14 Extended Assign to Spoolfile Function

If multi-terminal monitor (MTM) is doing an assign on behalf of a task, it assigns the lu to itself, then sends the lu to the task. If the assign is intercepted, the information retrieved via the PEEK option 3 is about MTM and not the expected task. For this reason, an extended spoolfile assign has been created.

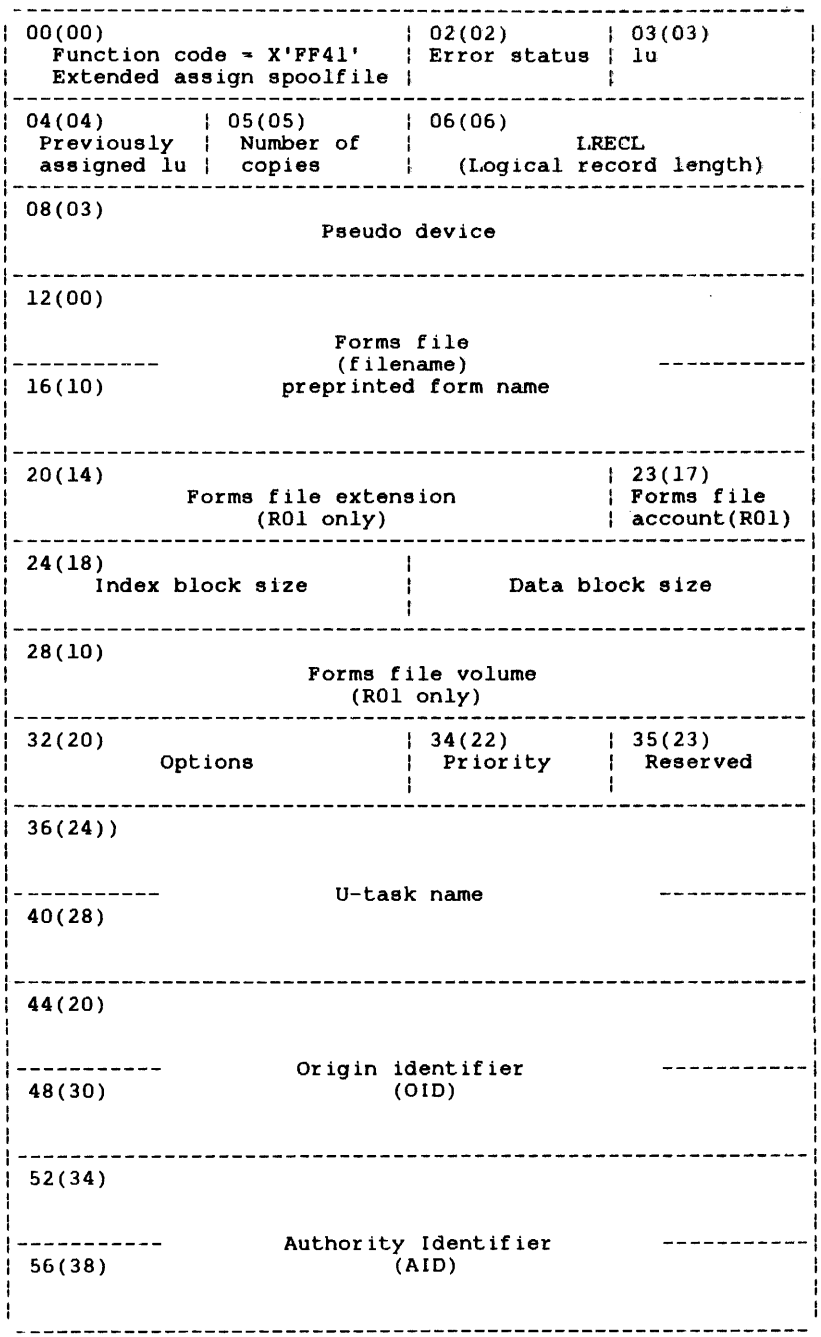


Figure 7-10 Extended Spoolfile Assign Parameter Block

Fields:

U-task name	is the name of the u-task for which the assign is being performed.
OID	is the origin identifier. The OID is used to determine the default print location for those devices or tasks associated with a group.
AID	is the authority identifier. The AID is used by SPL/32 to identify if an origin location has authority to perform the requested function.
Priority	is the current priority of the task requesting the assign.

7.2.1.15 Assign to Pseudo Device Function

In order to utilize spooling from the task level, the user can issue an SVC7 assignment (function code bit 1) for a pseudo device. The spooler will intercept the SVC7 assign and will process it.

The task's monitor name, priority and task name will be available through the PEEK option 3. If the task is running under the control of MTM, its AID and OID will be available as the originating user console device; otherwise this field will contains zeros. Any u-task that uses this standard SVC7 assign to a spooler pseudo device will continue to function without modification.

7.2.1.16 Access Privileges

This 3-bit modifier field contains the access privileges indicating the file's current reading and writing restrictions and is required for these functions:

- Assign
- Change access privilege
- Rename
- Reprotect

Access privileges allow other tasks to access an assigned file or prevent such access. Table 7-3 lists access privileges and their meanings that are established when the file is assigned and subsequently changed through the change access privilege function. The rename and reprotect functions require the file to have an assigned ERW access privilege before executing.

TABLE 7-3 ACCESS PRIVILEGE DEFINITIONS

ACCESS PRIVILEGE	MEANING	BIT SETTING
SRO	This task can read from the assigned file but cannot write to it. Other tasks can read from and write to the assigned file.	000=SRO
ERO	This task can read from the assigned file but cannot write to it. Other tasks can write to but cannot read from the assigned file.	001=ERO
SWO	This task can write to the assigned file but cannot read from it. Other tasks can read from and write to the assigned file.	010=SWO
EWO	This task can write to the assigned file but cannot read from it. Other tasks can read from but cannot write to the assigned file.	011=EWO
SRW	Tasks can read from and write to the assigned file. This is the default.	100=SRW
SREW	This task can read from and write to the assigned file. Other tasks can read from but cannot write to the assigned file.	101=SREW
ERSW	This task can read from and write to the assigned file. Other tasks can write to but cannot read from the assigned file.	110=ERSW
ERW	This task can read from and write to the assigned file. Other tasks cannot read from or write to the assigned file.	111=ERW

7.2.1.17 Change Terminal Mode

This SVC7 is used to change the access method of a Model 3270 drop without eliminating and regenerating a Model 3270 support LCB. This extended SVC7 allows a task to switch between Model 3270 Pass Thru and Model 1200 Emulation, and/or change the read method between read immediate modified and read immediate unprotected as the application requires. The format is similar to other extended SVC7 parameter blocks with the exception of the following settings in the function code.

FUNCTION CODE	EXTENDED SVC7 FUNCTIONS
X'FF30'	Model 1200 terminal emulation read immediate unprotected
X'FF31'	Model 3270 rerminal pass thru read immediate unprotected
X'FF32'	Model 1200 terminal emulation read immediate modified
X'FF33'	Model 3270 terminal pass thru read immediate modified

7.2.1.18 Data Communications Access Methods

This 2-bit modifier field contains the access methods used by data communications. The access methods are listed in Table 7-4. See the OS/32 Basic Data Communications Reference Manual.

TABLE 7-4 DATA COMMUNICATIONS ACCESS METHOD DEFINITIONS

DATA COMMUNICATIONS ACCESS METHOD	MEANING	BIT SETTING
Terminal level access	This device-independent support of a communications terminal through the data communications terminal manager is in both buffered and unbuffered mode. This is accomplished through SVC1.	00 = terminal level access 01 = terminal level access with VFC
Line level access	This is the device-dependent support of a communications line through the data communications driver. This is accomplished through SVC15.	11 = line level access

7.2.1.19 File Types

This 3-bit modifier field contains file types used and required by the allocate function. The file types are:

- Contiguous files
- Extendable contiguous files
- Indexed files
- Nonbuffered indexed files
- Long record files
- Data communications buffered terminal manager

The file type field is also used to select the density of write operations to a magnetic tape drive. This selection is made when the magnetic tape driver is assigned to an lu through the SVC7 assign function. The software density selections available to the assign function are described below.

BIT SETTING	DENSITY SELECTION
000	Manual density (Telex drives only)
100	800 bits per inch (bpi) NRZI density (STC and Telex drives only)
101	1600 bpi PE density (STC and Telex drives only)
110	6250 bpi GCR density (STC and Telex drives only)

For STC and Telex drives, neither software density selection nor manual density selection has any effect on read operations. The tape is always read at the density at which it was recorded.

For drives that require software enabling of manual density selection (i.e., Telex drives), a value of zero should be placed in the file type field if manual density selection is desired. For drives that require manual enabling of software density selection (i.e., STC), software select should be enabled on the operator panel before the first output operation is attempted. Otherwise, the tape is written at the manually selected density. In addition, if the magnetic tape drive does not support software selection of density and the file type field does not contain zero, the drive is not assigned and status code X'09' is returned in the SVC7 parameter block.

7.2.1.20 Read/Write Key Fields (SVC7.RKY/SVC7.WKY)

The read/write key fields should contain the hexadecimal number indicating the file or device read/write protection keys established at allocation time. When a task is assigned to a file or device through an lu, the read/write protection keys specified at assign time are compared to the keys established at allocation time for a match. If they match, the condition is met, and the task can be assigned for the protected access mode (conditionally protected). Files and devices can be unprotected, allowing any key specified at assign time to be accepted. Files and devices can also be unconditionally protected, causing rejection of any keys specified at assign time. Table 7-5 lists the read/write protection keys.

TABLE 7-5 READ/WRITE PROTECTION KEYS DEFINITIONS

KEYS	MEANING
00	Unconditionally unprotected; the file or device is unprotected for the specified access mode (read or write). Any key specified at assign time is accepted. If no keys are specified, this key is the default.
01 through FE	Conditionally protected; the file or device is protected for the specified access mode (read or write). Matching keys must be specified at assign time to gain access to the device or file.
FF	Unconditionally protected; the file or device is protected for the specified access mode (read or write). No u-task can assign for protected access mode.

7.2.1.21 File Size Field (SVC7.SIZ)

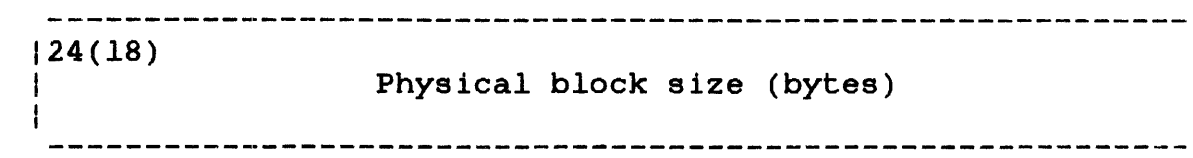
The file size field must contain a hexadecimal number indicating the file size established at allocation time on a direct access device. For contiguous files, this field must contain the number of sectors in the file.

24(18)
Number of sectors

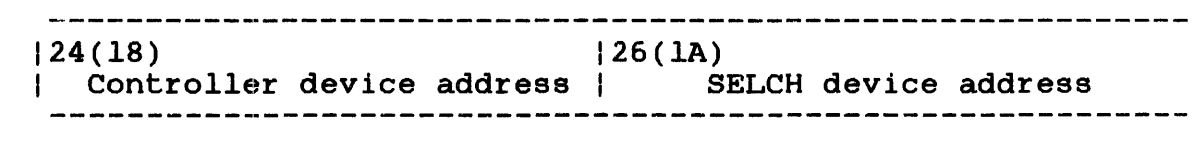
For indexed, nonbuffered indexed and extendable contiguous files, the first two bytes of the file size field must contain the index block size in increments of sectors (256 bytes); the remaining two bytes of the file size field must contain the data block size in increments of sectors. If zeros are specified for index block size or data block size, the file is allocated to the system default appropriate for that particular parameter.

24(18) Index block size (sectors) (SVC7.ISZ)	26(1A) Data block size (sectors) (SVC7.DSZ)

For data communications buffered terminals, this field must contain the physical block size in bytes.



For bare disk devices, the first two bytes of the file size field contain the controller device address if the bare disk is attached to a controller. If the bare disk is not attached to a controller, the first two bytes equal zero. The remaining two bytes of this field contain the SELCH device address if the disk runs from a SELCH device; otherwise, this 2-byte field contains a zero.



After executing a fetch attributes call, this field receives the current size of a file on a direct access device. This field is not used for nondirect access devices.

7.3 SVC7: EXTENDED FUNCTIONS FOR PRIVILEGED TASKS

The OS/32 file manager provides additional resource management services to privileged tasks. These services are accessed through the extended function codes of SVC7. These functions include:

- bare disk assignment,
- fetch attributes for bare disk devices,
- device rename and reprotect, and
- extended close.

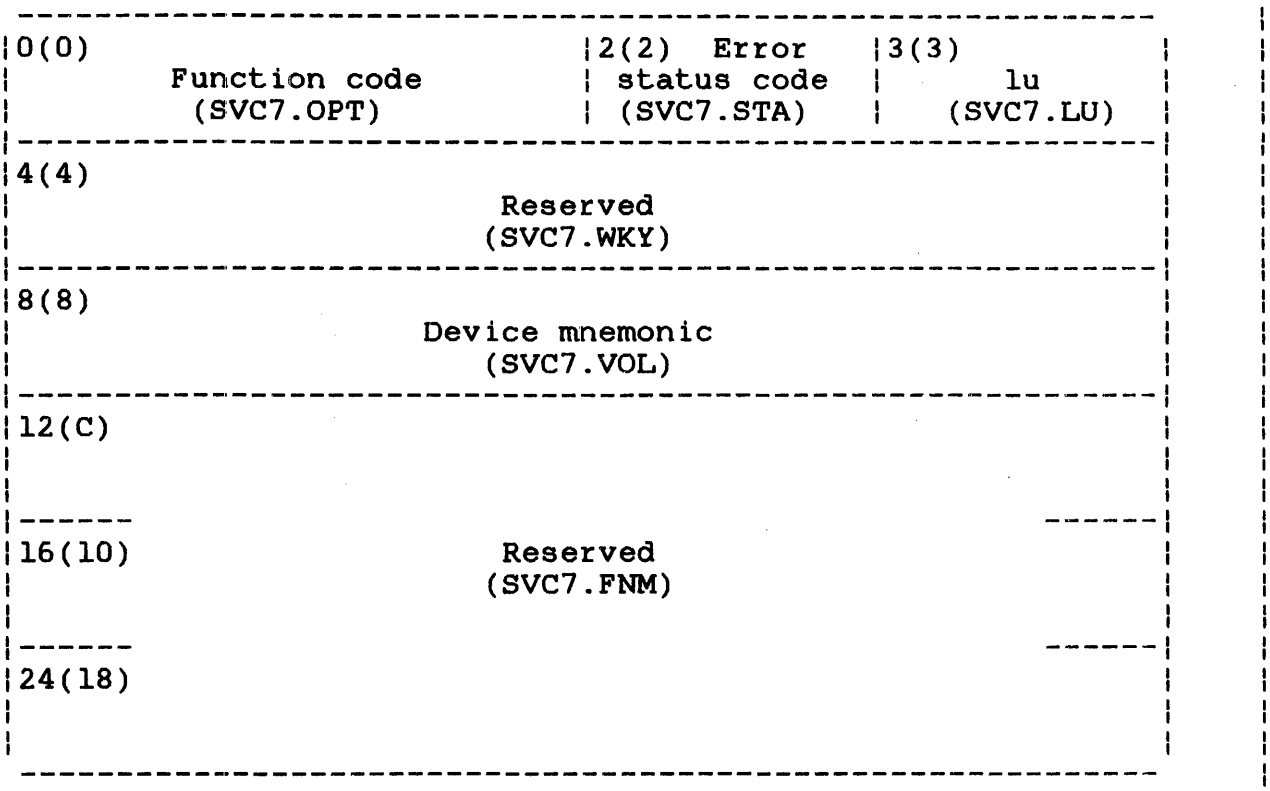
The following sections describe how to access the SVC7 extended functions for privileged tasks.

7.3.1 SVC7: Bare Disk Assignment

An e-task, privileged u-task or diagnostic task (d-task) with bare disk privileges can bypass the file manager and directly assign to a disk device. When such a task issues an SVC1 I/O request directly to a disk device, the operation is referred to as bare disk I/O and should always be random access. The supported I/O functions are read, write and test and set.

Direct assignment to a disk device can be performed only by a task that has the bare disk task option enabled. E-tasks always have this option enabled. A u-task or d-task is given bare disk privileges by specifying the disk privilege option in the OPTION command (OPTION DISC) when the task is built by Link. However, if the task loader has the e-task option prevented, the privileged task is loaded into memory with the bare disk privilege option changed to the default, no bare disk privilege. Therefore, bare disk I/O cannot be performed by the task.

The SVC7 parameter block and coding for bare disk assignments are shown in Figure 7-11.




```
          SVC   7,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    X'function code'
          DB    0
          DB    lu
          DC    0
          DC    C'4-character device mnemonic'
          DC    0,0,0,0
```

Figure 7-11 SVC7 Bare Disk Assignment Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary-aligned and located in a task-writable segment. A description of each field in the parameter block follows.

Fields:

Function code (SVC7.OPT)	is a 2-byte field that contains the hexadecimal number indicating the assign function (bit 1 must be set). In addition, the appropriate access privileges (bits 8 through 10) must be set as follows:

- 000 = SRO
- 001 = ERO
- 010 = SWO
- 011 = EWO
- 100 = SRW
- 101 = SREW
- 110 = ERSW
- 111 = ERW

CAUTION

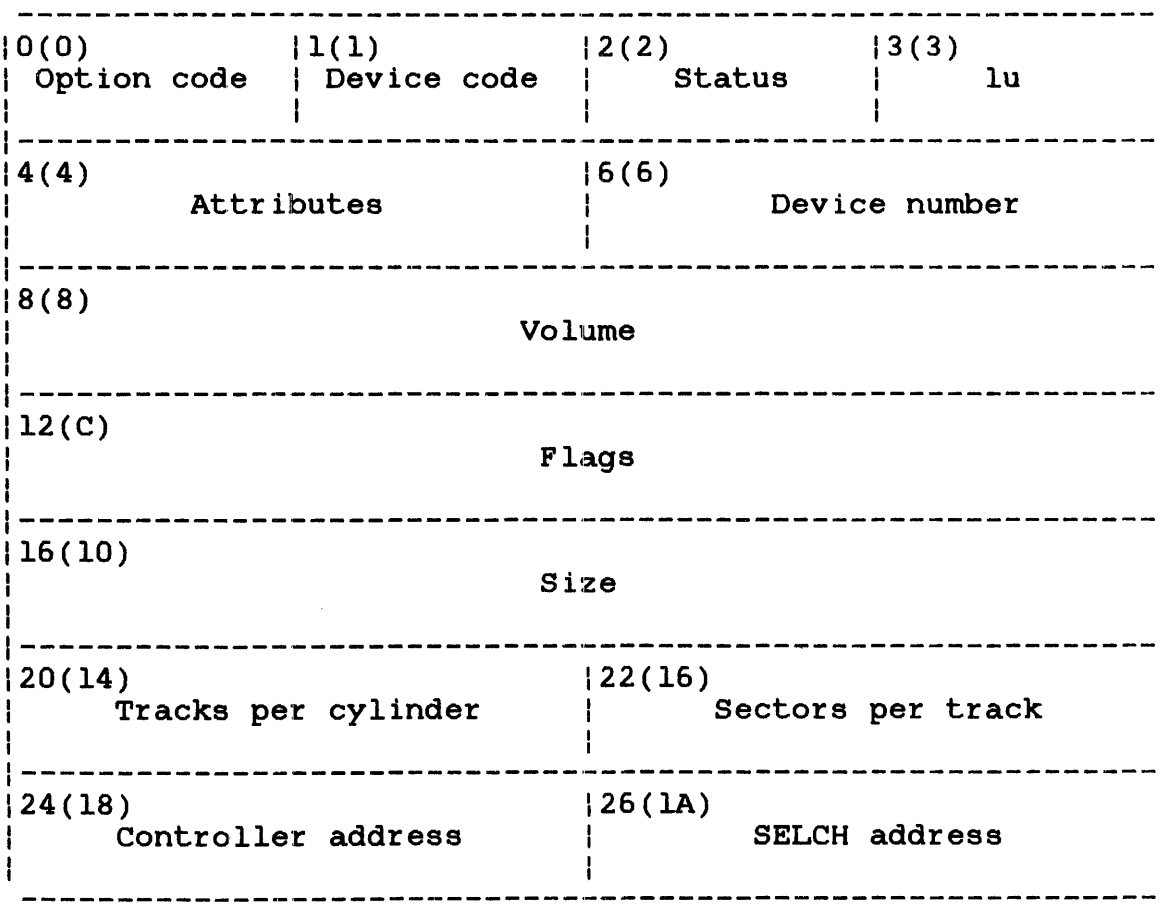
IF THE BARE DISK IS MARKED ON-LINE, ONLY ASSIGNMENTS FOR SRO ARE ALLOWED. ANY OTHER ACCESS PRIVILEGE IS REJECTED, AND A PRIVILEGE ERROR STATUS (07) IS GIVEN.

Error status codes (SVC7.STA)	is a 1-byte field that receives an error code when an error occurs during SVC7 execution If no error occurs, a value of 0 is returned to this field. See Table 7-6 for a list of SVC7 error codes.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit to be assigned to the bare disk device.
Reserved (SVC7.WKY)	is a 4-byte reserved field that must contain a zero.
Device mnemonic (SVC7.VOL)	is a 4-byte field containing the device mnemonic of the bare disk device.
Reserved (SVC7.FNM)	is a 16-byte field that must be reserved with zeros.

7.3.2 SVC7 Code 0: Fetch Attributes for Bare Disk Devices

The fetch attributes function fetches the attributes of a bare disk device through its assigned lu. The write attribute is reset in the attributes halfword field (SVC7.ATRB) of the parameter block if the disk is marked on protected.

This SVC7 function is available only to tasks with bare disk privileges or to e-tasks. Before issuing this SVC, the task must have the bare disk already assigned to the lu. Figure 7-12 shows the parameter block format and coding for SVC7 code 0.




```
          SVC   7,parablk
          .
          .
          .
          ALIGN 4
parblk    DB    0
          DS    2
          DB    lu
          DS    24
```

Figure 7-12 SVC7 Code 0 Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary-aligned and located in a task-writable segment. A general description of each field in the parameter block follows.

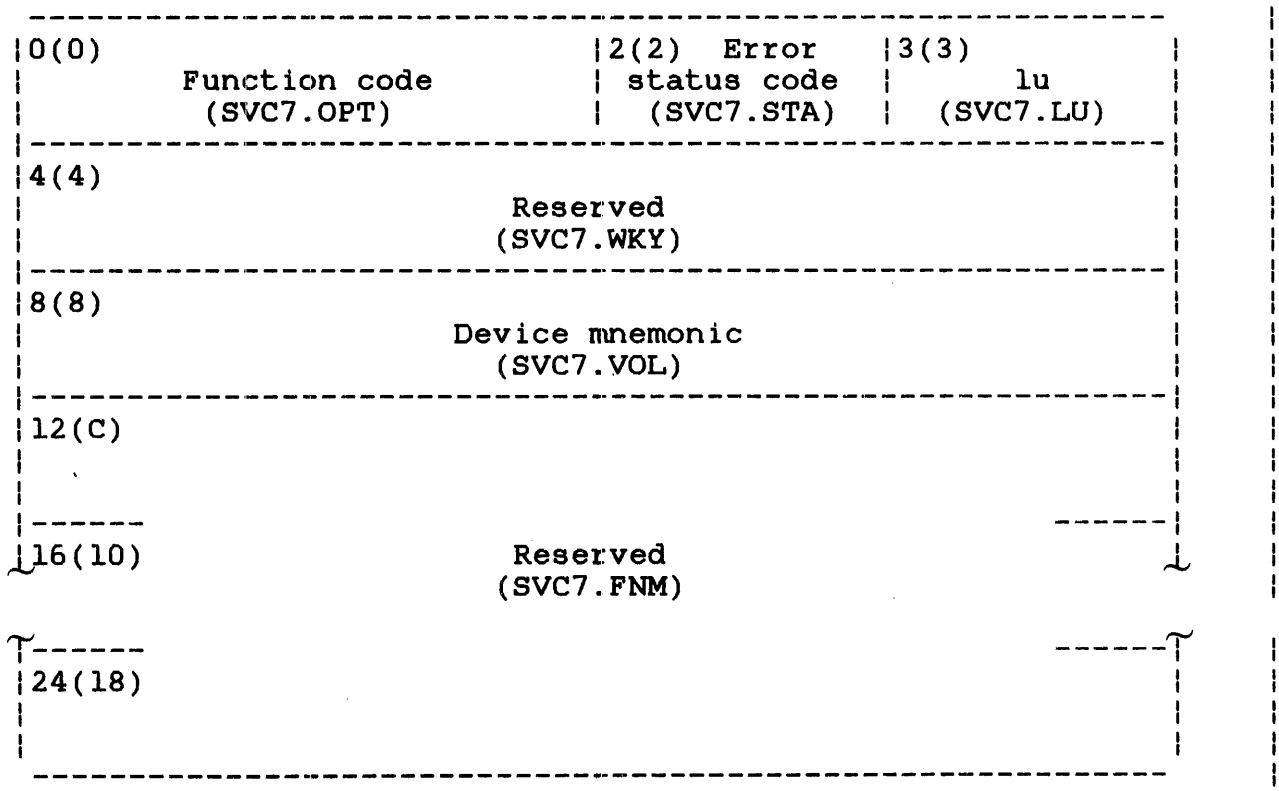
Fields:

Option	is a 1-byte field that must contain X'00' to indicate the fetch attributes function.
Device code	is a 1-byte field that receives the device code of the bare disk device.
Status	is a 1-byte field that receives the return status of the bare disk device.
lu	is a 1-byte field that must contain the logical unit for which attributes are returned.
Attributes	is a 2-byte field that receives the attributes of the bare disk device.
Device number	is a 2-byte field that receives the device address of the bare disk.
Volume	is a 4-byte field that receives the device mnemonic for the bare disk.
Flags	is a 4-byte field that receives the device flags for the bare disk.
Size	is a 4-byte field that receives the number of sectors on the bare disk.
Tracks per cylinder	is a 2-byte field that receives the number of tracks per cylinder on the bare disk.
Sectors per track	is a 2-byte field that receives the number of sectors per track on the bare disk.
Controller address	is a 2-byte field that receives the controller address for the bare disk device.
SELCH address	is a 2-byte field that receives the selector channel address for the bare disk device.

7.3.3 SVC7: Device Rename

E-tasks can use the SVC7 rename function to rename devices. The e-task must have the device already assigned to the lu with ERW access privileges.

The SVC7 parameter block format and coding for renaming devices is shown is Figure 7-13.




```
          SVC   7,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    X'1000'
          DB    0
          DB    lu
          DC    0
          DC    C'4-character device mnemonic'
          DB    0,0,0,0
```

Figure 7-13 SVC7 Device Rename Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary-aligned, and located in a task-writable segment. A description of each field in the parameter block follows.

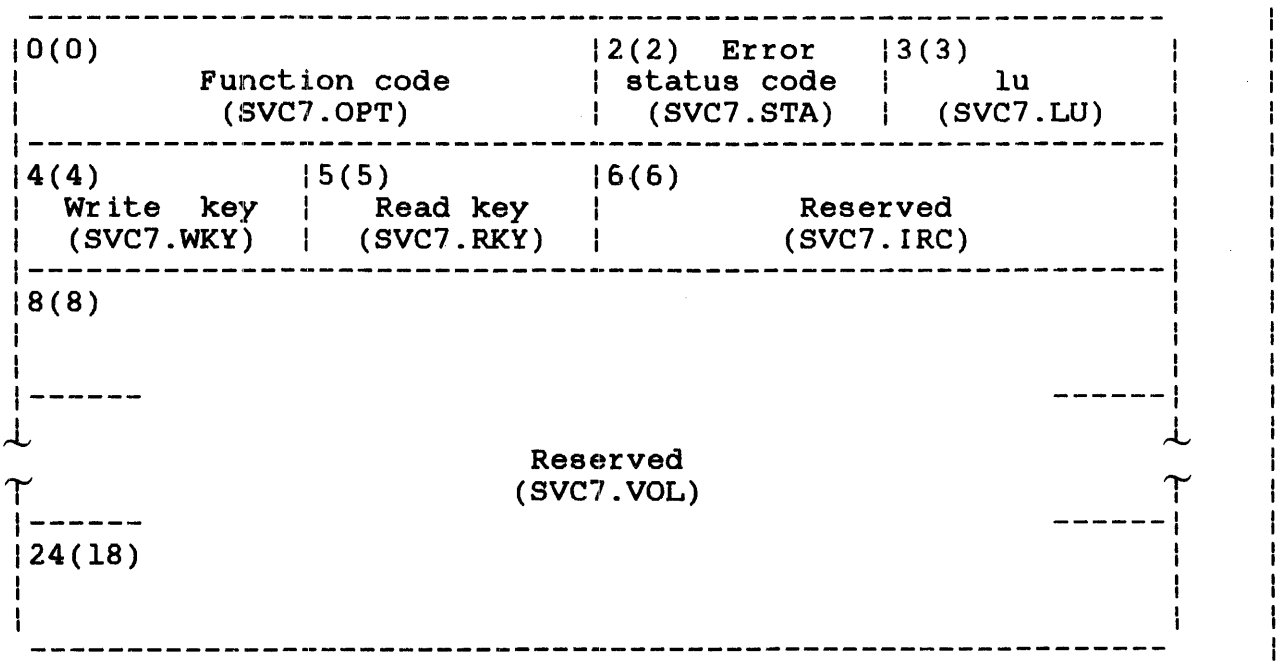
Fields:

Function code (SVC7.OPT)	is a 2-byte field that contains the hexadecimal number X'1000' indicating that the rename function is to be performed.
Error status codes (SVC7.STA)	is a 1-byte field that receives an error code when an error occurs during SVC7 execution. If no error occurs, a value of 0 is returned is returned to this field. See Table 7-6 for a list of SVC7 error codes.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit assigned to the device that is to be renamed.
Reserved (SVC7.WKY)	is a 4-byte reserved field that must contain a zero.
Device mnemonic (SVC7.VOL)	is a 4-byte field containing the device mnemonic that is to replace the current device mnemonic in the device mnemonic table.
Reserved (SVC7.FNM)	is a 16-byte field that must be reserved with zeros.

7.3.4 SVC7: Device Reprotect

E-tasks can use the SVC7 reprotect function to define the type of access allowed to a device (e.g., read-only, write-only, etc.). The e-task must have the device already assigned to the lu with ERW access privileges.

The SVC7 parameter block format and coding for reprotecting devices is shown in Figure 7-14.




```
          SVC   7,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    X'0800'
          DB    0
          DB    lu
          DB    'write key'
          DB    'read key'
          DC    H'0'
          DC    0
          DB    0,0,0,0
```

Figure 7-14 SVC7 Device Reprotect Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary-aligned, and located in a task-writable segment. A description of each field in the parameter block follows.

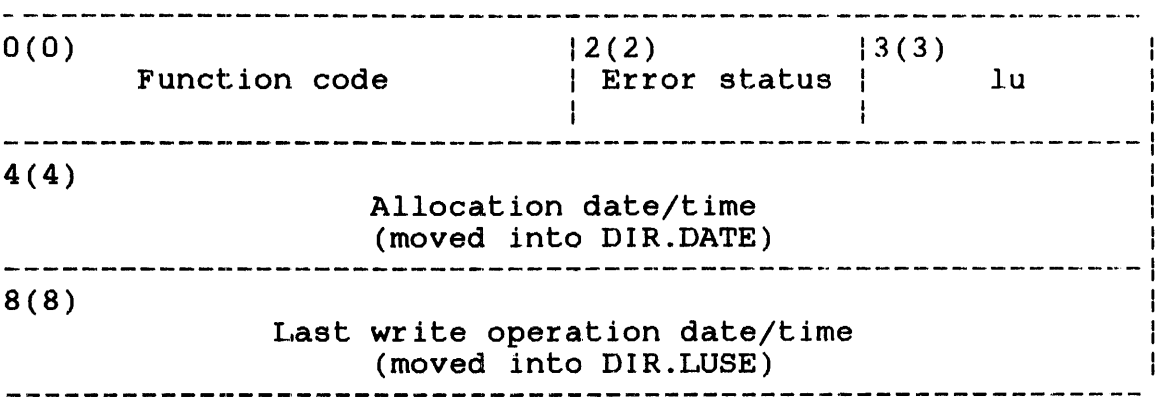
Fields:

Function code (SVC7.OPT)	is a 2-byte field containing the hexadecimal number X'0800' indicating that the reprotect function is to be performed.
Error status codes (SVC7.STA)	is a 1-byte field that receives an error code when an error occurs during SVC7 execution. If no error occurs, a value of 0 is returned to this field. See Table 7-6 for a list of SVC7 error codes.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit assigned to the device that is to be reprotected.
Write key (SVC7.WKY)	is a 1-byte field that contains a hexadecimal number indicating the new write protection keys for the device.
Read key (SVC7.RKY)	is a 1-byte field that contains a hexadecimal number indicating the new read protection keys for the device.
Reserved (SVC7.LRC)	is a 2-byte reserved field that must contain a zero.
Reserved (SVC7.VOL)	is a 20-byte unused field that should be initialized with zeros.

7.3.5 SVC7 Code X'FF80': Extended Close Function

The extended close function closes an lu and replaces the date and time of allocation and the last write (or write filemark) operation in the disk directory with information stored in the SVC7 parameter block. This SVC7 function is available only to e-tasks or privileged u-tasks and d-tasks with the bare disk task option enabled.

Figure 7-15 shows the parameter block format and coding for SVC7 code X'FF80'.




```
          SVC   7,parblk
          .
          .
          .
          ALIGN 4
parblk    DC    X'FF80'
          DB    1
          DB    lu
          DC    Y'allocation date/time'
          DC    Y'last write operation date/time'
```

Figure 7-15 SVC7 Code X'FF80' Parameter Block Format and Coding

This parameter block must be 12 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Function code — is a 2-byte field that contains the function code X'FF80' indicating that the SVC7 extended close function is to be performed.

Error status — is a 1-byte field that receives an error code when an error occurs during SVC7 execution. If no error occurs, a value of 0 is returned to this field. If a nonprivileged task attempts to execute this SVC, a value of 1 is returned. See Table 7-6 for a list of SVC7 error codes.

Allocation date/time	is a 4-byte field that contains the date and time that is to replace the date and time of allocation in the DIR.DATE field of the disk directory. The format of the date and time must be the same format generated by the DATE.DIR routine in the File Manager Utility (FMUT) module.
Last write operation date/time	is a 4-byte field that contains the date and time that is to replace the date and time of the last write operation in the DIR.LUSE field of the disk directory. The format of the date and time must be the same format generated by the DATE.DIR routine in the FMUT module.

7.4 SVC7 ERROR CODES

If an error occurs during execution of an SVC7 function, execution of the current function stops, and any other functions to the right of the current function are not executed. The error code indicating the type of error is stored in the error status field of the parameter block. See Table 7-6 for the list of SVC7 error codes.

TABLE 7-6 SVC7 ERROR CODES

ERROR CODE	FUNCTIONS AFFECTED	MEANING
0	All	Normal termination
1	All	Illegal function code
2	All except allocate	Illegal lu specified
3	All except rename	Specified volume is not mounted.
4	Allocate rename	Specified filename already exists on specified volume.
	Assign	Specified filename does not exist on specified volume.
5	Allocate	Insufficient space exists on specified volume to allocate a file of the specified size.

TABLE 7-6 SVC7 ERROR CODES (Continued)

ERROR CODE	FUNCTIONS AFFECTED	MEANING
6	Assign	Read/write protection keys do not match.
	Change	Read/write protection keys do not match.
	Access privilege	
7	Allocate assign	Entire disk is currently assigned as ERW. Specified filename or device cannot be assigned because requested access privileges cannot be granted.
	Change access privilege	Current access privileges are not changed to new access privileges because the requested access privileges cannot be granted.
	Reprotect	File is not assigned ERW.
	Delete	File assigned to another task (not closed).
	Rename	Read/write protection keys do not match. File is not assigned ERW.
8	Assign	There is insufficient space for file control block (FCB) and buffers.
	Close	System space pointer or pointers have become corrupted.
	Delete	Task has exhausted its allocation of dynamic system space determined by Link.
9	Assign	The lu is already assigned or device is off-line. Magnetic tape drive does not support software density selection.
	Rename	The lu is not assigned.
	Reprotect	
	Close	
	Fetch attributes	
	Change access privileges	

TABLE 7-6 SVC7 ERROR CODES (Continued)

ERROR CODE	FUNCTIONS AFFECTED	MEANING
	VFC	
A	Allocate rename	Specified volume is not a direct access device.
B	Reprotect all	The fd format is incorrect.
C	Assign	Specified trap-generating device does not exist in the system, is not a connectable device or is busy and cannot be connected.
D	Allocate delete	Allocation or deletion was attempted on a system or group file.
	Spoolfile assign to pseudo device	SPL/32 is inactive.
E	Spoolfile assign to pseudo device	Specified form is not recognized by SPL/32.
F	Spoolfile Assign to Pseudo Device	There are conflicting options.
10-7F	N/A	Reserved
80-FF	N/A	SVC1 I/O error; see Tables 2-3 and 2-4.

CHAPTER 8
LOAD TASK STATUS WORD (TSW) SUPERVISOR CALL 9 (SVC9)

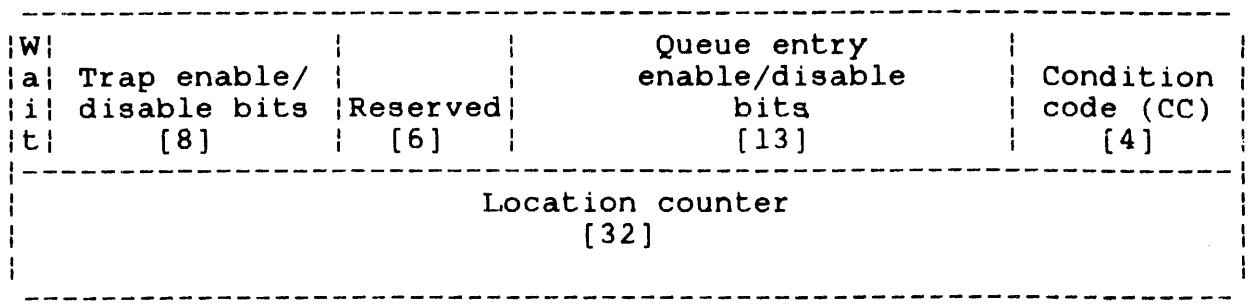
8.1 INTRODUCTION

SVC9 sets the initial TSW or replaces the current TSW located in the task control block (TCB) with a new user-specified TSW. The SVC9 parameter block is shown in Figure 8-1. Other methods used for setting the TSW are:

- The TSW is optionally specified by Link.
- A resident task terminates by reaching end of task, which causes the current TSW to be replaced with zeros.
- A task trap occurs causing a TSW swap.

Storing TSW values into the user-dedicated location (UDL) does not change the current TSW.

8.2 SVC9: LOAD TASK STATUS WORD (TSW)




```
            SVC   9,A(parblk)
            .
            .
            .
            ALIGN 4
parblk      DC    Y'bits 0 through 31'
            DCF   A(location counter)
```

NOTE

The decimal numbers enclosed within brackets ([ ]) in the parameter block indicate the number of bits the field contains.

Figure 8-1 SVC9 Parameter Block Format and Coding

This parameter block must be eight bytes long and fullword boundary-aligned. A description of each field in the parameter block follows.

Fields:

Wait — is a 1-bit field indicating whether the task is to enter a suspended state or is currently waiting for a trap. This field corresponds to bit 0 of the TSW.

Trap enable/ disable bits — is an 8-bit field that must indicate, through its trap bit settings, whether a trap is to be taken when a trap-causing condition occurs. This field corresponds to the trap enable/disable bits of the TSW.

Reserved	is a reserved 6-bit field that must contain zeros.
Queue entry enable/ disable bits	is a 13-bit field that must indicate, through its queue bit settings, whether an item is to be added to the task queue when a queue entry-causing condition occurs. This field corresponds to the queue entry enable/disable bits of the TSW.
Condition code (CC)	is a 4-bit field stored in the processor CC. For an explanation of the CC, see the appropriate processor user's manual. This field corresponds to the CC bits of the TSW.
Location counter	is a 4-byte field that must contain the address where task execution is to start or resume. This field corresponds to the location counter (LOC) of the TSW.

8.2.1 Function and Description of the Task Status Word (TSW)

The TSW consists of two fullwords (see Figure 8-2). The first fullword, the status portion of the TSW, contains the:

- Trap wait bit
- Trap enable/disable bits
- Reserved bits
- Queue entry enable/disable bits
- CC bits

The second fullword of the TSW contains the LOC.

SVC9 allows the user to enable or disable the trap wait, trap and queue entry bits in the status portion of the TSW. It also allows the user to set the CC setting in the status portion and the LOC in the counter portion of the TSW. See Table 8-1 for the TSW bit definitions.

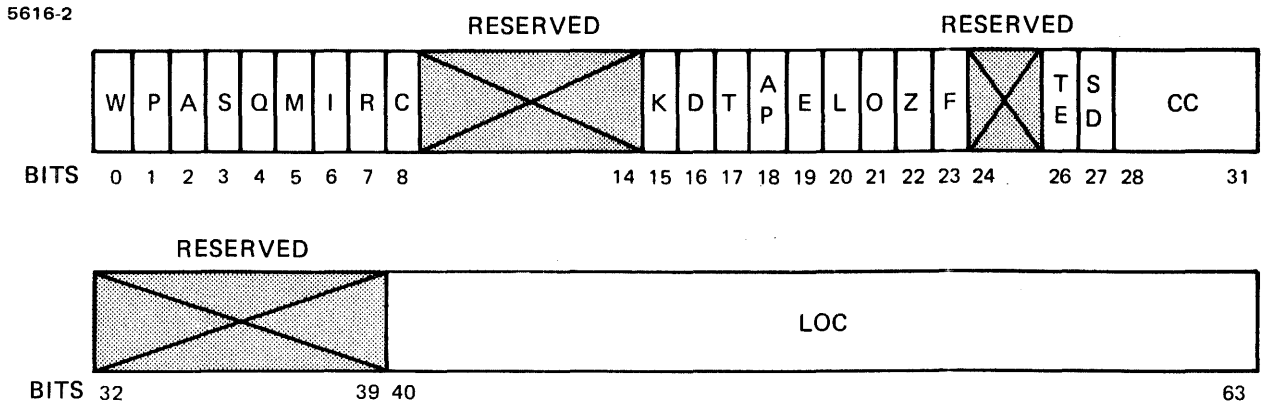


Figure 8-2 Task Status Word (TSW)

TABLE 8-1 TSW BIT DEFINITIONS

BIT POSITION	BIT NAME AND MASK	MEANING
0 (W)	Trap wait (TSW.WTM) (Y'80000000')	Task is suspended until a trap occurs or until cancelled.
1 (P)	Power restoration trap enable/disable (TSW.PWRM) (Y'40000000')	A trap occurs when power is restored after a power failure. After power is restored, all outstanding timer traps are lost. Any trap wait or time wait conditions in effect are lost, and task execution continues with the instruction following the one that caused the trap.
2 (A)	Arithmetic fault trap enable/disable (TSW.AFM) (Y'20000000')	A trap occurs when an arithmetic fault occurs.
3 (S)	SVC14 execution trap enable/disable (TSW.S14M) (Y'10000000')	Allows execution of SVC14.

TABLE 8-1 TSW BIT DEFINITIONS (Continued)

BIT POSITION	BIT NAME AND MASK	MEANING
4 (Q)	task queue service trap enable/disable (TSW.TSKM) (Y'08000000')	A trap occurs when an item is added to the task queue or when at least one item exists on the queue.
5 (M)	Memory access fault trap enable/disable (TSW.MAFM) (Y'04000000')	A trap occurs when the task attempts to access memory outside its task boundaries.
6 (I)	Illegal instruction trap enable/disable (TSW.IITM) (Y'02000000')	A trap occurs when the task tries to execute an illegal instruction.
7 (R)	Data format trap enable (TSW.DFFM) (Y'01000000')	A trap is taken when the task executes an instruction that causes a data format or alignment fault.
8 (C)	Central processing unit (CPU)-override status (TSW.CPOM) (Y'00800000')	Task is executed on the CPU and cannot be transferred to an auxiliary processing unit (APU) for processing. (This bit applies only to tasks running on the Model 3200MPS System.)
9-14	Reserved	Must contain zeros.
15 (K)	Subtask queue entry enable/disable (TSW.SUQM) (Y'00010000')	An item is added to the monitor task queue each time the subtask status changes.
16 (D)	Device interrupt queue enable/ disable (TSW.DIQM) (Y'00008000')	An item is added to the task queue when a trap-generating device connected to a task interrupts task execution, or when an SVC6 sint function is directed to a task.
17 (T)	Task call queue entry enable/ disable (TSW.TCM) (Y'00004000')	An item is added to the task queue when an SVC6 queue parameter function is directed to this task.

TABLE 8-1 TSW BIT DEFINITIONS (Continued)

BIT POSITION	BIT NAME AND MASK	MEANING
18 (AP)	Queue entry on signal from APU (TSW,APTM) Y'0000 2000'	Adds a parameter to the task queue when an APU signals the CPU.
19 (E)	Task Message Queue Entry Enable/Disable (TSW.PMM) (Y'00001000')	An item is added to the task queue when an SVC6 send message function is directed to a task.
20 (L)	Load and Proceed Completion Queue Entry Enable/ Disable (TSW.LODM) (Y'00000800')	An item is added to the task queue when an SVC6 load and proceed function is executed and the load is completed.
21 (O)	Input/Output (I/O) Completion Entry Enable/Disable (TSW.IOM) (Y'00000400')	An item is added to the task queue when an SVC1 I/O and proceed function is executed and the I/O is completed.
22 (Z)	Time Interval Completion Queue Enable/Disable (TSW.TMCM) (Y'00000200')	An item is added to the task queue when an SVC2 code 23 is executed and the interval has elapsed.
23 (F)	SVC15 Function SVC1 Buffer Transfer Completion (TSW.ITM) (Y'00000100')	An item is added to the task queue when an SVC15 function is completed. See the OS/32 Basic Data Communications Reference Manual. An item is added to the task queue each time the magnetic tape driver adds a buffer to the OUT-QUEUE.
24-25	Reserved	Must contain zeros.
26 (TE)	Event Queue Service Enable/Disable (TSW.TESM) (Y'00000020')	A trap occurs when an item is added to the system event queue or when at least one item exists on that queue. For more information on the event queue service enable/ Disable, see the OS/32 System Level Programmer Reference Manual.

TABLE 8-1 TSW BIT DEFINITIONS (Continued)

BIT POSITION	BIT NAME AND MASK	MEANING
27 (SD)	Queue entry on send data call enable (TSW.SDM) (Y'00000010)	An item is added to the queue when an SVC6 send message function is directed to the task.
28-31 (CC)	CC	The CC following SVC9 is set from these bits.
32-63 (LOC)	Location counter (TSW.LOC)	Contains the current location counter.

NOTE

See the OS/32 Application Level Programmer Reference Manual for a description of the items that can be added to the task queue.

If execution of an SVC9 loads a TSW with the trap wait bit enabled, the task is placed in a suspended state until one of the traps that are enabled in the same TSW occurs. However, if the task is placed in a suspended state and all other trap bits are disabled in the same TSW, the task remains in a suspended state indefinitely or until it is cancelled.

If execution of an SVC9 loads a TSW with one of the trap bits enabled and that trap occurs, the trap is handled as described in the OS/32 Application Level Programmer Reference Manual.

If execution of an SVC9 loads a TSW with one of the queue entry bits enabled and a previously allocated item is placed on the task queue, no trap occurs unless the queue service trap bit of the TSW is enabled.

When a TSW swap occurs and the current TSW is replaced with a new TSW, task execution resumes with the instruction located at the address specified by the LOC of the new TSW. If the address of the new TSW is outside the task boundaries, the task is paused and a message is displayed. If execution of an SVC9 loads a TSW that has zeros in the LOC field, execution resumes with the instruction following the SVC9.

When SVC9 loads a new TSW, the CC of the new TSW becomes the current CC. Any value ranging from 0 to 15 (X'00' to X'0F') is legal. If the TSW being loaded was previously saved as an old TSW during a TSW swap, the CC is restored.

CHAPTER 9
OVERLAY LOADING SUPERVISOR CALL 10 (SVC10)

9.1 SVC10: OVERLAY LOADING

SVC10 is an internal call that provides for the automatic loading of overlays generated by Link. SVC10 is not available to users.

If an overlay load fails to occur, a message indicating the reason for the failure is displayed to the log device. Overlay load failure can result from an input/output (I/O) error or from faulty coding that destroys the overlay control structure. For example, user code can be written in such a way as to destroy data in the overlay reference table. This table, which forms a part of the root segment and of each overlay area, contains pointers into the task overlay descriptor table (ODT), which contains the information needed to process the overlay. Without this information, SVC10 cannot perform the load function.

The overlay descriptor table entry (ODTE) is part of the overlay reference table and represents the position in the ODT that contains processing information for the overlay to be loaded. Both the overlay reference table and the ODT are operating system data structures and are defined in the system macro library.

9.2 MESSAGES

The following message is displayed when a load failure occurs as a result of an I/O error.

Format:

```
I/O ERROR xxxx LOADING OVERLAY nnnnnnnn
FAULT LOCATION yyyyyy (zzzzzz)
```

Where:

xxxx	is the I/O error status (see Table 9-1).
nnnnnnnn	is the name of the overlay that was being processed when the error occurred.
yyyyyy	is the virtual address of the SVC that caused entry into the SVC10 handler.
zzzzzz	is the physical address of yyyyyy.

The following message is displayed when an overlay load failure occurs as a result of faulty coding within an overlay control structure.

Format:

```
OVERLAY ERROR xx NAME = nnnnnnnn
FAULT LOCATION yyyyyy (zzzzzz)
```

Where:

xx	is the error status. See Table 9-1 for error definitions.
nnnnnnnn	is the name of the overlay that was being processed when the error occurred. If it cannot be determined whether the error occurred in the root or in an overlay, NAME = nnnnnnnn is omitted from the message.
yyyyyy	is the virtual address of the SVC that caused entry into the SVC10 handler.
zzzzzz	is the physical address of yyyyyy.

If the overlay load failure resulted from a malfunction of SVC10, the task is paused with the current program status word (PSW) pointing to the SVC10 instruction causing the failure.

TABLE 9-1 OVERLAY ERROR CODES AND MEANINGS

ERROR CODE	MEANING
10*	.ODTE exists outside the range of the ODT range.
20	There is user space violation of overlay start address.
21	There is user space violation of overlay end address.
22	Highest level overlay required by this SVC was not found.
23	OVL size is less than 10 bytes; eight bytes for two fullword overlay reference table entry pointers plus a 2-byte instruction (BR) is the minimum size for an overlay.
30	There is user space violation of the overlay reference table address in ODT entry.
31*	Pointers to overlay reference table entries are unreliable. The address difference between these pointers must be zero or an even multiple of eight bytes.
32*	There is user space violation of overlay reference table entry pointers.
33*	.ODTE index in ORT is out of ODT range.

* Indicates possible destruction of data

CHAPTER 10
AUXILIARY PROCESSING UNIT (APU) CONTROL SUPERVISOR CALL 13 (SVC13)

10.1 SVC13: AUXILIARY PROCESSING UNIT (APU) SERVICES

SVC13 provides a task with an interface to the APU in a Model 3200MPS System environment. SVC13 gives a task the ability to:

- access status information on all APUs and APU task queues in the system,
- direct the flow of tasks to an APU task queue, and
- direct the execution of tasks on an APU.

Table 10-1 lists the SVC13 function codes, which provide these capabilities.

Functions 0 and 1 are available to any task in a Model 3200MPS System. Function 2 is available only to tasks in a Model 3200MPS System that have been linked with the APMAPPING task option. Function 3 is available only to tasks in a Model 3200MPS System that have been linked with the APCONTROL task option. See the OS/32 Link Reference Manual for more information on building a task with these task options set.

TABLE 10-1 SVC13 FUNCTION CODES

FUNCTION CODE	MEANING
SVC13 code 0	Read APU assignment and mapping information.
SVC13 code 1	Read APU/APU queue status.
SVC13 code 2	Execute queue mapping option.
SVC13 code 3	Execute APU control option.

The following sections describe how to access each of the SVC13 functions from an application task running on a Model 3200MPS System.

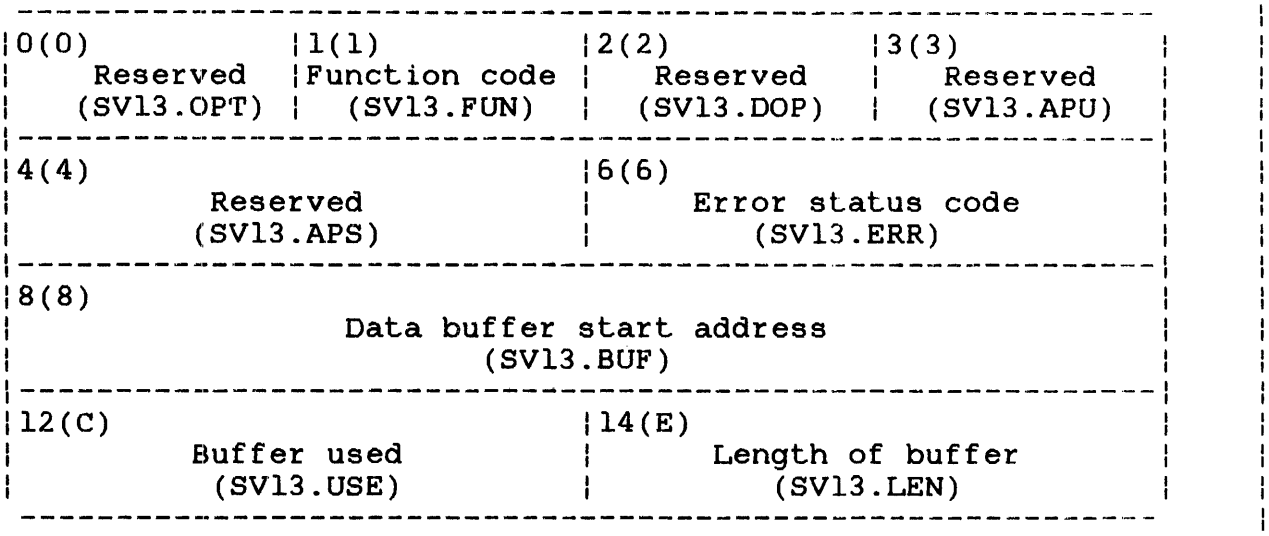
10.2 SVC13 CODE 0: READ AUXILIARY PROCESSING UNIT (APU) ASSIGNMENT AND MAPPING INFORMATION

SVC13 code 0 enables a task to copy onto a data buffer the maximum number of APUs (m) and logical processing units (LPUs) (n) allowed on the system followed by the tables of APU-to-APU queue assignment and of LPU-to-APU queue mapping. Both m and n are determined at OS/32 system generation (sysgen). See the System Generation/32 (Sysgen/32) Reference Manual.

The table of APU-to-APU queue assignment contains one entry for the central processing unit (CPU) and each APU number. Each entry is 1-byte, sequentially arranged starting with the entry for CPU (always 0), then for APU1 and ending with the entry for APUn, and containing the number of APU queue to which the respective processor is assigned. The APU gueue number value varies from 0 to n.

The table of LPU-to-APU queue mapping contains one entry for each LPU number. Each entry is 1-byte, sequentially arranged starting with the entry for LPU0 (always 0) and ending with the entry for LPUn, and containing the number of the APU queue to which the LPU is mapped. The APU queue number varies from 0 to n.

The data buffer where the above information is copied must be located in a writable segment of the task's address space. This buffer must begin on a fullword boundary. Figure 10-1 shows the parameter block and coding for SVC13 code 0.




```
          SVC   13,parblk
          .
          .
          .
          ALIGN 4
parblk    DB    0,0,0,0
          DB    H'0'
          DS    2
          DC    A(BUFFER)
          DS    2
          DC    H'number of bytes'
```

Figure 10-1 SVC13 Code 0 Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows:

Fields:

Reserved (SV13.OPT)	is a 1-byte unused field that should be initialized to zero.
Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 0 to indicate SVC13 code 0.
Reserved (SV13.DOP)	is a 1-byte unused field that should be initialized to zero.
Reserved (SV13.APU)	is a 1-byte unused field that should be initialized to zero.

Reserved (SV13.APS)	is a 2-byte unused field that should be initialized to zero.
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC13 code 0. See Table 3-10 for a list of the SVC13 status codes.
Data buffer start address (SV13.BUF)	is a 4-byte field that contains the address of a user-specified buffer to which the operating system returns the assignment and mapping information. The buffer can be variable in length but must begin on a fullword boundary in a task-writable segment.
Buffer used (SV13.USE)	is a 2-byte field that receives the actual number of bytes used in the buffer specified by the SV13.BUF field.
Length of buffer (SV13.LEN)	is a 2-byte field that contains a decimal number indicating the maximum length (in bytes) of the data buffer specified in the SV13.BUF field.

When SVC13 code 0 is executed, APU assignment and mapping information is returned to the user's buffer in the format shown in Figure 10-2.

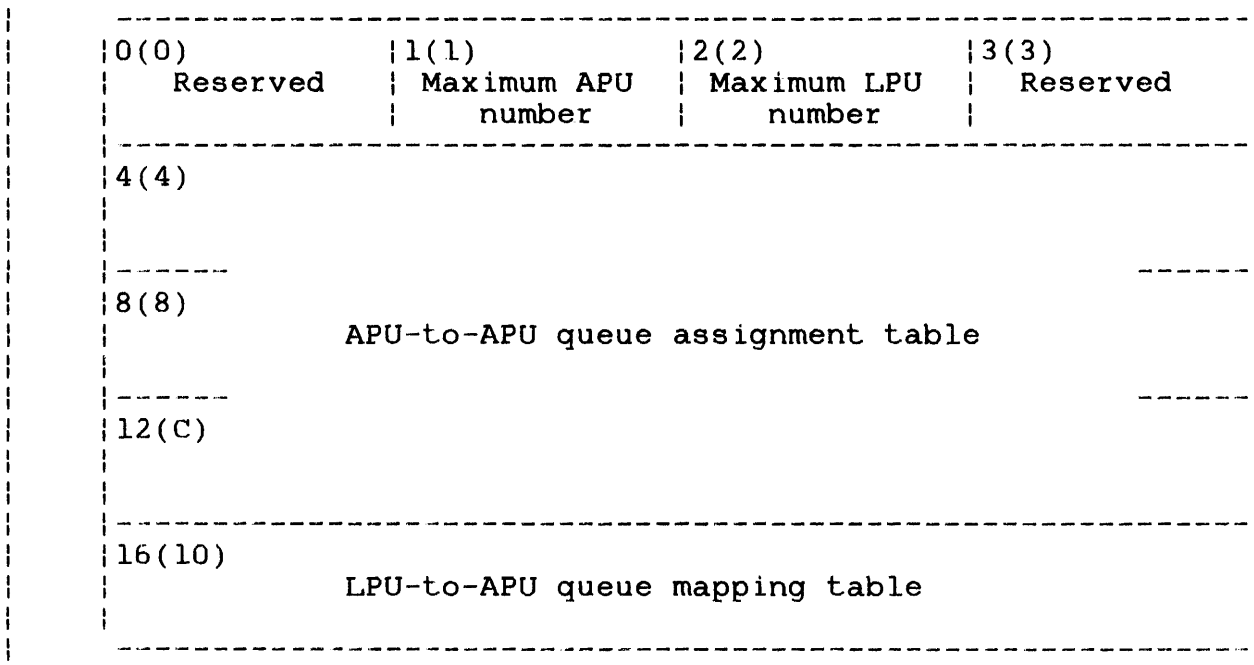


Figure 10-2 Data Buffer Format for SVC13 Code 0

Fields:

Reserved	is a 1-byte field that is reserved for future use.
Maximum APU number	is a 1-byte field containing the maximum APU number allowed in the system. This number is determined at sysgen.
Maximum LPU number	is a 1-byte field containing the maximum LPU number allowed in the system. This number is determined at sysgen.
Reserved	is a 1-byte field that is reserved for future use.
APU-to-APU queue assignment table	is a variable length array (0:m) of 1-byte entries, each entry number i containing the number of the APU queue to which APUi is mapped, where APU0 = CPU. The CPU entry is always 0. For example, in a system with four APUs, this table might represent byte sequence 003230000000, where the CPU entry is 0, APU2 and APU4 are assigned to queue 3, and APU3 is assigned to queue 2.
LPU-to-APU queue mapping table	is a variable length array (0:n) of 1-byte entries, each entry number k containing the number of the APU queue to which LPUk is mapped. The LPU0 is always mapped to queue 0. For example, in a system with six LPUs, this table might represent a byte sequence 021033xx, where LPU0 and LPU3 are mapped to queue 0, LPU1 is mapped to queue 2, LPU2 is mapped to queue 1, LPU4 and LPU5 are mapped to queue 3, and the table is filled with undefined bytes x to the fullword boundary.

10.3 SVC13 CODE 1: READ AUXILIARY PROCESSING UNIT (APU)/APU QUEUE STATUS

SVC13 code 1 allows a task to access information on the status of each APU and/or APU queue in a Model 3200MPS System. The APU status includes the:

- APU hardware status
- Number of the APU queue to which APU is assigned
- Status of APU writable control store (WCS)
- APU software status

- APU software status after power fail
- Configuration options set for the APU
- Name of the task currrently active on the APU or of the task for which the APU is waiting
- Name of task with control rights over the APU
- WCS image file descriptor (fd)

The APU queue status includes the:

- Queue processing status
- Number of APUs assigned to the queue
- Number of LPUs mapped to the queue
- Name of the task with mapping rights over the queue
- Names of all tasks in the queue (or name of the task having exclusive rights to the queue)

Figure 10-3 shows the parameter block format and coding for SVC13 code 1.

0(0) Options (SV13.OPT)	1(1) Function code (SV13.FUN)	2(2) Reserved (SV13.DOP)	3(3) APU/APU queue number (SV13.APN)
4(4) APU hardware status (SV13.APS)		6(6) Error status code (SV13.ERR)	
8(8) Data buffer start address (SV13.BUF)			
12(C) Buffer used (SV13.USE)		14(E) Length of buffer (SV13.LEN)	

```
          SVC   13,parblk
          .
          .
          .
parblk    ALIGN 4
          DB    0,1,0
          DC    'APU or APU queue number'
          DS    4
          DC    A(buffer)
          DS    2
          DC    H'number of bytes in buffer'
```

Figure 10-3 SVC13 Code 1 Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

Options (SV13.OPT) contain one or both of the following fetch status functions:

OPTION	FUNCTION
X'80'	Fetch APU status.
X'40'	Fetch APU queue status.

NOTE

If both options are specified, the APU status is fetched; then the status of the queue to which the APU is assigned is fetched.

Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 1 to indicate SVC13 code 1.
Reserved (SV13.DOP)	is a 1-byte unused field that should be initialized to zero.
APU/APU queue number (SV13.APN)	is a 1-byte field specifying a decimal number that represents: • APU number (1-9) to which this call is directed if option X'80' is specified. • APU queue number (0-9) to which this call is directed. Applicable only if option X'40' is specified by itself.
APU hardware status (SV13.APS)	is a 2-byte field that receives the APU response status from the APU processor hardware for option X'80'.
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC13 code 1. The first byte of this field indicates bit position (0 through 1) of the SVC13 code 1 option being executed when the error occurred. The second byte contains one of the SVC13 error status codes. See Table 10-10 for a list of the SVC13 status codes. If no error occurs, both bytes contain 0.
Data buffer start address (SV13.BUF)	is a 4-byte field containing the address of a data buffer to which SVC13 is to return the fetched status information. The buffer can be variable in length, but it must begin on a fullword boundary and be located in a task-writable segment. If the buffer is less than eight bytes in length, an error code (insufficient buffer space) is returned and no data is written to the buffer.
Buffer used (SV13.USE)	is a 2-byte field that receives a decimal number indicating the actual number of bytes used in the buffer specified by the SV13.BUF field.
Length of buffer (SV13.LEN)	is a 2-byte field that contains a decimal number indicating the maximum length (in bytes) of the data buffer specified in the SV13.BUF field.

When SVC13 code 1 option X'80' only is executed, information on the status of the specified APU is returned to the user's buffer in the format shown in Figure 10-4.

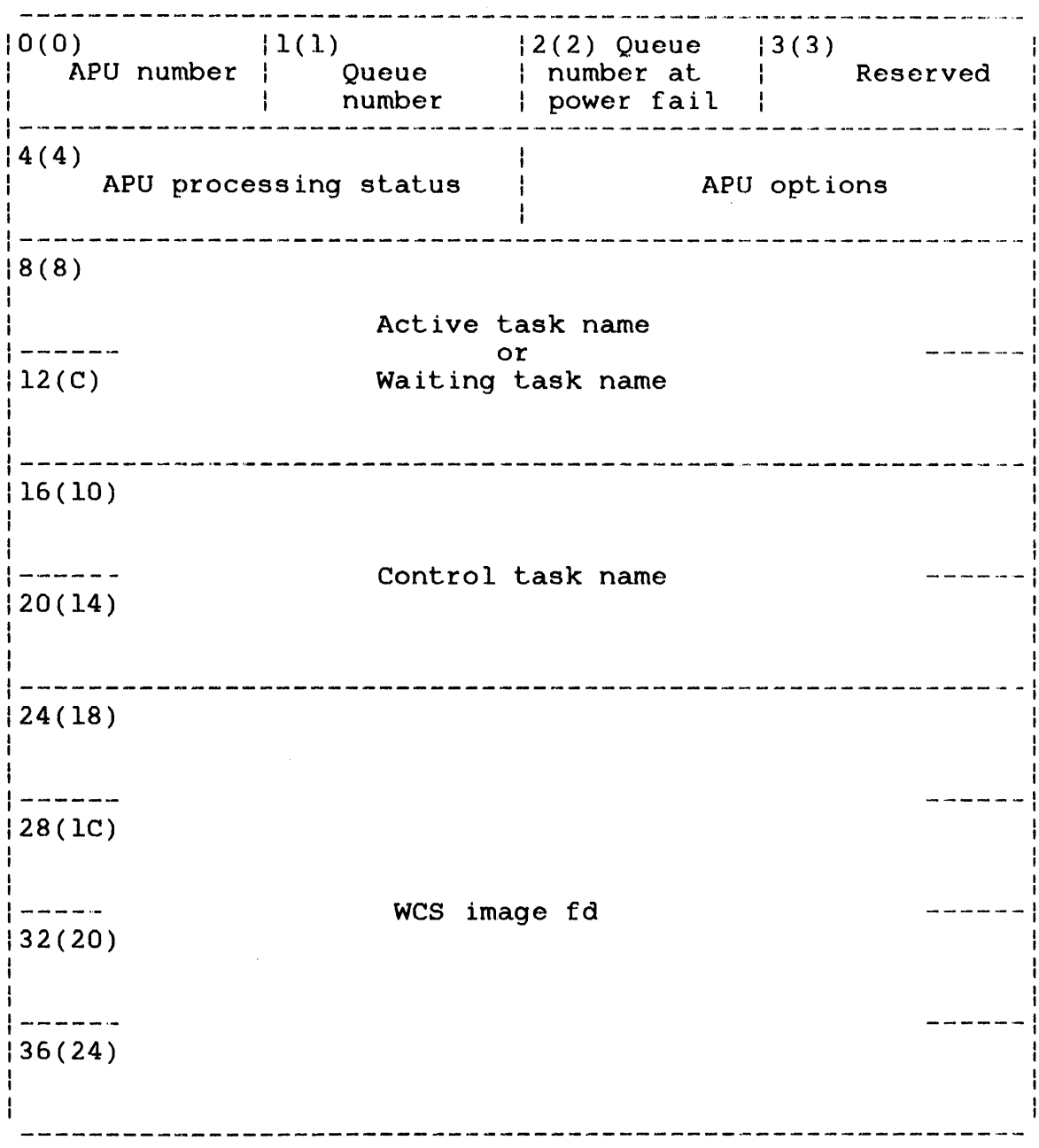


Figure 10-4 Data Buffer Format for SVC13 Code 1 Option X'80' only

Fields:

APU number	is a 1-byte field containing the number of the APU to which the status information applies.
Queue number	is a 1-byte field containing the number of the APU queue to which the APU is assigned.
Queue number at power fail	is a 2-byte field containing the number of the APU queue to which the APU has been assigned at the time of the last power fail.
APU processing status	is a 2-byte field that contains the WCS state for the specified APU, the current APU software state and the APU state at the time of the last power fail. Figure 10-5 shows the APU processing status field. See Table 10-2 for the bit definitions for this field.
APU options	is a 2-byte field that is set according to the configuration options in effect for the specified APU. See Table 10-3 for the bit definitions for this field.
Active task name or waiting task name	is an 8-byte field containing the name of the currently active task. If the APU is stopped and waiting for a task, this field contains the name of the currently waiting task. The task name is left-justified with trailing blanks. If no currently active or waiting task exists, the entire field is filled with blanks.
Control task name	is an 8-byte field containing the name of the task having control rights over the specified APU. If no control task exists, the entire field is filled with blanks.
WCS image fd	is a 16-byte field containing the file descriptor of a writable control store image file loaded into the APU during its power-up initialization.

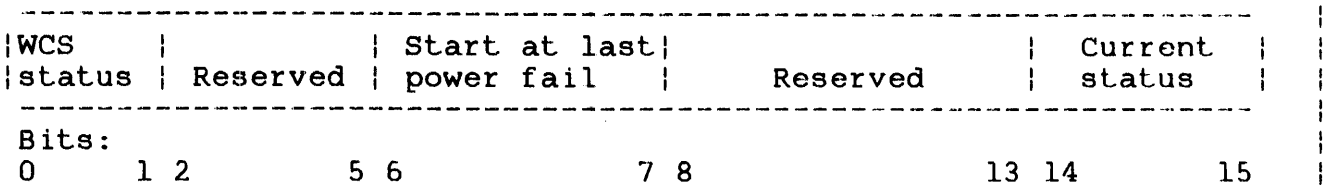


Figure 10-5 Format of APU Processing Status Field Returned to U-Task Buffers

TABLE 10-2 BIT DEFINITIONS FOR APU PROCESSING STATUS FIELD RETURNED TO U-TASK BUFFER

APU STATUS	BIT POSITION	BIT SETTING AND MEANING
WCS state	0	1 = WCS initialized 0 = WCS not initialized
	1	1 = WCS loaded 0 = WCS not loaded
Reserved for future use	2-5	
State at last power fail	6-7	00 = APU disabled 10 = APU enabled 11 = APU enabled and waiting for task
Reserved for future use	8-13	
Current status	14-15	00 = APU disabled 10 = APU enabled 11 = APU enabled and waiting for task

TABLE 10-3 BIT DEFINITION FOR APU OPTIONS FIELD RETURNED TO U-TASK BUFFER

APU CONFIGURATION OPTION	BIT POSITION	BIT SETTING AND MEANING
WCS	0	0 = APU has WCS 1 = APU has no WCS
Floating point support	1	0 = APU has floating point support 1 = APU has no floating point support
Trap block wait	2	0 = APU will continue processing during CPU fault handling 1 = APU will stop processing and wait for a task during CPU fault handling
Reserved for future APU options	3-15	

When SVC13 code 1 option X'40' is executed, information on the status of the specified APU queue is returned to the user's buffer in the format shown in Figure 10-6.

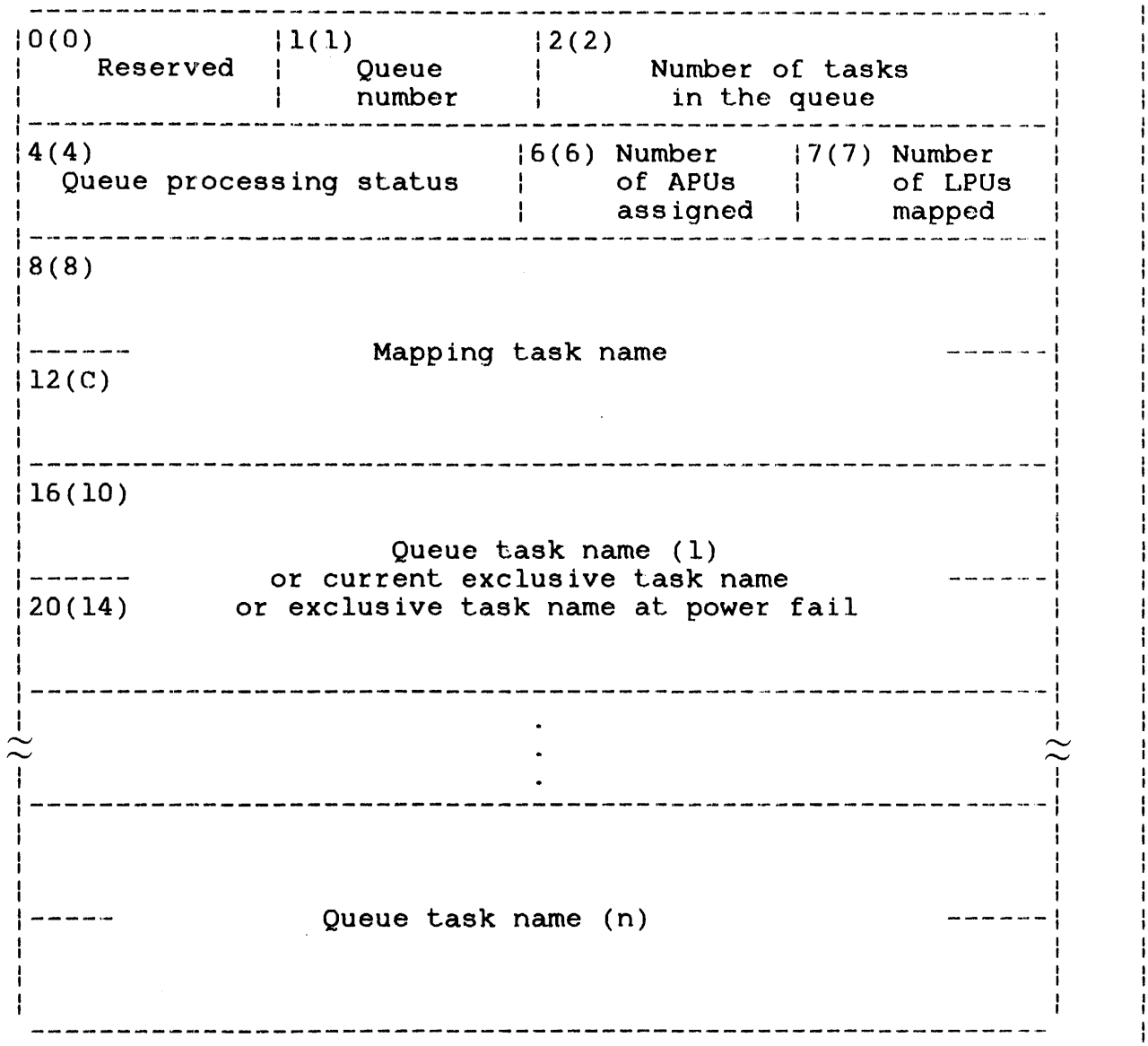


Figure 10-6 Data Buffer Format for SVC13 Code 1 Option X'40' Only

Fields:

Queue number	is a 1-byte field containing the number of the queue to which the status information applies.
Number of tasks in the queue	is a 2-byte field containing the total number of tasks waiting in the specified APU's queue. Any task currently executing on the APU is not included in the total.
Queue processing status	is a 1-byte field that contains the current queue status and the queue status at the time of the last power fail. Figure 10-7 shows the APU queue processing status field. See Table 10-4 for the bit definitions for this field.
Number of APU assigned	is a 1-byte field containing the number of APUs assigned to the specified queue.
Number of LPUs mapped	is a 1-byte field containing the number of logical processor mapping table (LPMT) entries mapped to the specified APU.
Mapping task name	is an 8-byte field containing the name of the task having mapping rights over the specified APU. If no mapping task exists, the entire field is filled with blanks.
Queue task name (n) or exclusive task name	is a variable length table of 8-byte fields containing the name of each task in the APU queue. The order of entries corresponds to the order of the tasks in the queue. When the specified queue has been marked on exclusively for one task, the queue is always empty. In this case, this field contains the name of the task having exclusive rights to the queue.

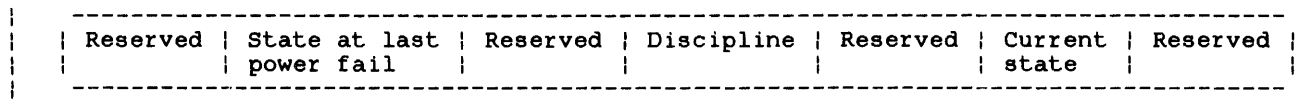


Figure 10-7 Format of APU Queue Processing Status Field Returned To U-Task Buffer

TABLE 10-4 BIT DEFINITIONS FOR APU PROCESSING STATUS FIELD RETURNED TO U-TASK BUFFER

QUEUE STATUS	BIT POSITION	BIT SETTING AND MEANING
Reserved for future use	0-2	
State at last power fail	3 and 4	00 = queue OFF 01 = queue ON 11 = queue ON-exclusive
Reserved for future use	5 through 7	
Queue Discipline	8 and 9	00 = no priority 01 = priority 11 = priority-enforced
Reserved for future use	10	
Current status	11 and 12	00 = queue OFF 01 = queue ON 11 = queue ON-exclusive
Reserved for future use	13 through 15	

When SVC13 code 1 option X'C0' (options X'80' and X'40' combined) is executed, information on the status of the specified APU followed by information on the status of the queue to which the APU is assigned is returned to the user's buffer. The first part (APU status) format is presented in Figures 10-4 and 10-5 and Tables 10-2 and 10-3. The second part (queue status) starts at byte offset 36(24), and its format is presented in Figures 10-6 and 10-7 and Table 10-4 where byte addresses of the Figure 10-4 are incremented by 36(24).

10.4 SVC13 CODE 2: AUXILIARY PROCESSING UNIT (APU) MAPPING FUNCTIONS

SVC13 code 2 allows a task to perform mapping functions on a specified APU queue, provided the task has the mapping rights to the specified queue. A task is granted mapping rights to a queue only if:

- the requesting task has been link-edited with the APMAPPING option, and
- no other task has been granted mapping rights to that queue. Operator commands for APU mapping are not accepted if a task already has these mapping rights.

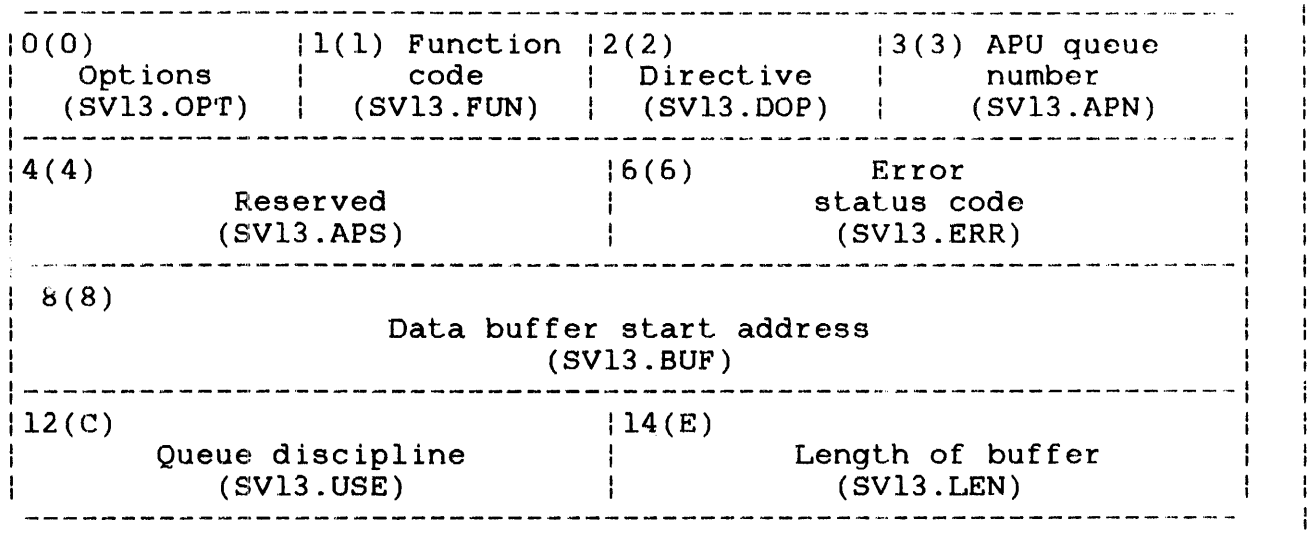
Once a task has been granted mapping rights to an APU, the task can:

- Mark the queue on or ON-exclusive to only one task.
- Map an LPU to the queue.
- Map to queue 0 all LPUs that are mapped to the specified queue.
- Set new queue discipline.
- Mark the queue OFF.

NOTES

1. Queue 0 is always marked on. Marking it ON-exclusive or OFF-exclusive is prohibited.
2. Marking a queue ON-exclusive is disallowed if more than one APU is assigned to the queue.

Figure 10-8 shows the parameter block format and coding for SVC13 code 2.




```
            SVC   13, parblk
            .
            .
            .
            ALIGN 4
parblk      DC    X'option(s)'
            DB    2
            DC    'LPU number' or 0
            DC    'APU queue number'
            DC    H'0'
            DS    2
            DC    A(BUFFER)
            DC    H'queue discipline'
            DC    H'number of bytes in buffer'
```

Figure 10-8 SVC13 Code 2, Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of the fields in this parameter block follows.

Fields:

Options (SV13.OPT) — is a 1-byte field containing a hexadecimal number specifying one or more of the mapping functions.

Figure 10-9 shows the APU mapping option field format. See Table 10-5 for the available options for this field. If more than one APU mapping option is specified, the options are executed in a left-to-right order.

Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 2 to indicate SVC13 code 2.
Directive option (SV13.DOP)	is a 1-byte field that contains the LPU number for option X'10'. All other options ignore this field.
APU queue number (SV13.APN)	is a 1-byte field that must contain the number of the queue to which this SVC is directed.
Reserved (SV13.APS)	is an unused 2-byte field that should be initialized to zero.
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC13 code 1. The first byte of this field indicates the bit position (0-7) of the SVC13 code 2 option being executed when the error occurred. The second byte contains one of the SVC13 error status codes (see Table 10-10). If no error occurs, both bytes contain 0.
Data buffer start address (SV13.BUF)	is a 4-byte field that specifies the address of the buffer containing the name of the task to be granted exclusive access to the specified queue. The task name specified in this buffer must be eight bytes long and left-justified with trailing blanks. If the entire buffer is filled with blanks, the task issuing the SVC is granted exclusive access to the specified queue. This field applies to option X'40' only; all other options ignore this field.
Queue discipline (SV13.USE)	is a 2-byte queue field that contains queue discipline code for option X'40'. All other options ignore this field. The discipline code is one of the following decimal numbers: • 0 indicates no-priority • 1 indicates priority • 3 indicates priority-enforced
Length of buffer (SV13.LEN)	is a 2-byte field indicating the length of the data buffer containing the name of the task to be granted exclusive access to the specified queue. This field applies to option X'40' only; all other options ignore this field.

Gain mapping rights	Mark queue ON-exclusive	Mark queue on	Map LPU to queue	Map all queue LPUs to queue 0	Set queue discipline	Mark queue off	Release mapping rights
Bits: 0	1	2	3	4	5	6	7

Figure 10-9 SVC13 APU Mapping Options Field (SV13.OPT)

TABLE 10-5 SVC13 CODE 2: APU MAPPING OPTIONS FIELD (SVC13.OPT) BIT DEFINITION

APU MAPPING OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Gain mapping rights	0	X'80'	Gain mapping rights to the specified APU queue for the task.	Task linked with APMAPPING task option. No other task has mapping rights to the specified queue.
Mark queue ON-exclusive	1	X'40'	Mark specified queue available for scheduling only the task with the name in the buffer starting at address SV13.BUF (see below).	Other than queue 0 specified. No more than one APU assigned to the queue.
Mark queue on	2	X'20'	Mark specified queue available for scheduling arbitrary tasks.	None
Map LPU to queue	3	X'10'	Map LPUn to queue. n is indicated in SV13.DOP.	None
Map all queue LPUs to queue 0	4	X'08'	Map to queue 0 all LPUs mapped to specified queue.	None

TABLE 10-5 SVC13 CODE 2: APU MAPPING OPTIONS FIELD (SVC13.OPT) BIT DEFINITION (Continued)

APU MAPPING OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Set queue discipline	5	X'04'	Set discipline for specified queue. Discipline code is indicated in SVC13.USE.	None
Mark queue off	6	X'02'	Mark specified queue unavailable for task scheduling.	Other than queue 0 specified.
Release mapping rights	7	X'01'	Give up mapping rights to the specified APU queue for the task.	None

10.5 SVC13 CODE 3: AUXILIARY PROCESSING UNIT (APU) CONTROL

SVC13 code 3 allows a task to perform control functions on a specified APU provided the task has obtained the control rights to the specified APU. OS/32 grants APU control rights to a requesting task only if:

- the task has been link-edited with the APCONTROL option, and
- no other task has been granted control privileges to the specified APU. Operator commands for APU control are not accepted if a task already has these control rights.

SVC13 code 3 gives a task the ability to:

- Initialize an APU (perform a power up link check), enabling it for execution.
- Send a directive to control APU task execution.
- Stop the APU and preempt the currently executing task.
- Select the next task to execute on the APU.
- Assign the APU to an APU queue.
- Disable an APU for on-line maintenance.

Figure 10-10 shows the parameter block format and coding for SVC13 code 3.

0(0) APU control options (SV13.OPT)	1(1) Function code (SV13.FUN)	2(2) Directive option (SV13.DOP)	3(3) APU number (SV13.APN)
4(4) APU hardware status (SV13.APS)		6(6) Error status code (SV13.ERR)	
8(8) Data buffer start address (SV13.BUF)			
12(C) APU queue number (SV13.USE)		14(E) Length of buffer (SV13.LEN)	

```
          SVC   13,parblk
            .
            .
            .
          ALIGN 4
parblk    DB    X'option'
          DB    3
          DB    X'directive option'
          DB    'APU number'
          DS    1
          DS    1 or DB 'byte sent for link check'
          DS    2
          DC    A(BUFFER)
          DC    H'APU queue number'
          DC    H'length of buffer'
```

Figure 10-10 SVC13 Code 3, Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary-aligned, and located in a task-writable segment. A general description of each field in the parameter block follows.

Fields:

APU control options (SV13.OPT)	is a 1-byte field specifying a hexadecimal number indicating the APU control option to be executed. Figure 10-11 shows the APU control option field format. See Table 10-6 for the available options for this field. If more than one APU control option is specified, the options are executed in a left-to-right order.
Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 3 to indicate SVC13 code 3.
Directive option (SV13.DOP)	is a 1-byte field specifying a hexadecimal command code to be sent to the specified APU (see Table 10-7). This field is used only if X'08' was specified in the APU control options field. The directive option field is ignored for all other APU control options.
APU number (SV13.APN)	is a 1-byte field that identifies the specified APU.
APU hardware status (SV13.APS)	is a 2-byte field that contains one of the following: • If option X'08' is specified and any command other than LINK CHECK (X'80') is specified in the SV13.DOP field, this field receives the APU response status returned after execution of the specified command. • If option X'08' is specified and the LINK CHECK command (X'80') is specified in the directive option (SV13.DOP) field, the right-most byte of the halfword defines a data pattern (determined by the user), which is sent to the APU. The APU complements the byte and sends it back to the left byte of the field.
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC13 code 3. The first byte of this field indicates the bit position of the option being executed when the error occurred. The second byte of this field contains one of the SVC13 error status codes. See Table 10-10 for a list of the SVC13 error codes.

Data buffer start address (SV13.LEN)	is a 4-byte field that specifies the address of a buffer containing the name of the task on the APU ready queue that is to be selected as the next task to be executed. This task must be an existing member of the queue. This field applies to option X'10' only and is ignored for all other APU control options.
APU queue number (SV13.USE)	is a 2-byte reserved field that specifies the number of the queue to which the APU is being assigned. This field applies to option X'04' only and is ignored for all other APU control options.
Length of buffer (SV13.LEN)	is a 2-byte field specifying a decimal number (8 or greater) indicating the length of the data buffer specified by the SV13.BUF field. This field applies to option X'10' only and is ignored for all other APU control options.

Gain control right	Enable APU	Stop and reschedule APU taske	Select next task	Send APU command	Assign to queue	Disable APU	Release control rights
Bits: 0	1	2	3	4	5	6	7

Figure 10-11 SVC13 APU Control Options Field (SV13.OPT)

TABLE 10-6 SVC13 CODE 3, APU CONTROL OPTIONS FIELD (SV13.OPT) BIT DEFINITIONS

APU CONTROL PRIVILEGE OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Gain control rights	0	X'80'	Task gains control rights to the specified APU.	Task link-edited with APCONTROL task option. No other task has control rights to specified APU.

TABLE 10-6 SVC13 CODE 3 APU CONTROL OPTIONS FIELD (SV13.OPT) BIT DEFINITIONS (CONTINUED)

APU CONTROL PRIVILEGE OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Enable APU	1	X'40'	Initializes specified APU by performing a power up link check. After power up Link check, APU is in an ENABLED state and idle.	APU must be in DISABLED state. APU must be self-initialized. APU initailizes itself when powered up.
Stop and reschedule APU task	2	X'20'	Stops execution of the current task on the specified APU, saves the task context, and returns the task to the queue according to queue discipline and task priority. APU becomes idle.	APU must be in ENABLED state. APU must have a currently executing task.
Select next task	3	X'10'	Select the task specified in the buffer from the APU queue as the next task for the APU to run. Tasks appearing in the APU ready queue after this task executes in order.	APU must be in ENABLED state. The queue must be no-priority with a single APU assigned.
Send APU Command	4	X'08'	Send the APU command specified in the SVC directive option field (SV13.DOP) to the specified APU.	See Table 10-7.

APU CONTROL PRIVILEGE OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Assign to queue	5	X'04'	Stop execution of current task if any, save the task context and return the task to the queue according to the queue discipline and task priority. Assign the APU to the queue.	None
Disable APU	6	X'02'	Disable the specified APU.	APU must be in ENABLED state.
Release control rights	7	X'01'	Task gives up control rights to the specified APU.	None

TABLE 10-7 SVC13 CODE 3, APU COMMANDS (SV13.DOP)

HEX CODE	MEANING	PREREQUISITES
X'01'	Start APU for task execution. APU enters running state.	APU must be idle.
X'02'	Execute single instruction.	APU must be idle. Reserved for diagnostic use.
X'04'	Transfer current task to CPU Not recommended for user-written tasks. See SVC6 example in Chapter 6.	APU must be idle. APU must have a current task.
X'07'	Start APU for nontask execution loads and starts APU using power fail image.	APU must be idle. Reserved for diagnostic use.

TABLE 10-7 SVC13 CODE 3 APU COMMANDS (SV13.DOP) (Continued)

HEX CODE	MEANING	PREREQUISITES
X'08'	Store power fail image.	APU must be idle. Reserved for diagnostic use.
X'0B'	Stop APU if task state saves context of currently executing task and stops APU (APU enters idle state.)	APU must be running in task state (program status word (PSW) bit 15=0). APU must have a current task.
X'80'	Real-time support module (RTSM) link check - APU sends back complement to data byte received.	APU must be idle. Reserved for diagnostic use.
X'83'	Reschedules task on APU; APU reschedules the current task to the rear of the APU queue.	APU must have a current task. APU must be idle.
	NOTE The X'83' command temporarily disrupts task order in a priority or priority-enforced queue until the operating system restores it based on APU signals. A better method of rescheduling a task on an APU is to issue an SVC13 code 3 with the X'20' APU control option specified.	
X'85'	Stop APU and save power fail image - APU saves context of currently executing task and stops APU (APU enters idle state).	Use only after attempt to stop APU with X'0B' has failed.
X'86'	NO-OP - APU sends its status only.	None
X'89'	Fetch APU error code.	Reserved for diagnostic use.
X'8A'	Checkpoint task state saves context of current task. Task continues execution if previously running or remains idle if previously idle.	APU must have a current task.

NOTE

An undefined command code in the SV13.DOP field is sent to the APU without OS/32 intervention. The APU identifies the code sent as an unrecognizable command code or as a sequence error. For more information on the SVC13 APU commands, see the Model 3200MPS System Instruction Set Reference Manual.

10.6 SVC13 AUXILIARY PROCESSING UNIT (APU) HARDWARE STATUS FIELD (SV13.APS)

After execution of SVC13 code 1 option X'80' or SVC13 code 3 option X'08', the status of the APU hardware is saved in the APU status (SV13.APS) field of the parameter block. This field consists of two bytes, a response byte and an error code byte, representing the response and error fields of the program status word (PSW). The hardware status is returned to this field in the format shown in Figure 10-12. See Table 10-8 for the response byte bit definitions. Error codes returned to the error code byte are listed in Table 10-9. See the Model 3200MPS System Instruction Set Reference Manual for more information.

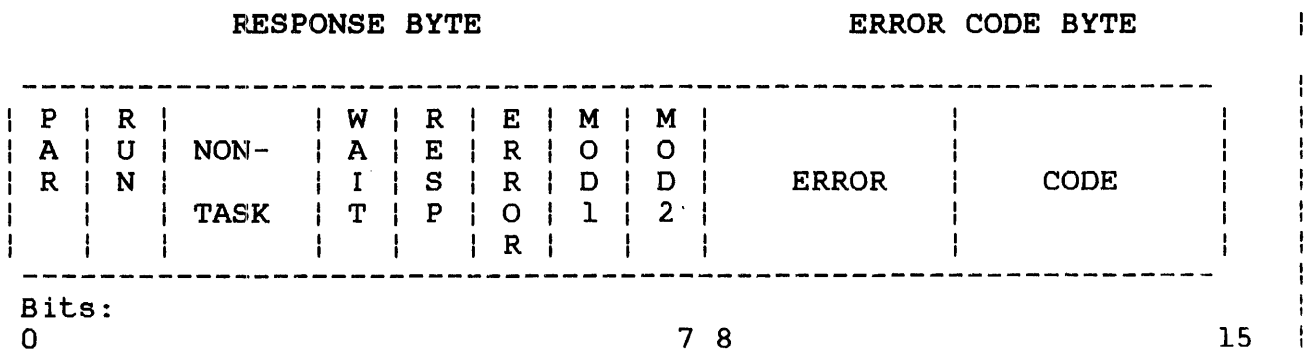


Figure 10-12 APU Hardware Response Byte (SV13.APS)

TABLE 10-8 APU HARDWARE RESPONSE BYTE/BIT DEFINITIONS

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
0	PAR	1= ensures that the response byte has an odd parity. 0= odd number of bits have been set for the remainder of the byte.
1	RUN	1= APU is running. 0= APU is idle.
2	NONTASK	1= current PSW bit 15 is set, indicating no context save area is available. 0= APU executing a task; the current PSW bit 15 is not set, indicating that the current task's context save area is ready to receive the processor task state.
3	WAIT	1= current PSW bit 16 is set, indicating APU is in a wait state or APU is working in an internal service state (e.g., scheduling a task). 0= current PSW bit 16 is not set, indicating the APU is executing instructions.
4	RESP	1= APU is responding to a command from the CPU. 0= APU is generating a signal indicating a change in APU function.
5	ERROR	1= APU detects an error condition that causes the APU to stop; the error condition is indicated by the setting of the error code bit (see Table 10-9). 0= no error condition is detected by APU.
6,7	MOD1 MOD2	Bit definitions for MOD1 and MOD2 depend on the definitions for RESP and ERROR, as follows: RESP=0, ERROR=0 00 = undefined 01 = APU entering queue wait state 10 = task rescheduled to APU ready queue 11 = task rescheduled to CPU

TABLE 10-8 APU HARDWARE RESPONSE BYTE/BIT DEFINITIONS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
6,7 (cont'd)		RESP=0, ERROR=1 00 = general error status 01 = error occurred while APU in queue wait state 10 = error occurred while locking queue 11 = undefined RESP=1, ERROR=0 00 = general response status 01 = task is waiting on APU queue 10 = APU attempting to lock a queue 11 = command sequence error; command was ignored RESP=1, ERROR=1 00 = error as a result of command 01 = response, error in queue wait 10 = response, error in queue lock 11 = error and command sequence error

NOTE

See the Model 3200MPS System Instruction Set Reference Manual for more information on the nontask and wait states that reflect the PSW bit definitions.

TABLE 10-9 ERROR CODES FOR ERROR CODE BYTE OF APU HARDWARE STATUS FIELD (SV13.APS)

ERROR CODE	MEANING
X'80'	No error
X'00'	No response from the APU
X'01'	APUID DEVICE FALSE SYNC
X'02'	ZERO APUID RETURNED BY RTSM
X'83'	CAN'T FETCH WORD @ X'C4' - ECC

TABLE 10-9 ERROR CODES FOR ERROR CODE BYTE OF APU HARDWARE STATUS FIELD (SV13.APS) (Continued)

ERROR CODE	MEANING
X'04'	APUID > MAX APU @ X'C7'
X'85'	BAD A(APB_DIR) - ECC/ZERO/ALIGN
X'86'	BAD A(APB) - ECC/ZERO/ALIGN
X'07'	BAD APB (FLAGS:APB#) WORD - ECC
X'08'	WRONG APB# IN APB
X'89'	APB PASSBACK
X'8A'	UNRECOGNIZED COMMAND
X'0B'	BAD APB A(CTCB) - ECC/ZERO/ALIGN
X'8C'	BAD A (APU TCB QUEUE) - ECC/ZERO/ALIGN
X'0D'	QUEUE LOCK TIMEOUT
X'0E'	EXECUTION SUSPENDED (TRAP PSW WAIT)
X'8P'	BAD SSTD - ECC
X'10'	CAN'T LOAD TASK CONTEXT
X'91'	CAN'T STORE TASK CONTEXT
X'92'	CAN'T LOAD PWR FAIL IMAGE
X'13'	CAN'T STORE POWER FAIL IMAGE
X'94'	CAN'T LOAD PSTD - ECC
X'15'	BAD APB PFAIL PTR - ECC/ZERO
X'16'	BAD APB MMF NEW PSW - PCC/ZERO
X'97'	BAD CTCB CTX PTR - ECC/ZERO/ALIGN
X'98'	BAD APB TCB CNT WORK - ECC
X'19'	BAD A(APU FRONT TCB) - ECC/ZERO/ALIGN
X'1A'	FRONT TCB PTR< TCB CNT DISAGREE
X'9B'	QUEUE TCB CNT UNDERFLOW
X'1C'	BAD APB A(CPU QUEUE) - ECC/ZERO/ALIGN
X'9D'	BAD TCB QHPTR - ECC/ZERO/ALIGN
X'9E'	INCORRECT TCB QUEUE HEAD PTR
X'1F'	BAD TCB BPTR - ECC/ZERO/ALIGN
X'20'	BAD BACK TCB FPTR - ECC/ZERO/ALIGN
X'A1'	BACK TCB FPTR NOT TO FRONT TCB
X'A2'	BAD FRONT TCB FPTR - ECC/ZERO/ALIGN
X'23'	BAD FWD TCB BPTR - ECC/ZERO/ALIGN
X'A4'	FWD TCB BPTR NOT TO FRONT TCB
X'25'	INCONSISTENT FRONT TCB FPTR & BPTR
X'26'	BAD FRONT TCB PTR - ECC/ZERO/ALIGN
X'A7'	BAD BACK TCB FPTR - ECC/ZERO/ALIGN

TABLE 10-9 ERROR CODES FOR ERROR CODE BYTE OF APU HARDWARE STATUS FIELD (SV13.APS) (Continued)

ERROR CODE	MEANING
X'A8'	BACK TCB'S FPTR NOT TO FRONT TCB
X'29'	TCB QUEUE OVERFLOW (CPU OR APU)
X'2A'	BAD MSH TIME ACCUMULATOR - ECC
X'AB'	BAD LSH TIME ACCUMULATOR - ECC
X'2C'	BAD TCB START TIME WORD - ECC
X'AD'	CAN'T READ RTSM CLOCK DATA
X'AE'	TCB ELAPSED TIME OVERFLOW
X'2F'	TCB "PENDING" FLAGS SET ON QUEUE OR CTCB
X'B0'	BAD "PENDING" FLAGS WORD - ECC
X'31'	INTERRUPT FROM RTSM XMTR
X'32'	CAN'T LOAD PFAIL PSTD - ECC
X'B3'	CAN'T LOAD PFAIL SSTD - ECC
X'B4'	BAD APB MSH TASK TIME ACC - ECC
X'B5'	BAD APB LSH TASK TIME ACC - ECC
X'B6'	WRONG LAST APUID IN CTCB

10.7 SVC13 ERROR STATUS CODE FIELD (SV13.ERR)

When execution of SVC13 is completed, the execution status is returned to the error status code (SV13.ERR) field of the parameter block. If no error occurs, a value of 0 is stored in this field. If SVC13 code 1, 2 or 3 is issued and an error occurs, the first byte of this field contains the bit position of the option that caused the error.

Table 10-10 lists the SVC13 error status codes and their applicable function codes.

TABLE 10-10 SVC13 ERROR STATUS CODES (SV13.ERR)

STATUS CODE	APPLICABLE FUNCTION CODES	MEANING
0	All	No errors occurred.
1	All	The specified data buffer does not begin on a fullword boundary.
2	0,1	The specified data buffer is not located in a writable segment of the task.
3	All	Insufficient space was available in the supplied data buffer. For functions 0 and 1, any data that does not fit in the available space is lost.
4	2,3	Task establishment options prohibit the task from gaining mapping or control rights.
5	2,3	Task has not been granted the rights to perform the attempted mapping or control function.
6	1 option X'80',3	The APU number specified is greater than the maximum allowed.
7	2 option X'10'	The LPU number specified is greater than the maximum allowed.
8	1,2,3	An invalid option was specified for this function.
9	2,3	The requested privilege is currently held by another task and cannot be granted.
10	2	The specified queue cannot be marked ON-exclusive from an ON state.
11	3	The function requested cannot be completed because the specified APU is in a DISABLED state.
12	1 option X'40', 2, 3 option X'04'	Access to the APU queue could not be obtained; SVC13 request aborted.

TABLE 10-10 SVC13 ERROR STATUS CODES (SV13.ERR) (Continued)

STATUS CODE	APPLICABLE FUNCTION CODES	MEANING
13	2 option X'10'	The task has not been granted mapping rights over the APU queue to which the specified APU is currently mapped.
14	3 option X'40'	Cannot enable APU unless in a DISABLED state.
15	3 option X'40'	APU could not pass power-up link check sequence. APU left in disabled state.
16	3 option X'02'	Cannot disable APU unless in an ENABLED state.
17	2 option X'40'	The APU could not be marked ON-exclusive because the specified task could not be found in the system.
18	3 option X'08'	Error encountered in transmission of the specified control command.
19	3 option X'10'	The preemptive task could not be found on the specified APU ready queue. The APU queue is unchanged.
20	3 option X'10'	The preemptive task is not on a no-priority queue to which a single APU is assigned. The queue is unchanged.
21	1 option X'40',2 3 option X'04'	The queue number specified is greater than the maximum allowed.
22	2	The specified option is not applicable to queue 0 to which the mapping function is directed.
23	2 option X'04'	The queue discipline specified is undefined.
24	2 option X'40'	The queue cannot be marked ON-exclusive because more than one APU is currently assigned to it.
25	3	No response from APU.

10.8 TYPICAL OPTION CODING SEQUENCES FOR SVC13 CODE 2 AND CODE 3

The options field (SV13.OPT) in the SVC13 parameter blocks for codes 2 and 3 allows the user to issue one call to execute multiple APU mapping or control functions. Multiple options are executed from left-to-right. Care must be taken when selecting the sequence of options to perform a designated mapping or control function. The following sections demonstrate specific option coding sequences that would be used by a typical APU control task in a Model 3200MPS System.

10.8.1 Auxiliary Processing Unit (APU) Initialization and Start-Up

Before a task can run on an APU, the APU must be initialized and started for task execution. This is accomplished through an SVC13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task.
X'40'	Enable (initialize) APU.
X'08'	Send APU start directive (X'01') specified in SV13.DOP field.
X'01'	Release control rights.

At system start-up, the APU is assigned to a queue with the same number as that of the APU. The queue is defined as no-priority.

10.8.2 Auxiliary Processing Unit (APU) Queue Mark On

Before a task can be scheduled to an APU queue, the queue must be activated and the task's LPU mapped to it. This is accomplished through an SVC13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task.
X'20'	Mark APU queue on.
X'10'	Map LPU to the queue.
X'01'	Release mapping rights.

If the APU is to be marked on exclusively for one task, the following option coding sequence is used for SVC13 code 2:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task.
X'40'	Mark APU queue ON-exclusive.
X'10'	Map LPU to the queue.
X'01'	Release mapping rights.

10.8.3 Setting Auxiliary Processing Unit (APU) Queue Discipline

To change the existing queue discipline definition, use SVC13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task.
X'40'	Set queue discipline.
X'01'	Release mapping rights.

10.8.4 Assigning Auxiliary Processing Unit (APU) to a Queue

To change the existing APU assignment, use SVC13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task.
X'04'	Assign APU to queue.
X'01'	Release control rights.

10.8.5 Task Scheduling on the Auxiliary Processing Unit (APU)

Normally, tasks are executed according to queue discipline and task priority. Consequently, it is possible to control the order of task execution on an APU priority or priority-enforced queue by changing task priorities through SVC6 mechanisms.

When minimal task context switch time is desired, a no-priority discipline can be used. Controlling the order of task execution on the APU no-priority queue may be accomplished by preempting (stopping) the current active task, rescheduling the task to the rear of the queue, and

- selecting the next task from the front of the queue for execution or
- selecting a task (by name) on the queue for execution.

To schedule the current task to the rear of the queue, use SVC13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task.
X'20'	Stop APU task execution and reschedule current task to rear of the queue.
X'08'	Send APU start directive (X'01') specified in SV13.DOP field to select the task at the front of the APU ready queue for execution.
X'01'	Release control rights.

To preempt the next ready task, thereby explicitly selecting the next task to be run, use SVC13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task.
X'20'	Stop APU task execution and reschedule current task to rear of the queue.
X'10'	Select designated task on the queue (changing the pointer in the task queue head to point to the task).
X'08'	Send APU start directive (X'01') specified in SV13.DOP field to start task.
X'01'	Release control rights.

10.8.6 Auxiliary Processing Unit (APU) Queue Mark Off

An APU queue can be marked off with or without remapping respective LPUs. To mark off an APU queue without remapping, use SVC13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task.
X'02'	Mark queue off.
X'01'	Release mapping rights.

To mark an APU queue off and remap respective LPUs to queue 0, use SVC13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task.
X'08'	Map queue's LPUs to queue 0.
X'02'	Mark queue off.
X'01'	Release mapping rights.

CHAPTER 11
USER SUPERVISOR CALL 14 (SVC14)

11.1 SVC14: USER

SVC14 gives a user-written task a means of accepting an SVC from a part of itself; e.g., a subroutine or other module.

Format:

```
SVC     14,A(X2) or          RX1,RX2 FORMATS
SVC     14,A(FX2,SX2)        RX3 FORMAT
```

The address field of SVC14 is not interpreted by OS/32 but can be defined by the task. Normally, it might be used to point to a parameter block.

If the user SVC trap enable bit in the current task status word (TSW) is enabled, SVC14 is enabled; otherwise, SVC14 is considered an illegal SVC.

When SVC14 is executed, the operating system stores the effective program address of the SVC14 second argument into the SVC14 address pointer location in the task user-dedicated location (UDL). A TSW swap then occurs, using the SVC14 TSW swap area in the UDL. The interpretation of this SVC is then left to the user. The effective program address is calculated as for an RX1, RX2 or RX3 instruction. This facility permits the user to build a virtual executive task (e-task) within a single task environment.

OS/32 AIDS, the OS/32 debugging utility, makes use of SVC14; consequently, a task should not use SVC14 while the OS/32 AIDS software is in operation.

See the OS/32 Application Level Programmer Reference Manual for more information on enabling and handling SVC14 task traps.

CHAPTER 12
DATA COMMUNICATIONS DEVICE-DEPENDENT INPUT/OUTPUT (I/O) SUPERVISOR CALL 15 (SVC15)

12.1 SVC15: DATA COMMUNICATIONS DEVICE-DEPENDENT INPUT/OUTPUT (I/O)

SVC15 allows a user-written task to access data communications devices at the device-dependent level. See the OS/32 Basic Data Communications Reference Manual for more information.

INDEX

T

U

PERKIN-ELMER

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, *etc.*

1. Publication number _____

2. Title of publication _____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

   _____

   _____

   _____

4. Was the publication easy to understand? If no, why not?

   _____

5. Were illustrations adequate? _____

   _____

   _____

6. What additions or deletions would you suggest? _____

   _____

   _____

7. Other comments: _____

   _____

   _____

From _____ Date _____

Position/Title _____

Company _____

Address _____

_____

_____

STAPLE STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Data Systems Group
106 Apple Street
Tinton Falls, NJ 07724

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

FOLD FOLD

STAPLE STAPLE

643