

PERKIN-ELMER

OS/32 SUPERVISOR CALL (SVC)

Reference Manual

48-038 F00 R01

The information in this document is subject to change without notice and should not be construed as a commitment by the Perkin-Elmer Corporation. The Perkin-Elmer Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license, and it can be used or copied only in a manner permitted by that license. Any copy of the described software must include the Perkin-Elmer copyright notice. Title to and ownership of the described software and any copies thereof shall remain in The Perkin-Elmer Corporation.

The Perkin-Elmer Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Perkin-Elmer.

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

© 1981, 1982, 1983 by The Perkin-Elmer Corporation

Printed in the United States of America

TABLE OF CONTENTS

PREFACE		xi	
 CHAPTERS			
1	SUPERVISOR CALLS (SVCs)		
1.1	INTRODUCTION	1-1	
1.1.1	The Supervisor Call (SVC) Parameter Block	1-3	
1.2	SUPERVISOR CALL (SVC) ERRORS	1-6	
1.2.1	Supervisor Call (SVC) Error Messages	1-6	
1.2.2	Supervisor Call (SVC) Status Codes	1-8	
2	INPUT/OUTPUT (I/O) REQUEST SUPERVISOR CALL 1 (SVC 1)		
2.1	INTRODUCTION	2-1	
2.2	SUPERVISOR CALL 1 (SVC 1)	2-3	
2.2.1	Data Transfer Requests	2-5	
2.2.1.1	Test and Set	2-8	
2.2.1.2	Input/Output (I/O) Proceed	2-14	
2.2.1.3	Queuing Input/Output (I/O) Requests	2-14	
2.2.1.4	Conditional Proceed	2-15	
2.2.1.5	Unconditional Proceed	2-15	
2.2.1.6	Wait Input/Output (I/O)	2-16	
2.2.1.7	Wait Only	2-16	
2.2.2	Command Function Requests	2-16	
2.2.2.1	Halt Input/Output (I/O)	2-18	
2.2.3	Logical Unit (lu)	2-18	
2.2.4	Device Independent Status	2-18	
2.2.5	Device Dependent Status	2-20	
2.2.6	Buffer Start/Buffer End Address	2-20	
2.2.7	Extended Options	2-21	
2.2.7.1	Nonmagnetic Tape Devices	2-21	
2.2.7.2	Magnetic Tape Devices	2-22	
2.2.7.3	Device Dependent Status Codes for Magnetic Tape Operations	2-27	
2.3	GAPLESS INPUT/OUTPUT (I/O) OPERATIONS	2-29	
2.3.1	Gapless Mode SVC 1 Parameter Block Format	2-29	
2.3.2	Standard Function Code Format - Gapless Mode	2-32	
2.3.3	Logical Unit (lu)	2-33	

CHAPTERS (Continued)

	2.3.4	Device Independent Status Codes	2-33
	2.3.5	Device Dependent Status Codes	2-35
	2.3.6	Buffer Queues	2-36
	2.3.6.1	Using the Buffer Queue	2-37
	2.3.6.2	Trap-Causing Events Resulting from Gapless Input/Output (I/O) Operations	2-38
	2.3.7	Buffer Length	2-38
	2.3.8	Length of Last Buffer	2-38
	2.3.9	Extended Options Field	2-39
3	GENERAL SERVICE FUNCTIONS SUPERVISOR CALL 2 (SVC 2)		
	3.1	INTRODUCTION	3-1
	3.2	SVC 2 CODE 1: PAUSE	3-5
	3.3	SVC 2 CODE 2: GET STORAGE	3-7
	3.3.1	SVC 2 Code 2, Option X'00'	3-9
	3.3.2	SVC 2 Code 2, Option X'80'	3-11
	3.4	SVC 2 CODE 3: RELEASE STORAGE	3-13
	3.5	SVC 2 CODE 4: SET STATUS	3-16
	3.5.1	SVC 2 Code 4, Option X'00'	3-18
	3.5.2	SVC 2 Code 4, Option X'80'	3-19
	3.6	SVC 2 CODE 5: FETCH POINTER	3-20
	3.7	SVC 2 CODE 6: CONVERT BINARY NUMBER TO ASCII HEXADECIMAL OR ASCII DECIMAL	3-23
	3.7.1	SVC 2 Code 6, Option X'00'+n	3-25
	3.7.2	SVC 2 Code 6, Option X'40'+n	3-25
	3.7.3	SVC 2 Code 6, Option X'80'+n	3-26
	3.7.4	SVC 2 Code 6, Option X'C0'+n	3-26
	3.8	SVC 2 CODE 7: LOG MESSAGE	3-27
	3.8.1	SVC 2 Code 7, Option X'00'	3-29
	3.8.2	SVC 2 Code 7, Option X'20'	3-30
	3.8.3	SVC 2 Code 7, Option X'40'	3-30
	3.8.4	SVC 2 Code 7, Option X'60'	3-30
	3.8.5	SVC 2 Code 7, Option X'80'	3-30
	3.8.6	SVC 2 Code 7, Option X'A0'	3-32
	3.8.7	SVC 2 Code 7, Option X'C0'	3-32
	3.8.8	SVC 2 Code 7, Option X'E0'	3-32
	3.9	SVC 2 CODE 8: INTERROGATE CLOCK	3-33
	3.9.1	SVC 2 Code 8, Option X'00'	3-34
	3.9.2	SVC 2 Code 8, Option X'80'	3-35
	3.9.3	SVC 2 Code 8, Option X'40'	3-36
	3.9.4	SVC 2 Code 8, Option X'C0'	3-37

CHAPTERS (Continued)

3.10	SVC 2 CODE 9: FETCH DATE	3-38
3.11	SVC 2 CODE 10: TIME OF DAY WAIT	3-41
3.12	SVC 2 CODE 11: INTERVAL WAIT	3-44
3.13	SVC 2 CODE 15: CONVERT ASCII HEXADECIMAL OR ASCII DECIMAL TO BINARY	3-46
3.13.1	SVC 2 Code 15, Option X'00'	3-47
3.13.2	SVC 2 Code 15, Option X'40'	3-49
3.13.3	SVC 2 Code 15, Option X'80'	3-50
3.13.4	SVC 2 Code 15, Option X'C0'	3-52
3.14	SVC 2 CODE 16: PACK FILE DESCRIPTOR	3-53
3.14.1	SVC 2 Code 16, Option X'00'	3-62
3.14.2	SVC 2 Code 16, Option X'40'	3-63
3.14.3	SVC 2 Code 16, Option X'10'	3-64
3.14.4	SVC 2 Code 16, Option X'50'	3-65
3.14.5	SVC 2 Code 16, Option X'20'	3-66
3.14.6	SVC 2 Code 16, Option X'60'	3-67
3.14.7	SVC 2 Code 16, Option X'80'	3-68
3.14.8	SVC 2 Code 16, Option X'C0'	3-71
3.14.9	SVC 2 Code 16 Options for Privileged Tasks	3-72
3.15	SVC 2 CODE 17: SCAN MNEMONIC TABLE	3-74
3.15.1	Building a Mnemonic Table	3-76
3.15.2	Executing SVC 2 Code 17	3-76
3.16	SVC 2 CODE 18: MOVE ASCII CHARACTERS	3-82
3.16.1	SVC 2 Code 18, Option X'00'+n	3-84
3.16.2	SVC 2 Code 18, Option X'80'+n	3-86
3.17	SVC 2 CODE 19: PEEK	3-91
3.17.1	Parameter Block for Option X'00'	3-91
3.17.2	Parameter Block for Option X'01'	3-97
3.17.3	Parameter Block for Option X'02'	3-103
3.17.4	Parameter Block for Option X'03'	3-105
3.17.5	Parameter Block for Option X'04'	3-111
3.18	SVC 2 CODE 20: EXPAND ALLOCATION	3-115
3.19	SVC 2 CODE 21: CONTRACT ALLOCATION	3-117
3.20	SVC 2 CODE 23: TIMER MANAGEMENT	3-118
3.20.1	SVC 2 Code 23 Parameter Block for Option X'00'	3-118
3.20.2	SVC 2 Code 23 Parameter Block for Option X'80'	3-121
3.20.3	SVC 2 Code 23 Parameter Block for Option X'40'	3-124
3.20.4	SVC 2 Code 23 Parameter Block for Option X'20'	3-128
3.20.5	SVC 2 Code 23 Parameter Block for Option X'10'	3-130
3.21	SVC 2 CODE 24: SET ACCOUNTING INFORMATION	3-133

CHAPTERS (Continued)

	3.22	SVC 2 CODE 25: FETCH ACCOUNTING INFORMATION	3-135
	3.23	SVC 2 CODE 29: UNPACK FILE DESCRIPTOR	3-137
4	END OF TASK SUPERVISOR CALL 3 (SVC 3)		
	4.1	INTRODUCTION	4-1
	4.2	SVC 3: END OF TASK	4-2
5	FETCH OVERLAY SUPERVISOR CALL 5 (SVC 5)		
	5.1	INTRODUCTION	5-1
	5.2	SVC 5: FETCH OVERLAY	5-2
6	INTERTASK COMMUNICATIONS SUPERVISOR CALL 6 (SVC 6)		
	6.1	INTRODUCTION	6-1
	6.2	SVC 6: INTERTASK COMMUNICATIONS	6-2
	6.2.1	Function Code (SVC6.FUN)	6-5
	6.2.2	Direction (SFUN.DOM,SFUN.DSM)	6-10
	6.2.3	End Task (SFUN.ECM,SFUN.EDM)	6-10
	6.2.4	Load Task Functions	6-11
	6.2.4.1	Load Task (SFUN.LM)	6-11
	6.2.4.2	Load Task with Extended Load Options (SFUN.LXM)	6-12
	6.2.5	Task Resident (SFUN.HM)	6-16
	6.2.6	Suspend (SFUN.SM)	6-17
	6.2.7	Send Data (SFUN.DB)	6-17
	6.2.7.1	Send Data Message Buffer for Sending Task	6-17
	6.2.7.2	Free Send Data Message Buffers for Receiving Task	6-19
	6.2.7.3	Sample Programs Using SVC 6 Send Data Function	6-21
	6.2.8	Send Message (SFUN.MM)	6-25
	6.2.8.1	Message Buffers	6-26
	6.2.9	Queue Parameter (SFUN.QM)	6-33
	6.2.10	Change Priority (SFUN.PM)	6-33
	6.2.11	Send Logical Unit (SFUN.XSM)	6-34
	6.2.12	Receive Logical Unit (SFUN.XRM)	6-34
	6.2.13	Connect (SFUN.OM)	6-35
	6.2.14	Thaw (SFUN.TM)	6-36
	6.2.15	Sint (SFUN.IM)	6-36
	6.2.16	Freeze (SFUN.FM)	6-37
	6.2.17	Unconnect (SFUN.UM)	6-37
	6.2.18	Assign LPU (SFUN.LPU)	6-37
	6.2.19	Transfer to LPU (SFUN.TL)	6-38
	6.2.20	Transfer to CPU (SFUN.TC)	6-38

CHAPTERS (Continued)

6.2.21	Release (SFUN.RM)	6-39	
6.2.22	Nonresident (SFUN.NM)	6-39	
6.2.23	Rollable (SFUN.RLM)	6-40	
6.2.24	Nonrollable (SFUN.NRM)	6-40	
6.2.25	Start (Bit Positions 29, 30, 31)	6-40	
6.2.26	Start Function for SVC 6 (SFUN.SIM)	6-41	
6.2.27	Start Function with Start Options for SVC 6 (SFUN.SOM)	6-41	
6.2.28	Delay Start Function for SVC 6 (SFUN.SDM)	6-42	
6.2.29	Delay Start Function with Start Options for SVC 6 (SFUN.SDM,SFUN.SOM)	6-42	
6.2.30	Wait Status Field (SVC6.TST)	6-43	
6.2.31	Error Codes (SVC6.STA)	6-44	
7	FILE HANDLING SERVICES SUPERVISOR CALL 7 (SVC 7)		
7.1	INTRODUCTION	7-1	
7.2	SVC 7: FILE HANDLING SERVICES	7-2	
7.2.1	Function Code Field (SVC7.OPT)	7-5	
7.2.1.1	Allocate Function	7-11	
7.2.1.2	Assign Function	7-12	
7.2.1.2.1	Temporary File Allocation and Assignment Function	7-13	
7.2.1.3	Change Access Privileges Function	7-14	
7.2.1.4	Rename Function	7-15	
7.2.1.5	Reprotect Function	7-15	
7.2.1.6	Close Function	7-16	
7.2.1.7	Delete Function	7-16	
7.2.1.8	Checkpoint Function	7-17	
7.2.1.9	Fetch Attributes Function	7-17	
7.2.1.10	Vertical Forms Control (VFC)	7-20	
7.2.1.11	Fetch Time and Date Attributes from Disk Directory	7-21	
7.2.1.12	Fetch Logical Attributer of Open File	7-24	
7.2.1.13	Access Privileges	7-26	
7.2.1.14	Data Communications Access Methods	7-27	
7.2.1.15	File Types	7-28	
7.2.1.16	Read/Write Key Fields (SVC7.RKY/SVC7.WKY)	7-29	
7.2.1.17	File Size Field (SVC7.SIZ)	7-30	
7.2.1.18	SVC 7 Error Codes	7-31	
8	LOAD TASK STATUS WORD (TSW) SUPERVISOR CALL 9 (SVC 9)		
8.1	INTRODUCTION	8-1	
8.2	SVC 9: LOAD TASK STATUS WORD (TSW)	8-2	
8.2.1	Function and Description of the Task Status Word (TSW)	8-3	

CHAPTERS (Continued)

9	OVERLAY LOADING SUPERVISOR CALL 10 (SVC 10)	
	9.1 SUPERVISOR CALL 10 (SVC 10)	9-1
	9.2 MESSAGES	9-1
	10 AUXILIARY PROCESSING UNIT (APU) CONTROL SUPERVISOR CALL 13 (SVC 13)	
	10.1 INTRODUCTION	10-1
	10.2 FETCHING LOGICAL PROCESSOR MAPPING TABLE (LPMT)	10-2
	10.3 FETCHING AUXILIARY PROCESSING UNIT (APU) STATUS INFORMATION	10-2
	10.4 AUXILIARY PROCESSING UNIT (APU) MAPPING FUNCTIONS	10-2
	10.5 AUXILIARY PROCESSING UNIT (APU) CONTROL FUNCTIONS	10-3
	11 USER SUPERVISOR CALL 14 (SVC 14)	
	11.1 SUPERVISOR CALL 14 (SVC 14)	11-1
	12 DATA COMMUNICATIONS DEVICE DEPENDENT INPUT/OUTPUT (I/O) SUPERVISOR CALL (SVC 15)	
	12.1 SUPERVISOR CALL 15 (SVC 15)	12-1

FIGURES

2-1	SVC 1 Parameter Block Format and Coding	2-3
2-2	Function Code Format for Data Transfer Requests	2-5
2-3	Extended Options Fullword Format for Nonmagnetic Tape Devices	2-21
2-4	Extended Options Fullword Format for Magnetic Tape I/O Operations	2-23
2-5	SVC 1 Gapless Mode Parameter Block Format and Coding	2-30
2-6	Function Code Format for Gapless Mode Data Transfer Requests	2-32
	2-7 IN-QUEUE or OUT-QUEUE Structure	2-36

FIGURES (Continued)

3-1	SVC 2 Code 1 Parameter Block Format and Coding	3-5
3-2	SVC 2 Code 2 Parameter Block Format and Coding	3-7
3-3	Task Impure Segment for SVC 2 Code 2, Option X'00'	3-10
3-4	Task Impure Segment for SVC 2 Code 2, Option X'80'	3-12
3-5	SVC 2 Code 3 Parameter Block Format and Coding	3-13
3-6	Task Impure Segment for SVC 2 Code 3	3-15
3-7	SVC 2 Code 4 Parameter Block Format and Coding	3-16
3-8	Program Status Word (PSW)	3-17
3-9	SVC 2 Code 5 Parameter Block Format and Coding	3-20
3-10	SVC 2 Code 6 Parameter Block Format and Coding	3-23
3-11	SVC 2 Code 7 Parameter Block Format and Coding	3-27
3-12	SVC 2 Code 8 Parameter Block Format and Coding	3-33
3-13	SVC 2 Code 9 Parameter Block Format and Coding	3-38
3-14	SVC 2 Code 10 Parameter Block Format and Coding	3-41
3-15	SVC 2 Code 11 Parameter Block Format and Coding	3-44
3-16	SVC 2 Code 15 Parameter Block Format and Coding	3-46
3-17	SVC 2 Code 16 Parameter Block Format and Coding	3-53
3-18	Packed File Descriptor Area	3-57
3-19	SVC 2 Code 17 Parameter Block Format and Coding	3-74
3-20	SVC 2 Code 18 Parameter Block Format and Coding	3-82
3-21	SVC 2 Code 19 Parameter Block Format and Coding for Option X'00'	3-91
3-22	SVC 2 Code 19 Parameter Block Format and Coding for Option X'01'	3-97
3-23	SVC 2 Code 19 Parameter Block Format and Coding for Option X'02'	3-103
3-24	SVC 2 Code 19 Parameter Block Format and Coding for Option X'03'	3-106
3-25	SVC 2 Code 19 Parameter Block Format and Coding for Option X'04'	3-112
3-26	SVC 2 Code 20 Parameter Block Format and Coding	3-115
3-27	SVC 2 Code 21 Parameter Block Format and Coding	3-117
3-28	SVC 2 Code 23 Parameter Block Format and Coding for Option X'00'	3-119
3-29	SVC 2 Code 23 Parameter Block Format and Coding for Option X'80'	3-122
3-30	SVC 2 Code 23 Parameter Block Format and Coding for Option X'40'	3-124
3-31	SVC 2 Code 23 Parameter Block Format and Coding for Option X'20'	3-128
3-32	SVC 2 Code 23 Parameter Block Format and Coding for Option X'10'	3-131
3-33	SVC 2 Code 24 Parameter Block Format and Coding	3-133
3-34	SVC 2 Code 25 Parameter Block Format and Coding	3-135
3-35	Area Receiving Accounting Information	3-136
3-36	SVC 2 Code 29 Parameter Block Format and Coding	3-137
5-1	SVC 5 Parameter Block Format and Coding	5-2

FIGURES (Continued)

6-1	SVC 6 Parameter Block Format and Coding	6-2
6-2	SVC 6 Function Code Field	6-6
6-3	Extended Load Options Field	6-13
6-4	Send Data Message Buffer Format for Calling Task	6-18
6-5	Send Data Message Buffer Format for Directed Task	6-20
6-6	Message Buffer Format for Directed Task	6-26
6-7	Single Buffer Ring	6-27
6-8	Single Buffer Chain	6-28
6-9	Multiple Buffer Ring	6-29
6-10	Multiple Buffer Chain	6-30
6-11	Error Status Field	6-44
7-1	SVC 7 Parameter Block Format and Coding	7-2
7-2	SVC 7 Function Code Field	7-6
7-3	SVC 7 Parameter Block Format and Coding for a Fetch Attributes Function	7-18
7-4	SVC 7 Parameter Block Format and Coding for VFC Function	7-21
7-5	SVC 7 X'FF00', X'FF01', or X'FF02' Parameter Block Format and Coding for Fetch Time and Date Attributes Function	7-22
7-6	SVC 7 X'FF03' Parameter Block Format and Coding for Fetch Time and Date Attributes Function	7-23
7-7	SVC 7 X'FF04' Parameter Block Format and Coding for Fetch Time and Date Attributes Function	7-24
7-8	SVC 7 X'FF0A' Parameter Block Format and Coding for the Fetch Logical Attributes of Open File Function	7-25
8-1	SVC 9 Parameter Block Format and Coding	8-2
8-2	Task Status Word	8-4

TABLES

1-1	OS/32 SUPERVISOR CALLS	1-1
2-1	FUNCTION CODE BIT POSITIONS FOR DATA TRANSFER REQUESTS	2-5
2-2	FUNCTION CODES FOR COMMAND FUNCTION REQUESTS	2-17
2-3	DEVICE INDEPENDENT STATUS CODES	2-19
2-4	DEVICE DEPENDENT STATUS CODES	2-20
2-5	SVC 1 EXTENDED OPTIONS FOR LOCAL AND REMOTE COMMUNICATIONS	2-22
2-6	EXTENDED FUNCTION CODES FOR CONTROL OPERATIONS	2-23
2-7	MAXIMUM NUMBER OF BYTES ERASED	2-24
2-8	EXTENDED FUNCTION CODES FOR DATA TRANSFER OPERATIONS	2-25

TABLES (Continued)

2-9	MAGNETIC TAPE DEVICE DEPENDENT STATUS CODES	2-28	
2-10	FUNCTION CODE BIT POSITIONS FOR GAPLESS MODE DATA TRANSFER REQUESTS	2-32	
2-11	MAGNETIC TAPE DEVICE DEPENDENT STATUS CODES (GAPLESS ONLY)	2-35	
2-12	EXTENDED FUNCTION CODES FOR GAPLESS I/O OPERATION	2-39	
3-1	SVC 2 FUNCTION CODES	3-1	
3-2	TIME OF DAY VALUES CALCULATED IN SECONDS FROM MIDNIGHT	3-42	
3-3	TASK OPTIONS FROM THE TASK CONTROL BLOCK	3-93	
3-4	SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE	3-99	
3-5	TASK WAIT STATUS BIT DEFINITIONS	3-109	
6-1	SVC6.FUN FUNCTIONS	6-5	
6-2	DESCRIPTION OF FUNCTION CODE FIELD FOR SVC 6 CALLS	6-6	
6-3	EXTENDED LOAD OPTIONS FIELD BIT DEFINITIONS	6-14	
6-4	WAIT STATUS BIT DEFINITIONS	6-43	
6-5	SVC 6 ERROR CODES	6-44	
7-1	SVC 7 FUNCTION CODE BIT DEFINITIONS	7-6	
7-2	ALLOWABLE ACCESS PRIVILEGE CHANGES	7-14	
7-3	DESCRIPTION AND MASK VALUES OF THE DEVICE ATTRIBUTES FIELD	7-19	
7-4	ACCESS PRIVILEGES DEFINITIONS	7-26	
7-5	DATA COMMUNICATIONS ACCESS METHODS DEFINITIONS	7-28	
7-6	READ/WRITE PROTECTION KEYS DEFINITIONS	7-30	
7-7	SVC 7 ERROR CODES	7-32	
8-1	TASK STATUS WORD BIT DEFINITIONS	8-4	
9-1	OVERLAY ERROR CODES AND MEANINGS	9-3	
10-1	SVC 13 FUNCTION CODES	10-1	
INDEX		Ind-1	

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 Supervisor Call, (SVC 1) that is used to request specific I/O services from the OS/32 I/O supervisor. Chapter 3 details 22 general service functions provided by SVC 2. Chapter 4 presents the format for the End of Task Supervisor Call, (SVC 3) which is used to terminate task execution. Chapter 5 provides information on user-controlled loading of Link-generated overlays through the Fetch Overlay Supervisor Call (SVC 5). Chapter 6 describes the Intertask Communications Supervisor Call (SVC 6). Chapter 7 details the File Handling Services Supervisor Call (SVC 7) which provides file and device handling functions supported by the file manager and the data communications subsystem. Chapter 8 describes how SVC 9 is used to replace the current task status word (TSW) located in the task control block (TCB) with a new user-specified TSW. Chapter 9 provides information on SVC 10, which handles the automatic loading of overlays generated by Link. Brief descriptions of the Auxiliary Processing Unit (APU) Control Supervisor Call (SVC 13), User Supervisor Call (SVC 14), and the Data Communications Device Dependent I/O Supervisor Call (SVC 15) are given in Chapters 10, 11, and 12 respectively.

Revision 01 includes a description of the SVC 2 code 29 for unpacking file descriptors, the SVC 7 fetch logical attributes of open file function, and the assign logical processing unit (LPU) and transfer to LPU function of SVC 6. This manual also introduces the APU Control Supervisor Call (SVC 13) that is used by tasks running on the Perkin-Elmer 3200MPS multiprocessor system.

This manual is intended for use with the OS/32 R06.2 software release and higher. However, additional material specifically related to the Model 3200MPS System has also been included. These Model 3200MPS System features are supported by the OS/32 R07.1 software release and higher. Throughout the text these features are identified as applicable only to the Model 3200MPS System.

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 a supervisor call (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.

This manual describes the SVCs designed for general use. SVCs designed exclusively for use in system level programs are described in the OS/32 System Level Programmer Reference Manual. These SVCs are indicated by an asterisk (*) or plus sign (+) in Table 1-1.

TABLE 1-1 OS/32 SUPERVISOR CALLS

SUPERVISOR CALL	FUNCTION
SVC 0*	User-written SVC
SVC 1	Input/output (I/O) request
SVC 2 Code 0*	Make journal entries
SVC 2 Code 1	Pause
SVC 2 Code 2	Get storage
SVC 2 Code 3	Release storage
SVC 2 Code 4	Set status
SVC 2 Code 5	Fetch pointer

TABLE 1-1 OS/32 SUPERVISOR CALLS (Continued)

SUPERVISOR CALL	FUNCTION
SVC 2 Code 6	Convert binary to ASCII hexadecimal or ASCII decimal
SVC 2 Code 7	Log message
SVC 2 Code 8	Interrogate clock
SVC 2 Code 9	Fetch date
SVC 2 Code 10	Time of day wait
SVC 2 Code 11	Interval wait
SVC 2 Code 14*	Internal reader
SVC 2 Code 15	Convert ASCII hexadecimal or ASCII decimal to binary
SVC 2 Code 16	Pack file descriptor (fd)
SVC 2 Code 17	Scan mnemonic table
SVC 2 Code 18	Move ASCII characters
SVC 2 Code 19	Peek
SVC 2 Code 20	Reserved for sequential tasking machines
SVC 2 Code 21	Reserved for sequential tasking machines
SVC 2 Code 23	Timer management
SVC 2 Code 24	Set accounting information
SVC 2 Code 25	Fetch accounting information
SVC 2 Code 26*	Fetch device name
SVC 2 Code 27*	Memory Management
SVC 2 Code 29	Unpack fd
SVC 3	End of task
SVC 5	Fetch overlay
SVC 6+	Intertask communication and control
SVC 7+	File handling services
SVC 9	Load task status word (TSW)
SVC 10	Overlay loading
SVC 13*	Auxiliary Processing Unit (APU) Control
SVC 14	Function determined by user
SVC 15	Communications device dependent I/O

LEGEND

* Documented in OS/32 System Level Programmer Reference Manual

+ Additional functions documented in OS/32 System Level Programmer Reference Manual.

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 The Supervisor Call (SVC) Parameter Block

Associated with each SVC (except SVC 3) 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 as follows:

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 System.

When building a parameter block structure, 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, SYSSTRUC.MLB. See the Common Assembly Language Macro/32 Processor (CAL MACRO/32) and Macro Library Utility Reference Manuals.

Use the following macro library utility commands to display the SYSSTRUC.MLB directory:

```
*L MLU32
```

```
*ST
```

```
CAL MACRO LIBRARY UTILITY/32 03-340R00-00
```

```
MLU >G SYSSTRUC.MLB/S
```

```
MLU >DIR
```

```
01/09/82
```

\$DCB\$	\$TCB\$	\$FCB\$	\$REGS\$	\$UDL\$	\$LIB\$	\$IOB\$
\$SVC1\$	\$ERRC\$	\$SVC13\$	\$APB\$	\$SOPT	\$RREGS	\$EREGS
\$UREGS	\$PSW	\$SPT	\$SPTE	\$\$SPT	\$TABL\$	\$IVT
\$STE	\$PDCB	\$DDCB	\$PSDCB	\$DDE	\$MAGDCB	\$VFDCB
\$SDCB	\$EVN	\$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	\$PFOB
\$FCB	\$FFLG	\$CCB	\$DATB	\$DFLG	\$DXFL	\$SVC1
\$SIXO	\$SVC1ERR	\$SVC4	\$SVC5	\$SVC6	\$SVC7	\$SVC13
\$APST	\$UDL	\$IOB	\$IOBF	\$IOH	\$SPOL	\$TERMUSR
\$AUF	\$MTMSTE	\$ATF	\$GERC	\$EFMG	\$ESYS	\$EMIL
\$MERC	\$ORT	\$ODT	\$SPR	\$TQE	\$TQH	\$TQ27
INTCPARM	\$QH	\$IPCB	\$IRCB	\$ICB	\$VFCHARS	\$TKQ
\$APB	\$APRC	\$APS	\$AOPT	\$TTB	\$LPMT	\$SYP

```
105 MACROS IN LIBRARY MTM:SYSSTRUC.MLB/S
```

```
MLU
```

Use the following macro library utility command to expand the desired structure:

Format:

```
LIST fd,macro
```

Example:

```

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 ADDR
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'                          UNCOND. 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 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 task status word (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 SVC 2, 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:

1. The address specified for the SVC parameter block lies outside task boundaries.
2. The address specified for the SVC parameter block is not aligned on a fullword boundary.
3. 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, if set for the SVC, depends on the particular SVC. Generally, a condition code 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 condition code, use a branch mnemonic that tests for a true condition.

Example:

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

| In this example, the condition code of the 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 Perkin-Elmer Processor User's Manual for
| your installation.

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

2.1 INTRODUCTION

SVC 1 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 to which the I/O is directed. These specifications are indicated through certain fields of the SVC 1 parameter block.

When requesting a read or write operation, the user must describe in the SVC 1 parameter block the data being transferred and the environment during the transfer. For proper execution of a simple data transfer request, specify:

- the structure of the file that a record is being transferred to/from (sequential or random),
- the form the data is in when transferred (ASCII or binary, formatted or image mode), and
- the state the calling task will be in during I/O (I/O proceed, I/O wait, 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 and then decide to 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, status fields, 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 length of data transfer field in the SVC 1 parameter block).

All testing and checking for I/O completion can be accomplished through the SVC 1 parameter block.

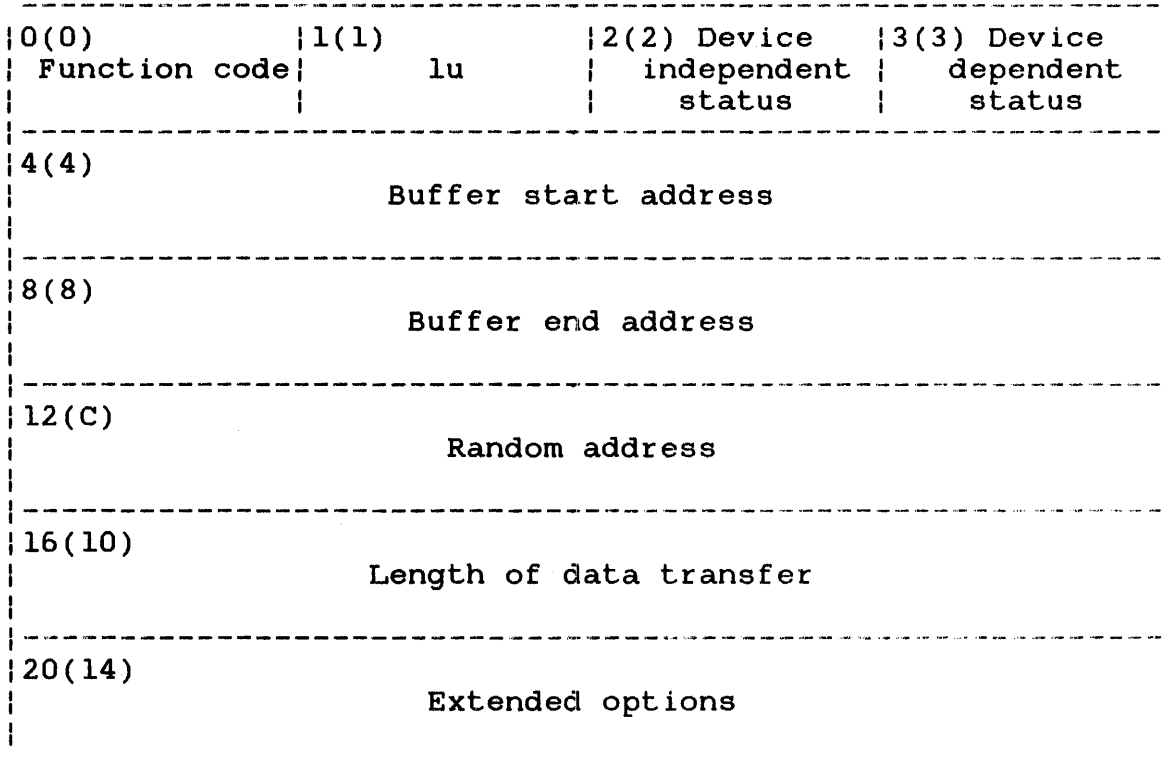
Specific I/O command function requests that can be made through SVC 1 include:

- 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 through the SVC 1 parameter block in Figure 2-1.

2.2 SUPERVISOR CALL 1 (SVC 1)

The SVC 1 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
      .
      .
      .
parblk ALIGN 4
      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 SVC 1 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 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.

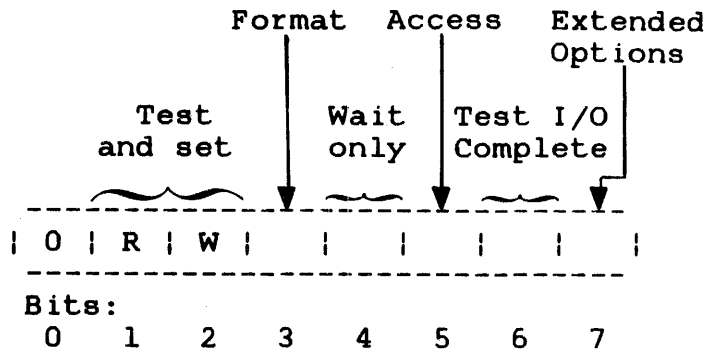


Figure 2-2 Function Code Format for Data Transfer Requests

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	11 = test if a specific record in a file is being used by another task.

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

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
3	ASCII	0 = the data to be transferred is in ASCII format.
	Binary	1 = the data to be transferred is in binary format. If bit 7 of the function code is set to indicate image I/O transfer, the setting of bit 3 is ignored and the data is transferred in image format.
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.

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

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
5	Sequential	0 = access the next logical record.
	Random	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.
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 into a wait state.
	Unconditional proceed	1 = any I/O request made to a device that is busy is rejected if 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 condition code is set to X'F'. However, if there is no outstanding I/O proceed request, the condition code is set to X'0'.

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

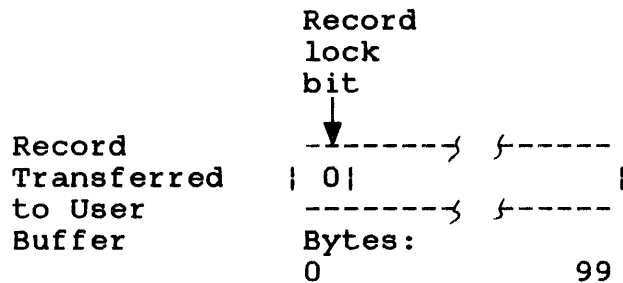
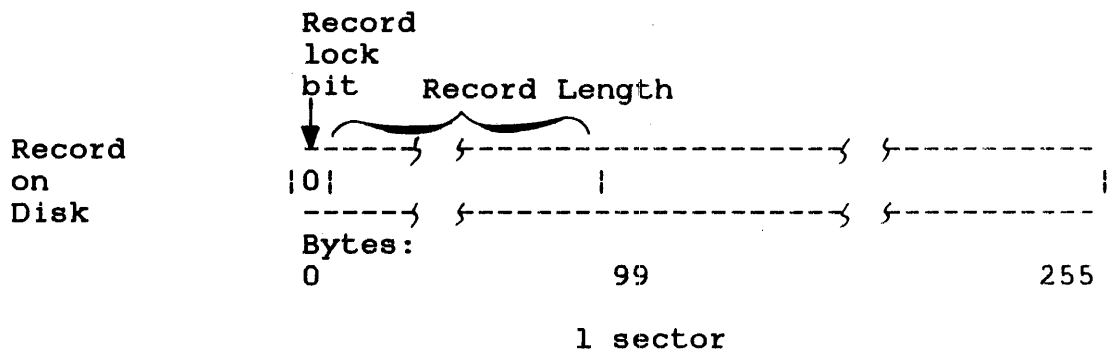
BIT POSITION	BIT NAME	BIT SETTING AND MEANING
		<p>When a test I/O complete request is issued, bit 6 is the only bit in the function code set. If bit 4 is set, it is ignored.</p>
7	<p>Format</p> <p>Extended/ image options</p>	<p>0 - the data being transferred is formatted as indicated by the bit 3 setting of the function code and according to the device type specified.</p> <p>1 - tests the setting of the XSVCL task option. If XSVCL 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.</p> <p>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 or line feeds on writes and will receive exactly what is input on reads.</p>

2.2.1.1 Test and Set

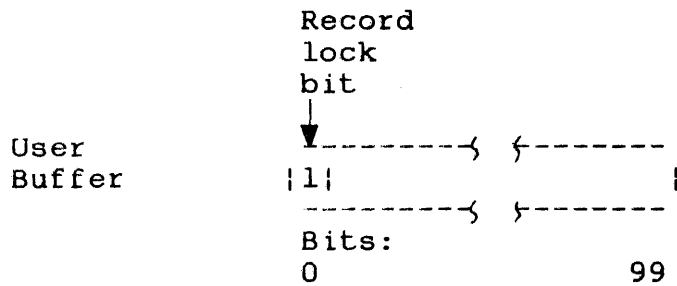
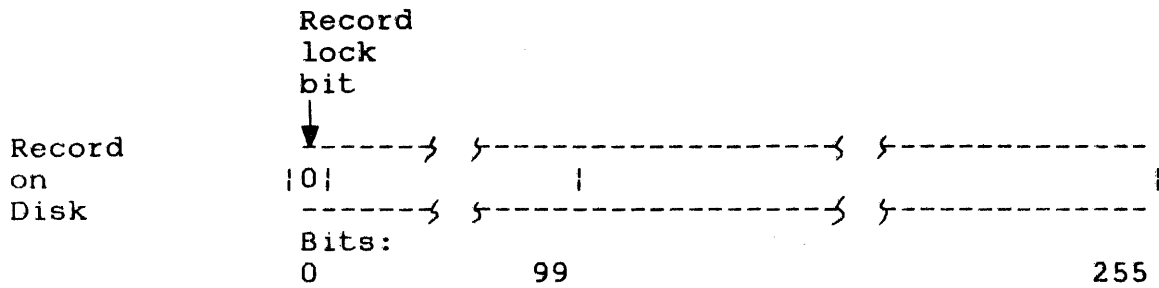
The test and set function can be used to write a program that prevents multiple tasks from modifying a record simultaneously. A task that issues an SVC 1 with the test and set bit enabled notifies other tasks that it is using a record by setting the first bit of that record to 1. This bit setting is called a one-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 zero. 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 SVC 1 with test and set enabled to read a record into its user buffer specified by the SVC 1 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.

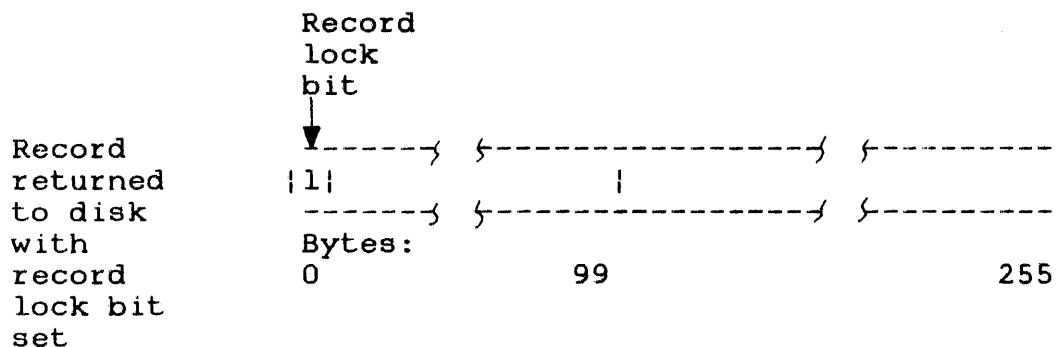


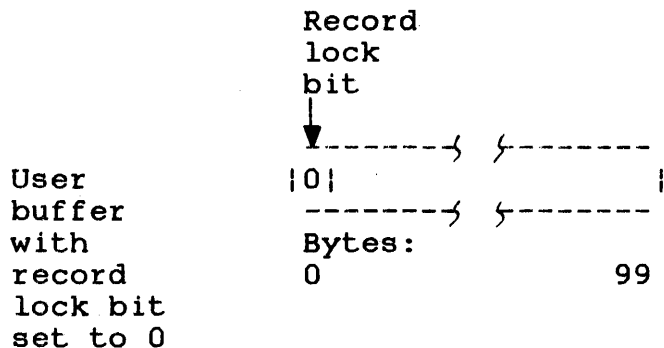
After the record is read into the user buffer, the test and set operation tests the record lock bit. If the bit is 0, the record lock bit in the user buffer is then set to 1. The following diagram shows the record lock bit settings after the test operation is performed.



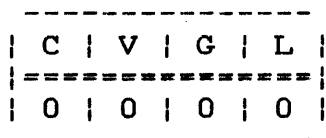
After the record lock bit is tested, the SVC 1 test and set function then sets the record lock bit on disk to 1 so that other tasks attempting to modify the record are notified that the record is in use. SVC 1 sets the record lock bit on disk by copying the contents of the user buffer to the record's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

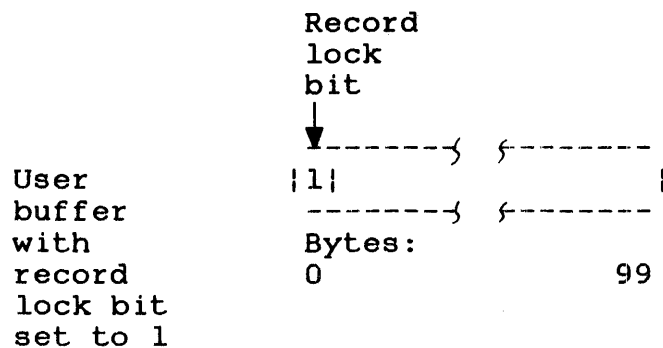
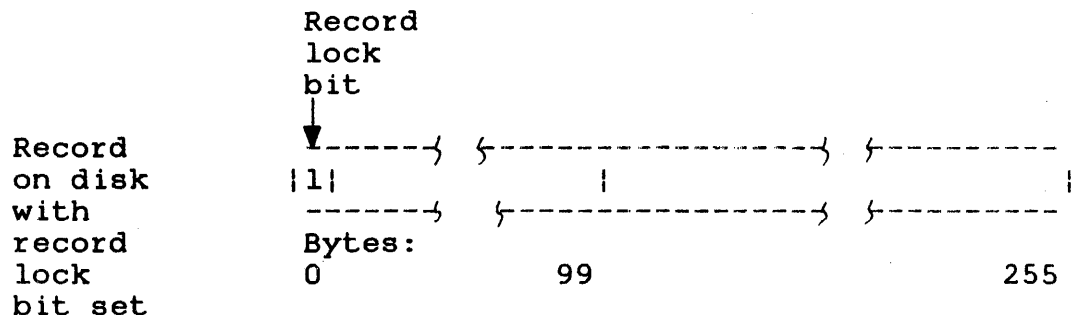




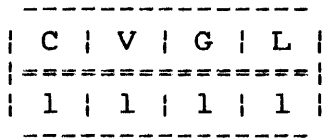
Condition Code



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



Condition Code



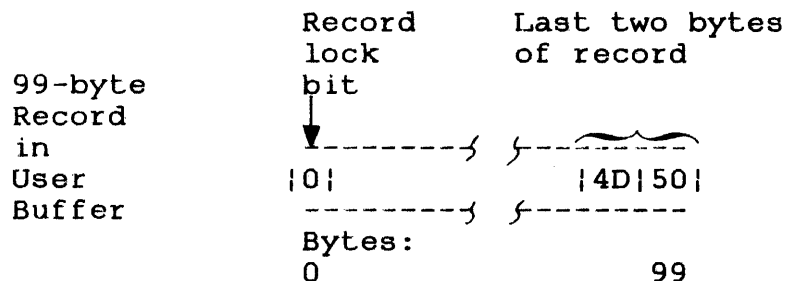
After a test and set operation, a record protection program checks the condition code. If the condition code is zero, the task can then proceed to modify the record. If the condition code is X'F', the task should retry the test and set operation before attempting to modify the record.

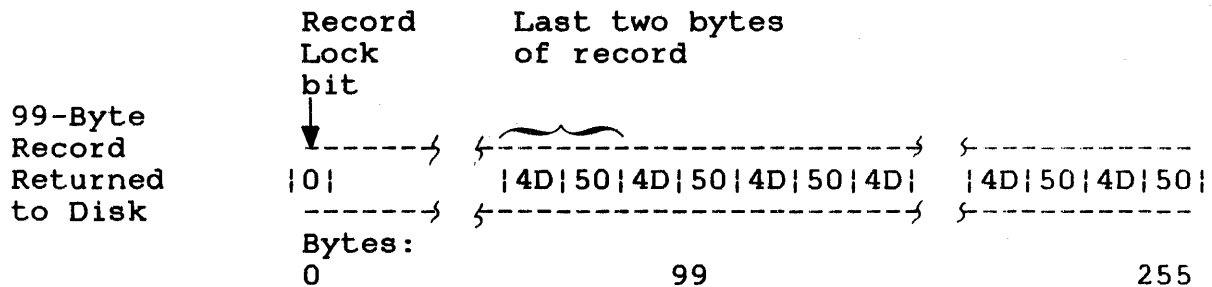
To unlock the record on disk, the task that set the record lock bit should write the record in its user buffer back into its original location on disk, whether or not the task modified the record.

If the size of the user buffer is less than the size of the record, the following will occur 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 will be 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 will be 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 condition code 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 set to 0), setting the condition code 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   *
*
LOOP     LIS   1,0           SET UP FIRST RECORD NO.
          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 FILE 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 four ways to check for I/O completion:

1. Execute a test I/O complete operation.
2. Monitor the status fields in the SVC 1 parameter block issuing the request.
3. Take a trap when I/O is completed and branch to a service routine.
4. Issue a wait I/O request to the device specified by the SVC 1 making the request. This function will stop task execution until I/O is completed.
5. Queue I/O requests by specifying the IOBLOCK parameter of the Link OPTION command and issue the wait only function. This will stop task execution until all queued requests to a specified device are completed.

An SVC 1 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 SVC 1 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 IQBLOCK= $\left. \begin{matrix} \text{(b)} \\ \text{(1)} \end{matrix} \right\}$

Parameter:

b is a decimal number from 1 through 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, SVC 1 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:

$$b + 1 + \text{number of logical units assigned to task}$$

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 condition code 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. Then, task execution 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 SVC 1 parameter block.

To issue the wait only request, set the SVC 1 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 SVC 1 wait only parameter block if any of the following conditions occur:

- Logical unit is illegal (code X'81').
- Logical unit is unassigned (code X'81').
- 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:

1. Execute a test I/O complete operation.
2. Monitor the status fields in the parameter block for the command function status to be returned.
3. Issue a wait only request to the device specified by the SVC 1 making the request. This function will stop 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'CO'	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'	Reserved for driver dependent functions.
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 through 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 SVC 7 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 are 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.
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 may 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 the 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'	<p>Parity - An even or odd parity error occurred on a data transfer request.</p> <p>Recoverable error - The I/O request is recoverable and can be retried. A write request was issued to a write-protected device.</p> <p>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.</p>
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 - I/O is completed and no error occurred.

2.2.5 Device Dependent Status

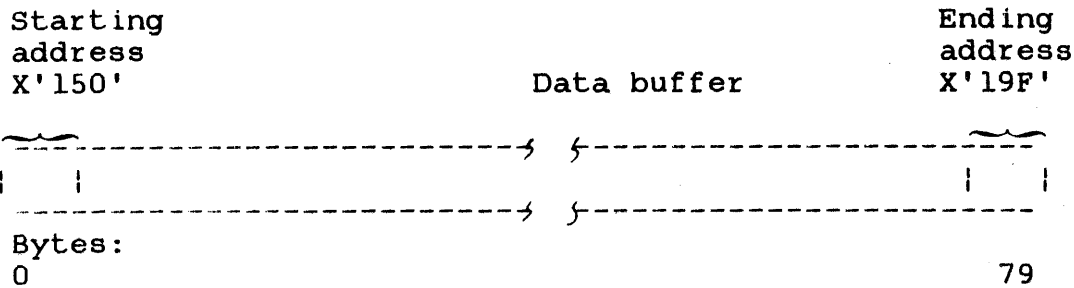
The execution status of an I/O request that is directly related to the unique characteristics of the device being used is returned to the device dependent status field of the parameter block. See Table 2-4. The data remaining in this field from a previous I/O request are not modified until a subsequent 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 timeout - A read or write timeout 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 completed and no error occurred.

2.2.6 Buffer Start/Buffer End Address

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 SVC 1 extended option field are executed. The extended options fullword format is dependent upon the device that an I/O request is directed to. 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 SVC 1 parameter block for devices supported by the communications subsystem.

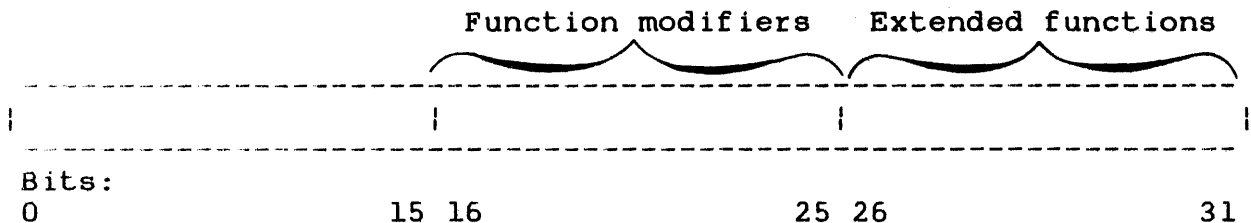


Figure 2-3 Extended Options Fullword Format for Nonmagnetic Tape Devices

| 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 SVC 1 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 SVC 1 EXTENDED OPTIONS FOR LOCAL AND REMOTE COMMUNICATIONS

BIT POSITIONS	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.
3-7		00000 = these bits are reserved for future use.
8	Vertical forms control (VFC)	1 = requests vertical forms control option for an ASCII I/O operation.
9-15		0000000 = reserved.

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 SVC 1 parameter blocks used for magnetic tape I/O operations.

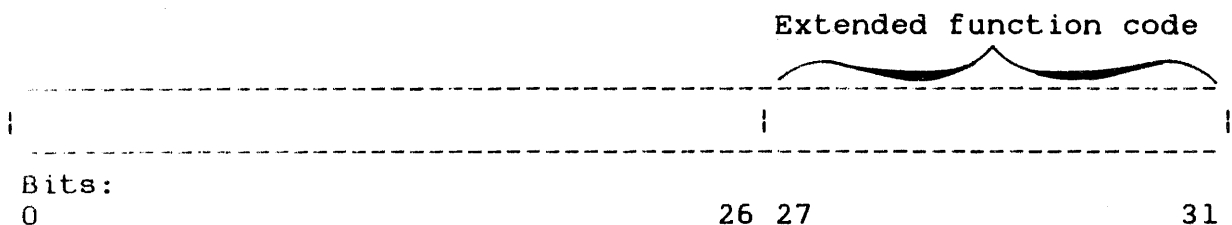


Figure 2-4 Extended Options Fullword Format for Magnetic Tape I/O Operations

If the extended function code requires an additional parameter, the most-significant bits (0-7) 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 SVC 1 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	OPERATION/EXPLANATION
0	REWIND AND UNLOAD - The tape is rewound to beginning of tape and 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.

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

EXTENDED FUNCTION CODE	OPERATION/EXPLANATION
8	<p>READ DRIVE STATUS - A task can read 8 status halfwords into the buffer space specified in the SVC 1 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.</p>
9	RESERVED
10	<p>ERASE TAPE - Erases a variable length of tape, beginning at the current position. The length of tape erased is determined by the following formula:</p> $\text{Length of Tape Erased} = \frac{\text{No. of Bytes in User Buffer}}{\text{Current Tape Density}}$ <p>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 OS will erase the maximum number of bytes and 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.</p> <p style="text-align: center;">NOTE</p> <p>For device code 65, the current density is assumed to be 800BPI. If the current density for device code 65 is 1600BPI, the length of tape erased is twice as long as requested.</p>
11-31	RESERVED

TABLE 2-7 MAXIMUM NUMBER
OF BYTES
ERASED

TAPE DENSITY (BPI)	NUMBER OF BYTES
800	200,000
1600	400,000
6250	1,000,000

Table 2-8 contains the extended function codes 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	OPERATION/EXPLANATION
0	NO EXTENDED FUNCTIONS The bit settings of the standard function (byte 1 of the SVC 1 parameter block) are read and used to determine the operation to be performed.
1	READ BACKWARD The tape drive will read 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 will perform retries on that operation up to a number of times corresponding to the value set in the sysgen macro library. (The read bit of the SVC 1 function code should be set.) Requires hardware support.

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

EXTENDED FUNCTION CODE	OPERATION/EXPLANATION
2	<p data-bbox="488 359 813 401">GAPLESS OPERATION</p> <p data-bbox="488 422 1398 695">The driver reads or writes multiple data buffers to or from magnetic tape with no interrecord buffers gaps, using only one SVC 1. Gapless operation requires the use of a special SVC 1 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.</p>
3	<p data-bbox="488 716 1365 779">GAPLESS OPERATION WITH BUFFER TRANSFER REPORTING</p> <p data-bbox="488 800 1398 1136">The driver reads or writes multiple data buffers to or from magnetic tape with no interrecord gaps, using only one SVC 1. The task receives a buffer trap each time the driver uses another buffer. Gapless operation requires the use of a special SVC 1 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.</p>
4	<p data-bbox="488 1157 1333 1199">READ FORWARD AND IGNORE DATA TRANSFER ERRORS</p> <p data-bbox="488 1220 1398 1572">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 data transfer field to verify that all of the specified data was actually read. (The read bit of the SVC 1 function code should be set.)</p>

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

EXTENDED FUNCTION CODE	OPERATION/EXPLANATION
5	<p>READ BACKWARD AND IGNORE DATA TRANSFER ERRORS</p> <p>The tape drive will read 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 SVC 1 function code should be set.) Requires hardware support.</p>
6	<p>USER CONTROL OF RETRIES FOR DATA TRANSFER ERRORS</p> <p>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 SVC 1 function code should be set.)</p>
<p>NOTE</p> <p>If extended function code 6 is not specified, the number of retries defaults to the value set in the sysgen macro library.</p>	

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

EXTENDED FUNCTION CODE	OPERATION/EXPLANATION
7	<p>READ BACKWARDS AND ALLOW USER CONTROL OF RETRIES FOR DATA TRANSFER ERRORS</p> <p>The tape drive will read 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 other they are read; i.e., reverse. If an error occurs, 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 is 255. (The read bit of the SVC 1 function code should be set.)</p>
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 SVC 1.

2.2.7.3 Device Dependent Status Codes for Magnetic Tape Operations

The device dependent and device independent status fields of the SVC 1 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 will cease upon detection of most of these errors.

TABLE 2-9 MAGNETIC TAPE DEVICE DEPENDENT STATUS CODES

STATUS CODE	MEANING
8282	Timeout - A read or write timeout 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	Timeout - A control operation timeout occurred.
82FD	Timeout - A read, read backward, read drive status, write or write filemark timeout condition occurred.
82FE	Read backward at load point - Load point was reached before a read backward operation terminated.
82FF	Timeout - A read, read backward, or write timeout 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 only supported 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	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	<p>Illegal function - the function code indicated a data transfer operation, but neither the read nor write bit was set.</p> <p>The function code indicated a control operation, but none of the other bits in the function code were set.</p> <p>The function code indicated an extended control operation, but the extended SVC 1 task option was disabled.</p> <p>The requested function is not supported by the device or assigned access privileges.</p> <p>Illegal extended function code - an undefined function, or a function not supported by the specified tape drive, was indicated.</p> <p>The extended function code indicated a read operation, but the standard function code has the write bit set.</p> <p>Buffer size too small - the buffer for a read, read backward, or write operation was less than 4 bytes. The buffer for the read drive status was smaller than 16 bytes.</p> <p>Erase tape at load point - an erase tape operation was attempted when a tape was at load point.</p> <p>User retries too large - the maximum number of retries specified for a write operation was greater than 45.</p>

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 SVC 1. A task can have only one ongoing gapless SVC 1 at a time. The format of a gapless mode SVC 1 parameter block differs from the standard SVC 1 parameter block. The gapless SVC 1 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 SVC 1 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, and then transferring these small segments of data sequentially. The gapless mode SVC 1 parameter block can only be used for gapless I/O operations.

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

The gapless mode SVC 1 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 SVC 1 parameter block. Figure 2-5 presents the gapless mode SVC 1 parameter block and coding example.

0(0) Function Code	1(1) lu	2(2) Device independent status	3(3) Device dependent status
4(4)	OUT-QUEUE start address		
8(8)	IN-QUEUE start address		
12(C)	Buffer length		
16(10)	Length of last buffer		
20(14)	Extended options		

```

SVC 1,parblk
.
.
.
ALIGN4
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 SVC 1 Gapless Mode Parameter Block Format and Coding

Fields:

Function code	is a 1-byte field indicating that the request is a data transfer request, the specific 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 for gapless operation.
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 will be written to tape. If the operation is a gapless read, these buffers will be 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. However, the amount of space used in the last buffer may 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 may 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 2-8 defines each function code bit setting.

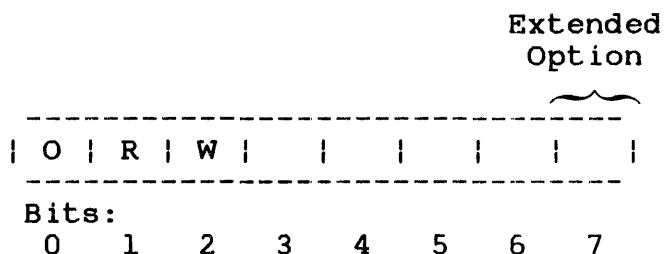


Figure 2-6 Function Code Format for Gapless Mode Data Transfer Requests

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.)

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

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
3	Not used in gapless mode	
4	<p>I/O proceed</p> <p>Wait I/O</p> <p>Wait only</p>	<p>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.</p> <p>1 = stop task execution, initiate data transfer to the device, and wait until the completion of I/O.</p> <p>1 = task execution stops and waits until the completion of all queued I/O proceed requests to the specified lu.</p> <p>When a wait only request is issued, bit 4 is the only bit set in the function code.</p>
5	Not used in gapless mode	
6	Conditional proceed	<p>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 into a wait state.</p>

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

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

2.3.3 Logical Unit (lu)

An lu is a decimal number ranging from 0 through 254. The highest lu number that a task can be assigned to 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 SVC 7 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.

2.3.4 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 2-9 and 2-11 presents the error status codes for gapless operation. A gapless operation will cease upon detection of any one of these errors.

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

STATUS CODE	MEANING
X'8485'	Read/write timeout - A read or write timeout 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 user on 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 SVC 1 handler will reject 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.

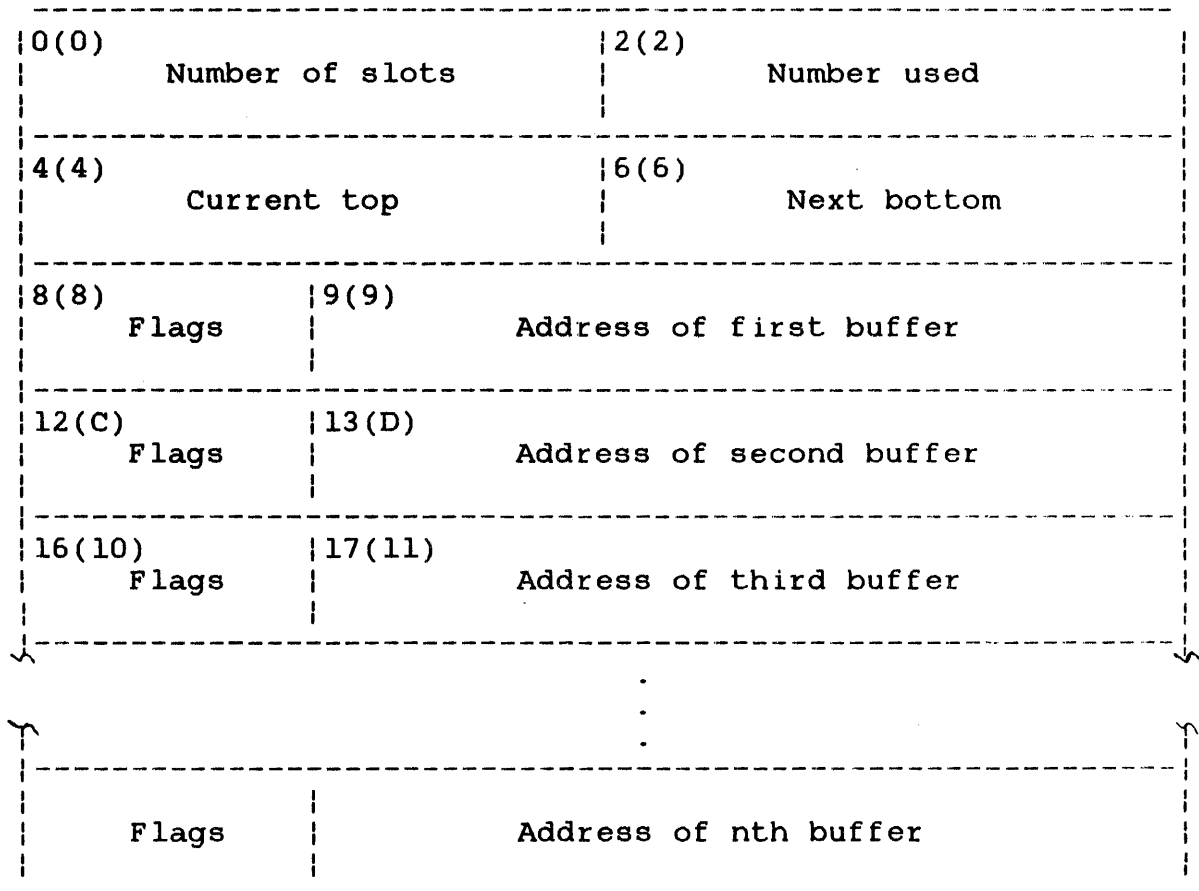


Figure 2-7 IN-QUEUE or OUT-QUEUE Structure

Fields:

Number of slots

Number used

Current top

Next bottom

} These fields are standard list parameters explained in the Common Assembly Language/32 (CAL/32) Reference Manual.

Flags

is a 1-byte field. The setting of bit zero in this field identifies whether the buffer is the last buffer in the list. If bit zero is set to 0, the buffer is not the last buffer. If bit zero 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 zero set to 1. A gapless read operation may be terminated in one of two ways. If the user wishes to only read 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 zero 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 zero 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 64k buffers, the total buffer space available is 320K bytes (or 120K bytes less than is required to complete the write operation). After the first buffer has been written, the starting address of the buffer will be 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 64K bytes 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 56K bytes. Note that when the task transfers the address of the second buffer from the OUT-QUEUE to the IN-QUEUE, the zero bit 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 SVC 1 parameter block prior to the start of the operation.

The task should use an ABL instruction to add buffer entries to the IN-QUEUE and an 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 SVC 1 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 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 will be 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 SVC 1 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 will be required. The first two buffers will contain 128kb of information; however the third buffer will contain only 22kb of information. The value 22kb will be returned to the length of last buffer field in this example. If desired, this field may 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 zero set to 1, this field must be given to the driver by the task.

NOTE

If a gapless read does not reach a 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 SVC 1 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 SVC 1. 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 0-1 and 4-31 are not used with the gapless mode SVC 1 parameter block.

CHAPTER 3
GENERAL SERVICE FUNCTIONS SUPERVISOR CALL 2 (SVC 2)

3.1 INTRODUCTION

SVC 2 provides general service functions distinguished from one another by a specific function code number. Each SVC 2 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 SVC 2 function codes with a brief description of each.

TABLE 3-1 SVC 2 FUNCTION CODES

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

TABLE 3-1 SVC 2 FUNCTION CODES (Continued)

SVC 2 CODE	NAME	FUNCTION
SVC 2 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).
SVC 2 code 6	Convert binary to ASCII hexadecimal or ASCII decimal	Converts a binary number to either an ASCII hexadecimal or ASCII decimal number.
SVC 2 code 7	Log message	Sends a message to the appropriate log device regardless of the current logical unit (lu) assignments.
SVC 2 code 8	Interrogate clock	Sends the user the current time of day calculated in seconds from midnight in binary or in formatted ASCII.
SVC 2 code 9	Fetch date	Sends the user the current date in formatted ASCII.
SVC 2 code 10	Time of day wait	Places the calling task in a wait state until a specific time of day.
SVC 2 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.
SVC 2 code 14*	Internal reader	Allows a foreground task loaded from the system console to invoke operator and CSS commands.
SVC 2 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 SVC 2 FUNCTION CODES (Continued)

SVC 2 CODE	NAME	FUNCTION
SVC 2 code 16	Pack file descriptor	Processes a user-specified unpacked file descriptor (fd) into a packed format to be used by the operating system.
SVC 2 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.
SVC 2 code 18	Move ASCII characters	Moves a specified number of ASCII characters in memory from the sending location to a receiving location.
SVC 2 code 19	Peek	Obtains user-related information from operating system data structures.
SVC 2 code 20	Expand allocation	Reserved for sequential tasking machines. Provides for compatibility with current 32-bit operating systems.
SVC 2 code 21	Contract allocation	Reserved for sequential tasking machines. Provides for compatibility with current 32-bit operating systems.
SVC 2 code 23	Timer management	<p>Schedules the addition of a parameter to a task queue on completion of a specified interval or a repetitive interval.</p> <p>Puts a task in a wait state until completion of an interval.</p> <p>Determines the time remaining for a previously established interval to expire.</p> <p>Cancels a previously established interval.</p>

TABLE 3-1 SVC 2 FUNCTION CODES (Continued)

SVC 2 CODE	NAME	FUNCTION
SVC 2 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.
SVC 2 code 25	Fetch accounting information	Fetches accounting information and stores it in a user-specified receiving area.
SVC 2 code 26*	Fetch device name	Searches the volume mnemonic table for a user-supplied volume name and returns the name of the device on which that volume is mounted.
SVC 2 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 task control block (TCB).
SVC 2 code 29	Unpack file descriptor (fd)	Converts a packed file descriptor (fd) from the file directory or an SVC 7 parameter block to its unpacked format.

LEGEND

* indicates SVC is documented in OS/32 System Level Programmer Reference Manual.

3.2 SVC 2 CODE 1: PAUSE

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

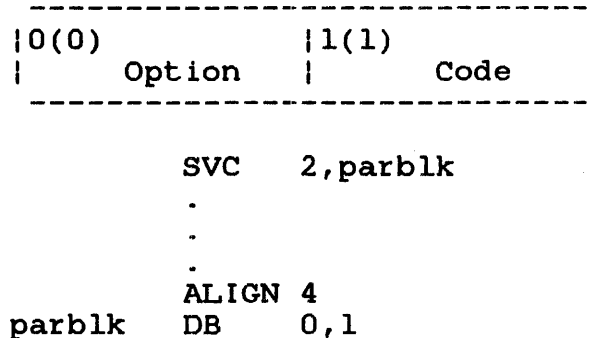


Figure 3-1 SVC 2 Code 1 Parameter Block Format and Coding

This parameter block must be 2 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 SVC 2.

After executing SVC 2 code 1, the following message is displayed on the system console:

```
TASK PAUSED
```

If the task is running under the multi-terminal monitor (MTM), the above message is displayed on the user console.

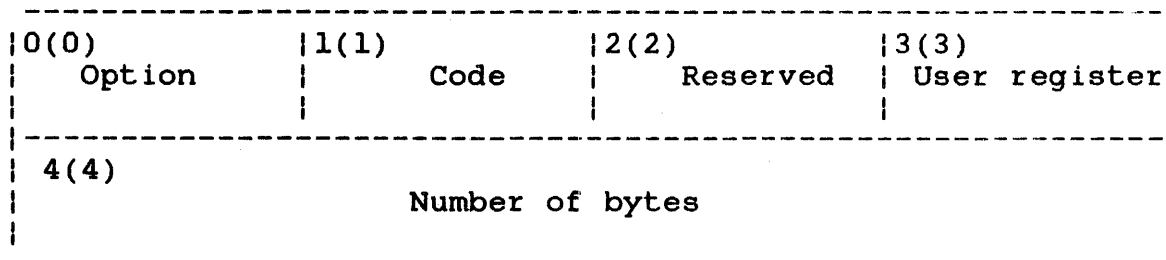
While the task is paused, the operator can issue commands directed to the paused task to change the task environment. To continue task execution, enter the CONTINUE command. Task execution resumes with the instruction immediately following SVC 2 code 1.

3.3 SVC 2 CODE 2: GET STORAGE

SVC 2 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 tasks impure segment between UTOP and CTOP. For more information on this segment, see the OS/32 Application Level Programmer Reference Manual.

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

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



```

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

```

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

This parameter block must be 8 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 all of 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 SVC 2.

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

User register is a 1-byte field that must contain a decimal number ranging from 0 through 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 SVC 6.

After executing SVC 2 code 2, the condition code is set as follows:

Condition codes

C	V	G	L
0	0	0	0
0	1	0	0

Normal termination

User-specified number of bytes is a negative value or a value greater than the task's allocated memory size.

NOTE

When SVC 2 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. SVC 2 code 5 updates the address in the UDL.

3.3.1 SVC 2 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

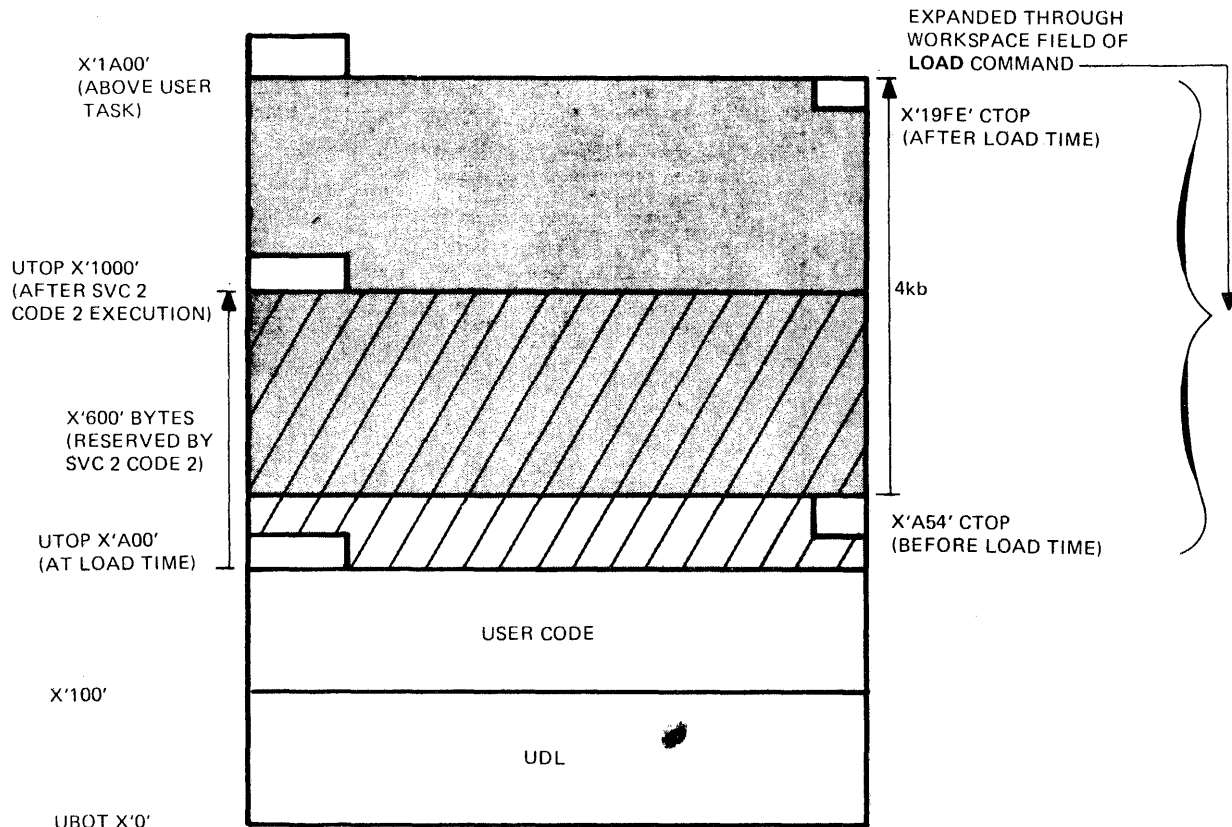
```

This example is illustrated in Figure 3-3. A task is loaded with a task workspace area of 4kb specified in the LOAD command. After the task is loaded, UTOP is located at X'A00' and CTOP is located at X'19FE'. After executing SVC 2 code 2, UTOP is adjusted to X'1000'. The remaining unused portion (area between X'A00' and X'1A00') 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 condition code is set to 4 (V bit set).

5630



NOTES

UTOP is the starting address of the first contiguous fullword outside the user code.

CTOP is the starting address of the last halfword within the allocated task address space.

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

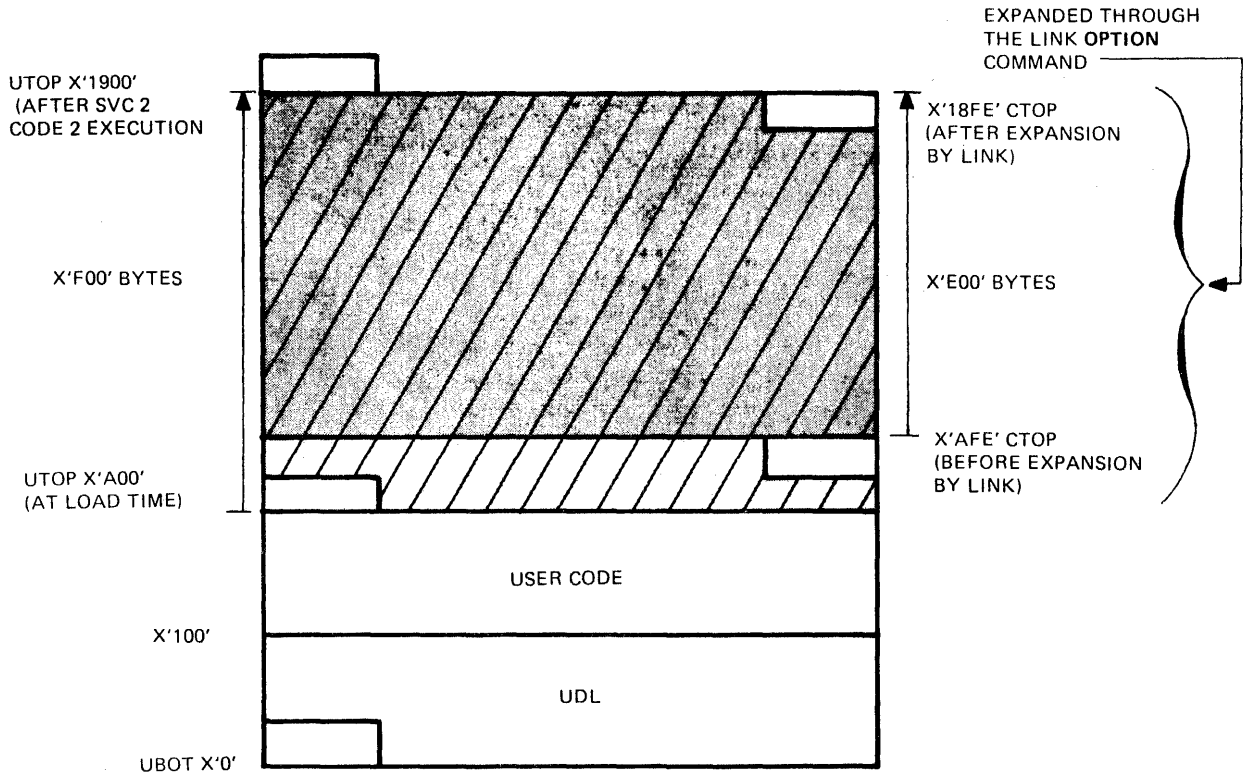
3.3.2 SVC 2 Code 2, Option X'80'

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 3-4. A task is linked with a workspace of 4kb. After the task is loaded, UTOP is located at X'A00'. After executing SVC 2 code 2, UTOP is adjusted to X'1900'.



NOTE

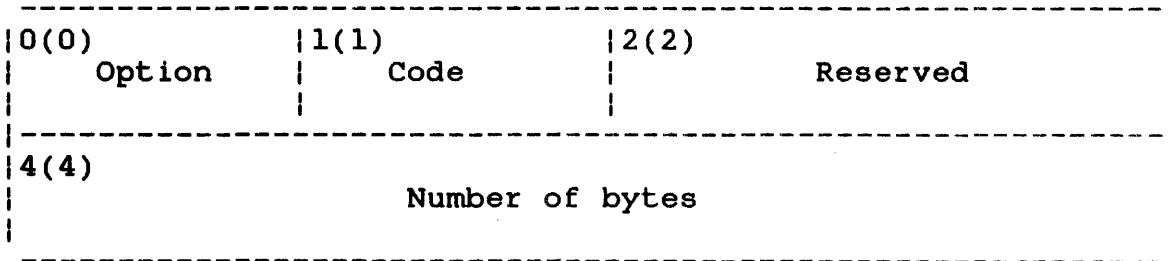
UTOP is the starting address of the first contiguous fullword outside the user code.

CTOP is the starting address of the last halfword within the allocated task address space.

Figure 3-4 Task Impure Segment for SVC 2 Code 2, Option X'80'

3.4 SVC 2 CODE 3: RELEASE STORAGE

SVC 2 code 3 releases the workspace area in the unused portion of of the task impure segment that had been reserved by a previous SVC 2 code 2 (See Section 3.3). Releasing the reserved workspace for external subroutines does not decrease the task's allocated memory size. The SVC 2 code 3 parameter block is shown in Figure 3-5.



```

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

```

Figure 3-5 SVC 2 Code 3 Parameter Block Format and Coding

This parameter block is 8 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 SVC 2.

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 SVC 2 code 3, the condition code is set as follows:

Condition codes

C	V	G	L
0	0	0	0
0	1	0	0

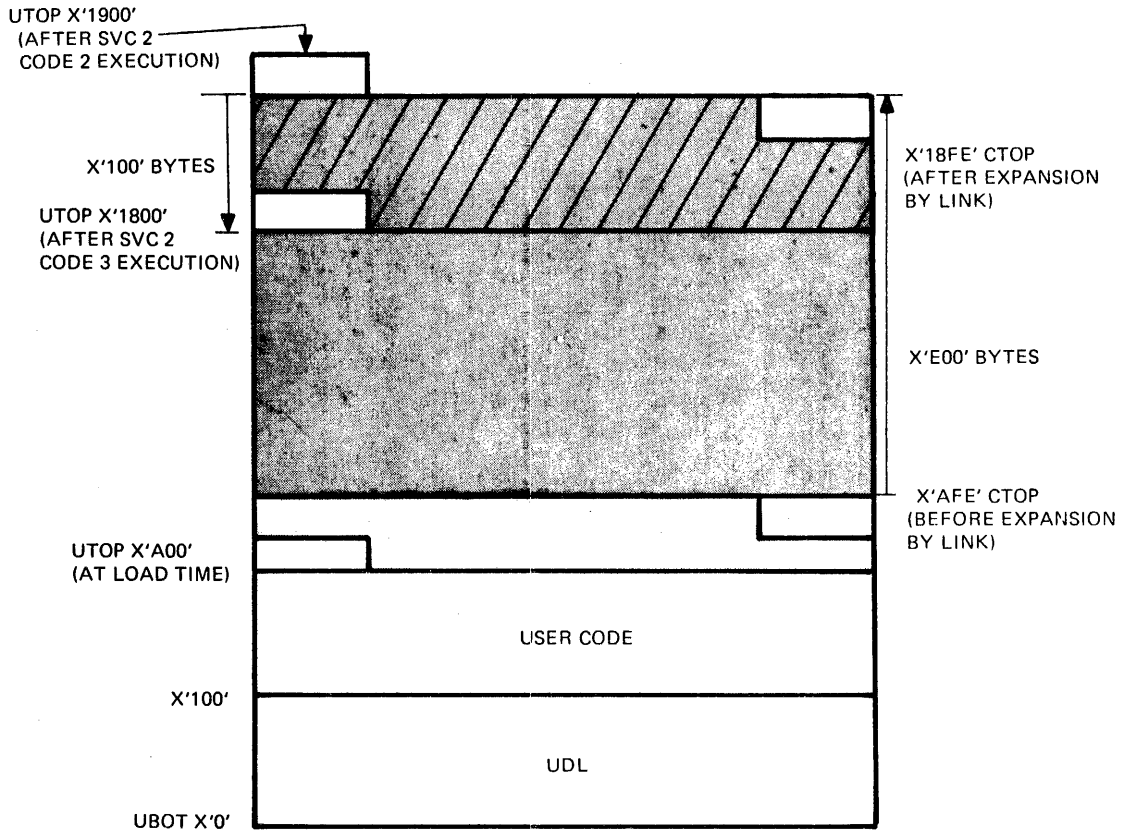
Normal termination

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 3-6 illustrates this example. A task was linked with a workspace of 4kb and loaded into memory. After the task is loaded, UTOP is located at X'A00' and CTOP is located at X'18FE'. After executing SVC 2 code 2, UTOP is adjusted to X'1900'. After executing SVC 2 code 3, 256 bytes of reserved storage are released, adjusting UTOP to X'1800'.



NOTE

UTOP is the starting address of the first contiguous fullword outside the user code.

CTOP is the starting address of the last halfword within the allocated task address space.

Figure 3-6 Task Impure Segment for SVC 2 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 condition code is set to 4 (V bit set).

```

-----
| SVC 2 |
| CODE 4 |
-----

```

3.5 SVC 2 CODE 4: SET STATUS

SVC 2 code 4 modifies the arithmetic fault interrupt bit and the condition code settings in the program status word (PSW). Figure 3-8 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 condition code setting is modified, the current 4-bit setting is replaced with a new 4-bit setting. This is accomplished through the SVC 2 code 4 parameter block shown in Figure 3-7.

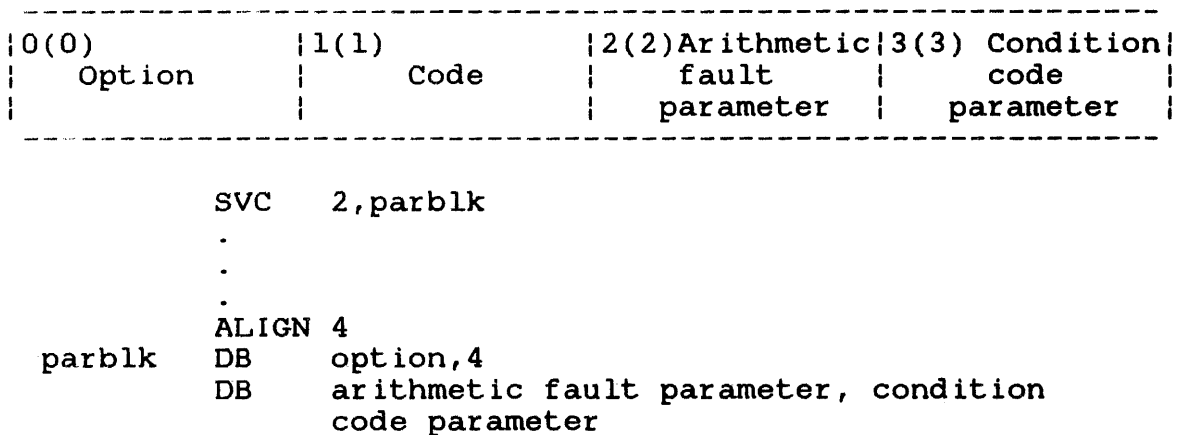


Figure 3-7 SVC 2 Code 4 Parameter Block Format and Coding

This parameter block is 4 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 condition code in the PSW.
 - Option X'80'-modifies only the condition code in the PSW. See Figure 3-8.

Code is a 1-byte field that must contain the decimal value 4 to indicate code 4 of SVC 2.

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 model 7/32 and 8/32 processors. For 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 parameter with a hexadecimal value ranging from X'00' to X'0F'.

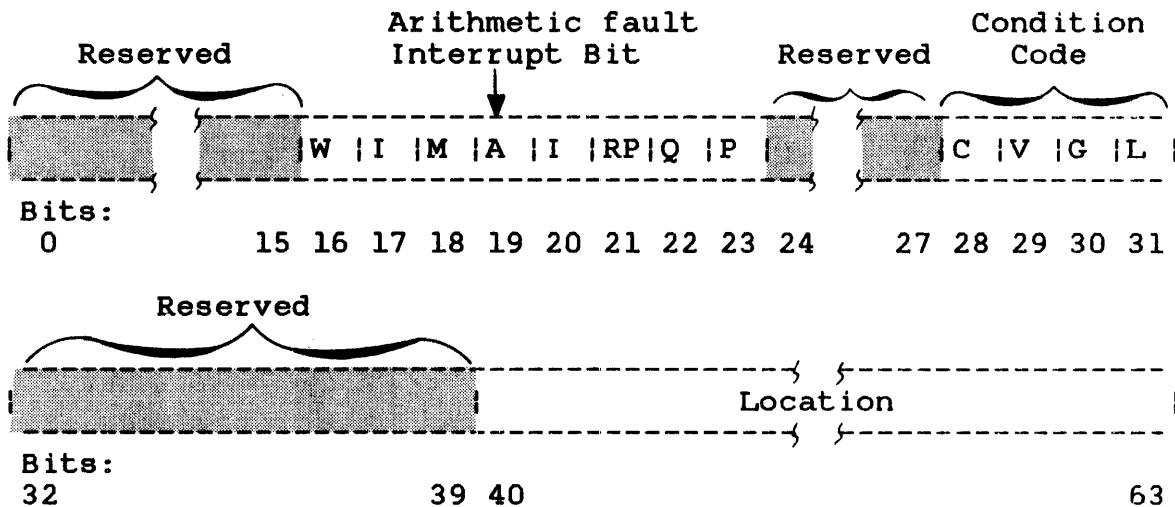


Figure 3-8 Program Status Word (PSW)

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

Condition codes

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.

3.5.1 SVC 2 Code 4, Option X'00'

If the SVC 2 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' through X'0F' in the condition code field, all arithmetic faults are ignored for model 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 condition code value in the PSW is replaced with the value specified in the condition code field of the parameter block.

If the SVC 2 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' through X'0F' in the condition code field, all arithmetic fault interrupts are enabled. The current condition code value in the PSW is replaced with the value specified in the condition code field of the parameter block.

3.5.2 SVC 2 Code 4, Option X'80'

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

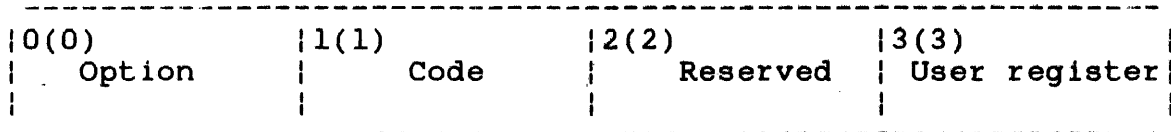
```

-----
| SVC 2 |
| CODE 5 |
-----

```

3.6 SVC 2 CODE 5: FETCH POINTER

SVC 2 code 5 loads the starting address of a task's user dedicated location (UDL) into a user-specified register. It then stores the current addresses of UBOT, UTOP, and CTOP, located in the task control block (TCB) into their corresponding locations in the task UDL. This is accomplished through the SVC 2 code 5 parameter block shown in Figure 3-9.



```

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

```

Figure 3-9 SVC 2 Code 5 Parameter Block Format and Coding

This parameter block is 4 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 SVC 2.
- Reserved is a reserved 1-byte field that must contain a 0.

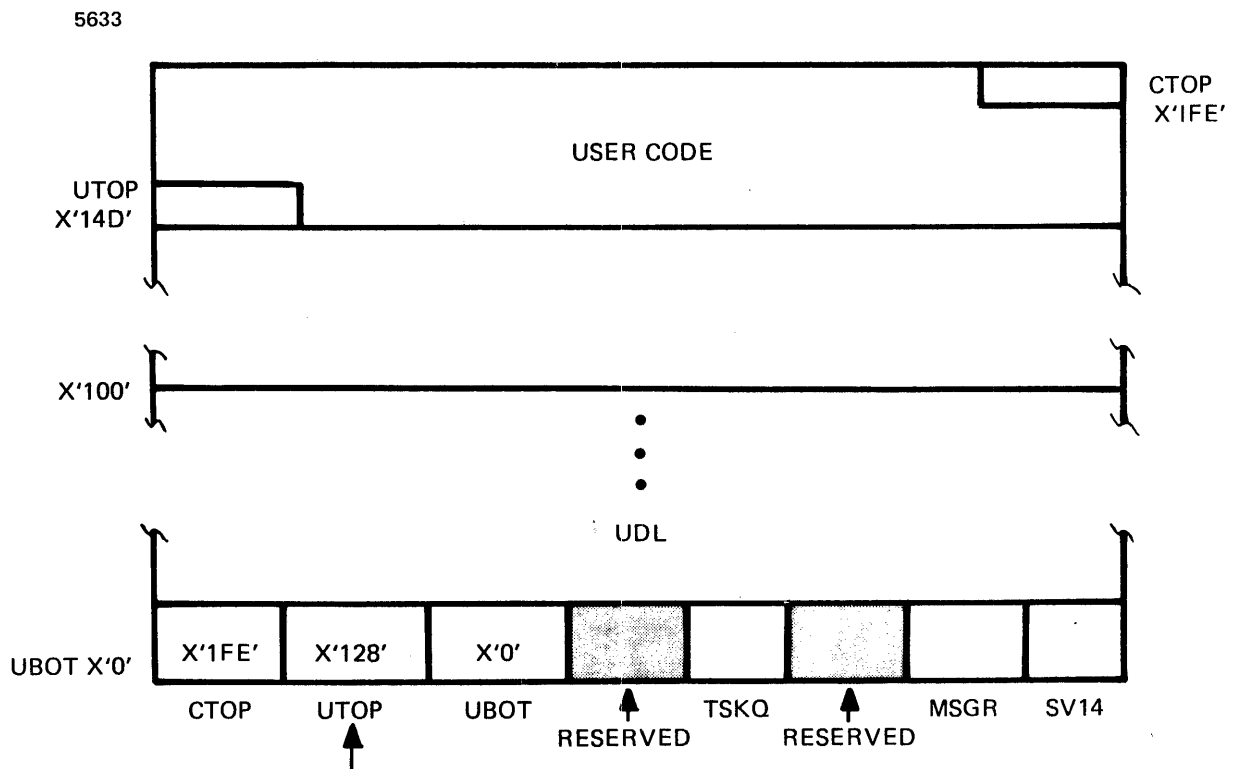
User register is a 1-byte field that must contain a decimal number from 0 through 15 indicating the register that receives the UDL starting address.

When executing this call, the UDL starting address, which is loaded into the user-specified register, varies for u-tasks and executive tasks (e-task). 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.

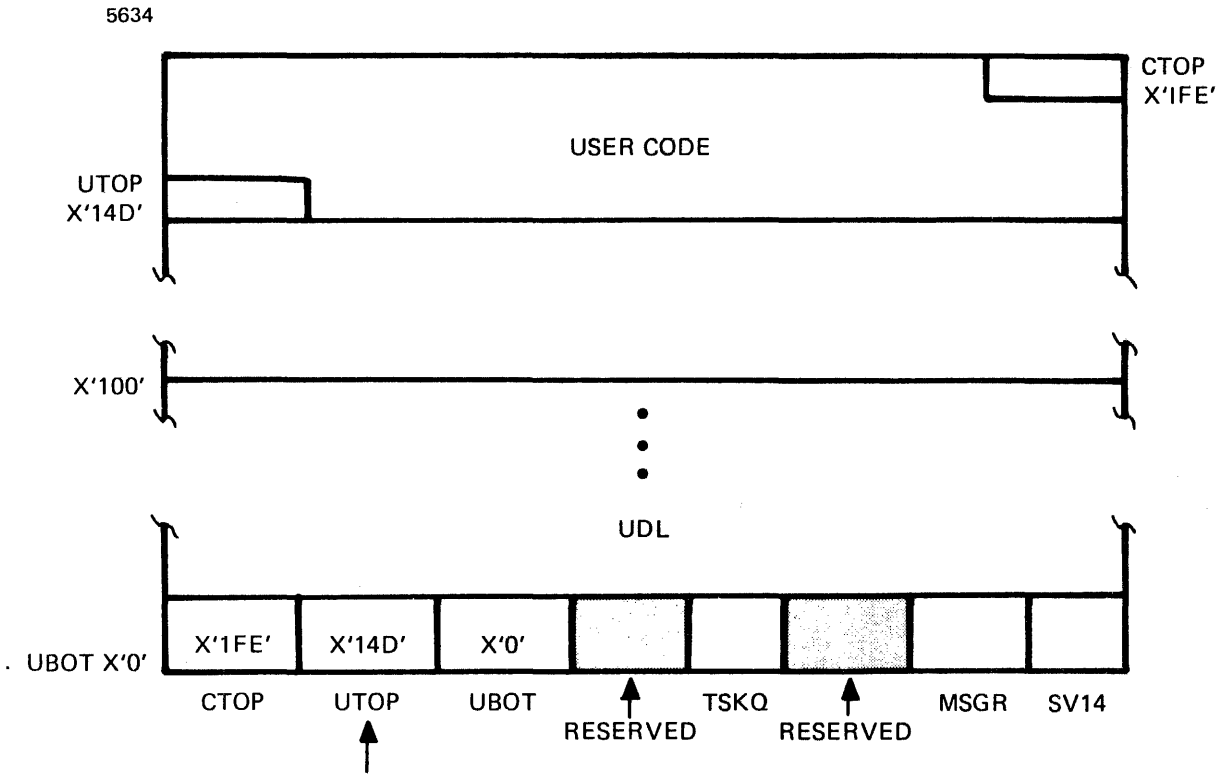
If the user modified the UDL by changing address pointers or if UTOP was changed by a GET or RELEASE STORAGE, the contents of CTOP, UTOP, and UBOT in the UDL might not be valid. SVC 2 code 5 restores this data to a valid state by storing the current values of CTOP, UTOP, and UBOT into the UDL.

Example:

UDL after execution of SVC 2 code 2 and before execution of SVC 2 code 5



UDL after execution of SVC 2 code 5



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

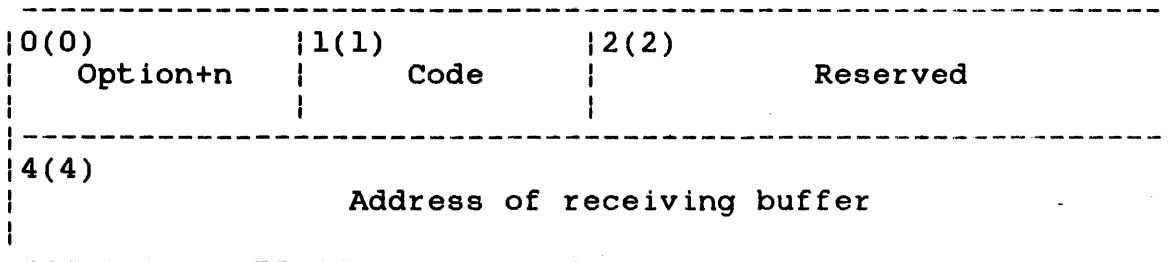

```

-----
| SVC 2 |
| CODE 6 |
-----

```

3.7 SVC 2 CODE 6: CONVERT BINARY NUMBER TO ASCII HEXADECIMAL OR ASCII DECIMAL

SVC 2 code 6 converts an unsigned 32-bit binary number located in the user register 0 to an ASCII hexadecimal number or an ASCII decimal number. This is accomplished through the SVC 2 code 6 parameter block shown in Figure 3-10.



```

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

```

Figure 3-10 SVC 2 Code 6 Parameter Block Format and Coding

This parameter block is 8 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 through 63 indicating the number of bytes in the buffer specified in the SVC 2 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, suppressing leading zeros.

Code is a 1-byte field that must contain the decimal number 6 to indicate code 6 of SVC 2.

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 any byte can be located on any byte boundary.

The receiving buffer should be defined to receive the largest number, which is $4,294,967,295$ ($2^{32} - 1$), 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 $2^{32} - 1$, 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 leftmost significant digits filled with ASCII zeros. However, if the converted number is longer than the buffer, the leftmost digits of the converted number are lost. If suppression of leading zeros is requested, the leftmost zeros in the receiving buffer are filled with spaces (hexadecimal 20).

3.7.1 SVC 2 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 leftmost 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 SVC 2 code 6

|0 0| 0 0|2 1|4 8| Hex

Receiving buffer after execution of SVC 2 code 6

Zero filled

|3 0| 3 0|3 0|3 0|3 2|3 1|3 4|3 8| Hex

3.7.2 SVC 2 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 leftmost significant digits filled with ASCII spaces (hexadecimal 20).

3.7.3 SVC 2 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 leftmost 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 SVC 2 code 6

|0 0| 0 0|3 F|C 2| Hex

Receiving buffer after execution of SVC 2 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

3.7.4 SVC 2 Code 6, Option X'C0'+n

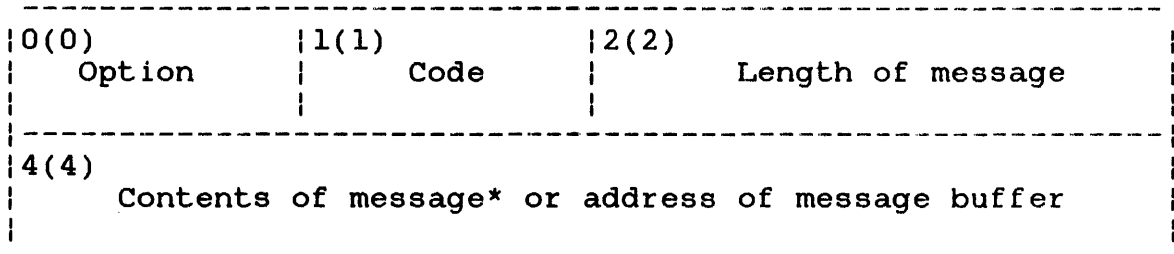
If option X'C0'+n is specified, the unsigned 32-bit binary number located in the user register 0 is converted to an ASCII decimal number. The resulting number is stored right-justified in the receiving buffer with the leftmost significant digits containing zeros filled with ASCII spaces (hexadecimal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

3.8 SVC 2 CODE 7: LOG MESSAGE

SVC 2 code 7 sends a user-specified message to the system console, user terminal, or user-specified log device, depending on task environment. This is accomplished through the SVC 2 code 7 parameter block in Figure 3-11. 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
                .
                .
                .
parblk        ALIGN 4
                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 3-11 SVC 2 Code 7 Parameter Block Format and Coding

This parameter block is 8 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 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 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 system console only.

Code is a 1-byte field that must contain the decimal number 7 to indicate code 7 of SVC 2.

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 system generation (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 line feed is automatically appended to the message, and
- the message terminates when the end of the buffer or message is reached or when a carriage return 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 SVC 2 code 7 for each line. However, each line should include a carriage return and line feed at the end. The image options should be used with caution because the amount of time that must remain for a carriage return to occur differs on various console devices.

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

Contents of message buffer before and after execution of SVC 2 code 7

```
-----  
|4F|50|45|52|0|45|52||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 |O |2 |8 | |O |N | |M |A |G |I |  
-----
```

Log device after execution of SVC 2 code 7

OPERATOR-PLS MOUNT TP028 ON MAG1

3.8.2 SVC 2 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.

3.8.3 SVC 2 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.

3.8.4 SVC 2 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.

3.8.5 SVC 2 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 TPO28 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 SVC 2 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|O|2|8| |O|N| |M|A|G|1|
```

Contents of message buffer before and after execution of second SVC 2 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 SVC 2 code 7

SET TAPE AT 800 BPI TPO28 ON MAG1

(no line feed appended, message overwritten)

3.8.6 SVC 2 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.

3.8.7 SVC 2 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.

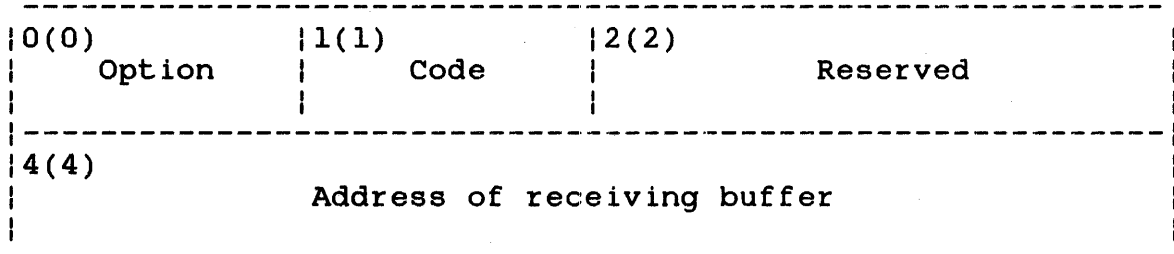
3.8.8 SVC 2 Code 7, Option X'E0'

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.

3.9 SVC 2 CODE 8: INTERROGATE CLOCK

SVC 2 code 8 sends the current time of day to a user-specified buffer. This is accomplished through the SVC 2 code 8 parameter block shown in Figure 3-12.



```

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

```

Figure 3-12 SVC 2 Code 8 Parameter Block Format and Coding

This parameter block is 8 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 as three fullwords of ASCII data in the form: hh:mm:ss.sss
- Option X'C0'-returns the time 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 code 8 of SVC 2.

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.

3.9.1 SVC 2 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 8-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 SVC 2 code 8 option X'00' |
when current time of day is 10:09:03.

```
-----  
| 3 1| 3 0| 3 A| 3 0| 3 9| 3 A| 3 0| 3 3| Hex  
|-----  
| 1 | 0 | : | 0 | 9 | : | 0 | 3 |  
|-----  
~~~~~                 ~~~~~                 ~~~~~  
hh                    mm                    ss
```

3.9.2 SVC 2 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 |
4 bytes long and aligned on a fullword boundary. |

Example:

Contents of buffer after execution of SVC 2 code 8 option X'80' |
when current time of day is 10:13:48

```
-----  
| 0 0| 0 0| 8 F| D C| Hex  
|-----  
~~~~~  
36828 = 10:13:48  
  (decimal)
```

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

3.9.3 SVC 2 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 SVC 2 code option X'40', when 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	
hh		mm		ss		sss						

3.9.4 SVC 2 Code 8, Option X'CO

If option X'CO' 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 8 bytes long and aligned on a fullword boundary.

Example:

Contents of buffer after execution of SVC 2 Code 8 option X'CO' when current time of day in ASCII is 10:41:32.8

| 00 | 00 | 96 | 5C | 00 | 00 | 03 | 20 | Hex

 ss sss

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

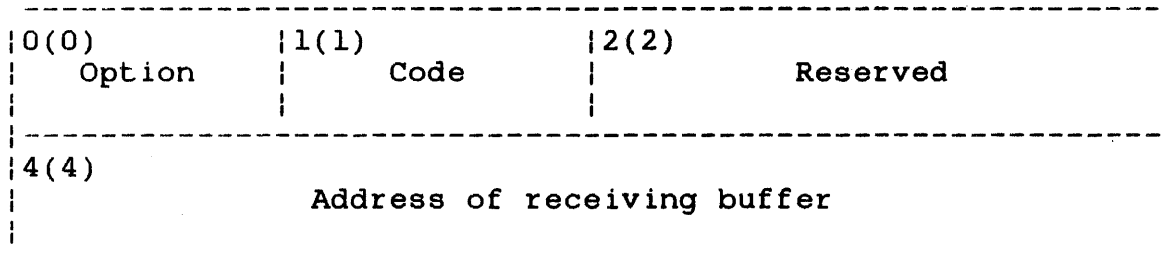
```

-----
| SVC 2 |
| CODE 9 |
-----

```

3.10 SVC 2 CODE 9: FETCH DATE

SVC 2 code 9 sends the current date to a user-specified buffer. This is accomplished through the SVC 2 code 9 parameter block shown in Figure 3-13.



```

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

```

Figure 3-13 SVC 2 Code 9 Parameter Block Format and Coding

This parameter block is 8 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 code 9 of SVC 2.

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 8 bytes long and located in a task writable segment. The buffer can be located on any boundary.

SVC 2 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 time.

Example:

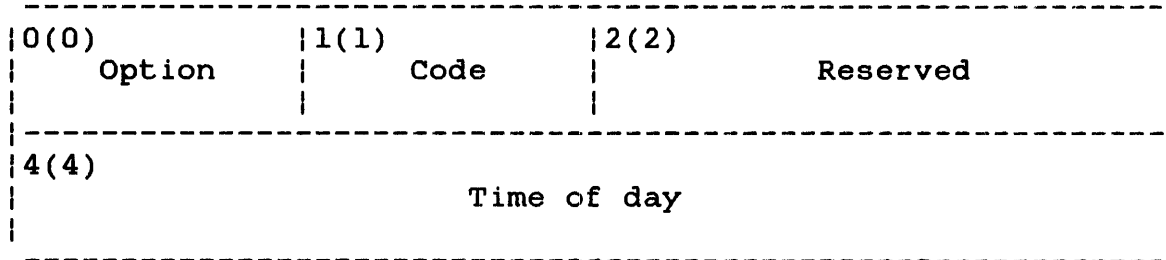
```
                SVC  2,DATE
                SVC  2,PAUSE
                .
                .
                .
DATE            ALIGN 4
                DB    0,9
                DC    H'0'
                DCF   A(BUF)
                ALIGN 4
PAUSE          DB    0,1
```

Contents of receiving buffer after execution of SVC 2 code 9
when the current date in ASCII is 07/06/81

```
-----
| 3 0 | 3 7 | 2 F | 3 0 | 3 6 | 2 F | 3 8 | 3 1 | Hex
|-----|
| 0 | 7 | / | 0 | 6 | / | 8 | 1 | ASCII
|-----|
  mm      dd      yy
```

3.11 SVC 2 CODE 10: TIME OF DAY WAIT

SVC 2 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 SVC 2 code 10 parameter block shown in Figure 3-14.



```

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

```

Figure 3-14 SVC 2 Code 10 Parameter Block Format and Coding

The SVC 2 code 10 parameter block is 8 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 code 10 of SVC 2.
- 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 through 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
2nd	172,800	259,199
4th	259,200	345,599
4th	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 SVC 2 code 10, the condition code is set to indicate if the call was successful. The possible condition code settings are:

Condition codes

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 condition code 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'
                ALIGN 4
PAUSE         DB   0, 1
    
```

Equal to 03:22:45 A.M.

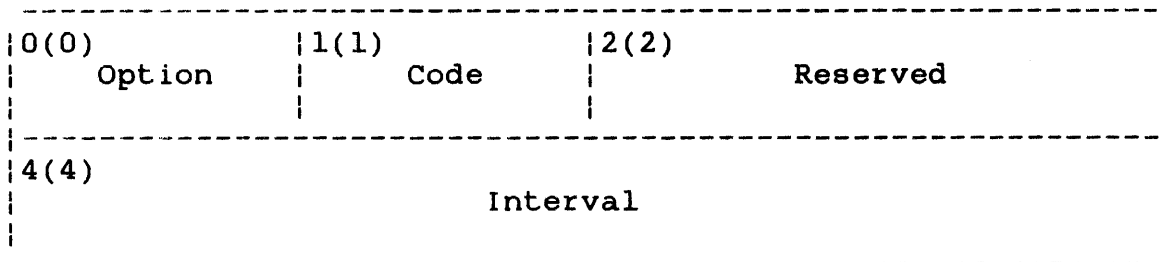
```

-----
|  SVC 2  |
| CODE 11 |
-----

```

3.12 SVC 2 CODE 11: INTERVAL WAIT

SVC 2 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 SVC 2 code 11 parameter block shown in Figure 3-15.



```

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

```

Figure 3-15 SVC 2 Code 11 Parameter Block Format and Coding

This parameter block is 8 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 code 11 of SVC 2.

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 through 268,435,455 ($2^{28} - 1$) 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 SVC 2 code 11, the condition code is set to indicate if the call was successful. The possible condition codes are:

Condition codes

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 condition code 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'
                ALIGN 4
                PAUSE DB    0, 1
                Equal to 32.768 seconds
```

```

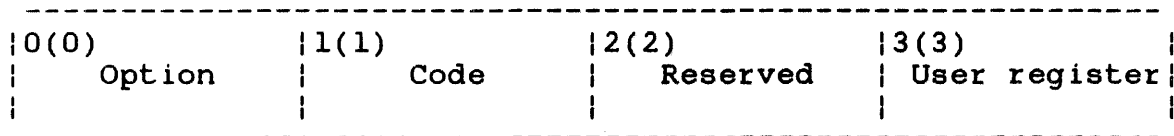
-----
|   SVC 2   |
| CODE 15   |
-----

```

3.13 SVC 2 CODE 15: CONVERT ASCII HEXADECIMAL OR ASCII DECIMAL TO BINARY

SVC 2 code 15, the inverse of SVC 2 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 lowercase.

The result is stored in the user register 0. This is accomplished through the SVC 2 code 15 parameter block shown in Figure 3-16.



```

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

```

Figure 3-16 SVC 2 Code 15 Parameter Block Format and Coding

This parameter block is 4 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, skip leading spaces.

- Option X'80' converts ASCII decimal to binary.
- Option X'C0' converts ASCII decimal to binary, skip leading spaces.

Code is a 1-byte field that must contain the decimal number 15 to denote code 15 of SVC 2.

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 SVC 2 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 condition code to be set to 0. The possible condition code settings that can occur after executing SVC 2 code 15 are:

Condition codes

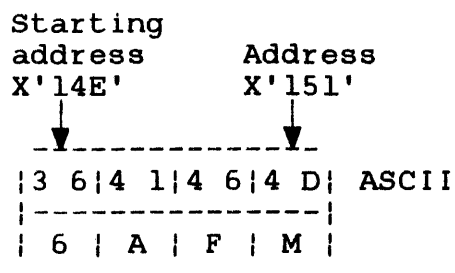
C	V	G	L	
0	0	0	0	Normal termination
0	0	0	1	No numbers converted; register 0 contains zeros.
0	1	0	0	Value of the number to be converted is greater than 2,147,483,647 ($2^{31} - 1$).

3.13.1 SVC 2 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 leftmost significant bits filled with zeros.

Example:

Buffer before and after execution of SVC 2 code 15



Register 0 after execution of SVC 2 code 15

| 0 0 | 0 0 | 0 6 | A F |

User-specified register before execution of SVC 2 code 15

| 0 0 | 0 0 | 0 1 | 4 E | Hex

User-specified register after execution of SVC 2 code 15

| 0 0 | 0 0 | 0 1 | 5 1 | Hex

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 condition code 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 leftmost significant bits truncated, and the condition code is set to 4 (V bit set).

Example:

Buffer before and after execution of SVC 2 code 15

```
Starting address          Address
X'152'                   X'15C'
  ↓                       ↓
-----
| 3 2|3 1|3 4|3 7|3 4|3 8|3 3|3 6|3 6|3 5|2 0| ASCII
-----
| 2  | 1  | 4  | 7  | 4  | 8  | 3  | 6  | 6  | 5  |
-----
```

User-specified register before execution of SVC 2 code 15

```
-----
|0 0|0 0|0 1|5 2| Hex
-----
```

```
Overflow                Register 0 after execution of SVC 2 code 15
  21                    -----
                        |4 7|4 8|3 6|6 5| ASCII
                        -----
```

User-specified register after execution of SVC 2 code 15

```
-----
|0 0|0 0|0 1|5 C| Hex
-----
```

Condition code

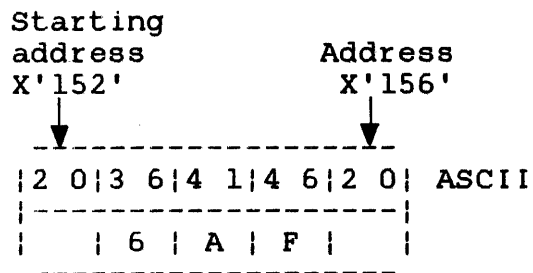
```
-----
| C | V | G | L |
-----
| 0 | 1 | 0 | 0 |  ASCII number greater than 231 -1
-----
```

3.13.2 SVC 2 Code 15, Option X'40'

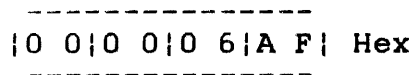
If option X'40' is specified, the ASCII encoded hexadecimal number in the buffer, specified by the address in the user register, is converted to a binary number with leading spaces ignored during the conversion. The resulting number is stored right-justified in register 0 with the leftmost significant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

Example:

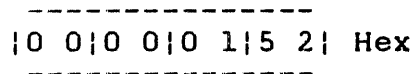
Buffer before and after execution of SVC 2 code 15



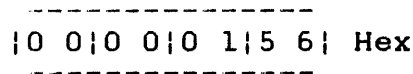
Register 0 after execution of SVC 2 code 15



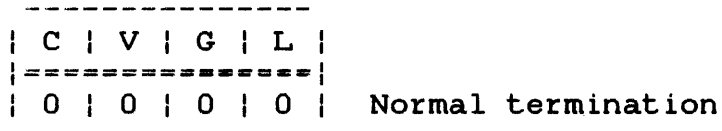
User-specified register before execution of SVC 2 code 15



User-specified register after execution of SVC 2 code 15



Condition code



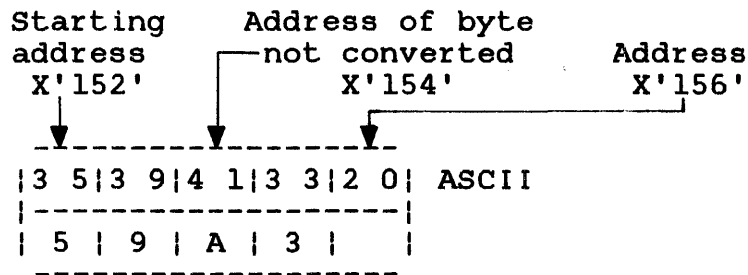
| 3.13.3 SVC 2 Code 15, Option X'80'

| 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 leftmost
| 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 SVC 2 code 15



Register 0 after execution of SVC 2 code 15

| 0 0 | 0 0 | 0 0 | 3 B | Hex

User-specified register before execution of SVC 2 code 15

| 0 0 | 0 0 | 0 1 | 5 2 | Hex

User-specified register after execution of SVC 2 code 15

| 0 0 | 0 0 | 0 1 | 5 4 | Hex

Condition codes

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 condition code is set to 1.

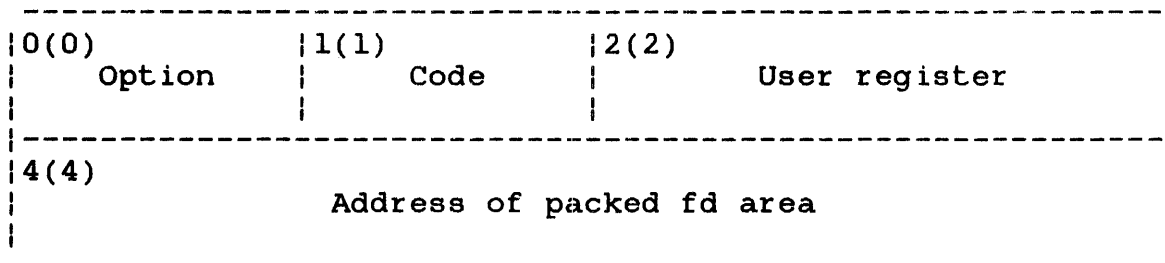
If the value of the ASCII decimal number is greater than 2,147,483,647 ($2^{31} - 1$), the number is converted, the resulting binary number is stored right-justified in register 0 with the leftmost significant bits truncated, and the condition code is set to 4 (V bit set).

3.13.4 SVC 2 Code 15, Option X'CO'

If option X'CO' 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 leftmost significant bits filled with zeros.

3.14 SVC 2 CODE 16: PACK FILE DESCRIPTOR

SVC 2 code 16 formats a user-specified unpacked file descriptor (fd) to the packed format used within the SVC 7 parameter block (see bytes 8 through 23 of the SVC 7 parameter block). Figure 3-17 illustrates the SVC 2 code 16 parameter block format.



```

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

```

Figure 3-17 SVC 2 Code 16 Parameter Block Format and Coding

This parameter block is 8 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	MEANING
X'00'	Default volume is the user volume.
X'40'	Default volume is the user volume. Skip leading spaces.
X'10'	Default volume is the system volume.
X'50'	Default volume is the system volume. Skip leading spaces.
X'20'	Default volume is the spool volume.
X'60'	Default volume is the spool volume. Skip leading spaces.
X'80'	No default volume.
X'C0'	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 non-privileged u-tasks only. These options pack file descriptors 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 will have an account number in its file class field. See Section 3.14.9 for the SVC 2 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 code 16 of SVC 2.

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 condition codes set after packing an fd are:

Condition codes

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 condition codes 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 condition code is set to 8 (C bit set).

All lowercase characters in the user-specified fd are converted to their equivalent uppercase 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 (uppercase)
- a through z (lowercase)
- 0 through 9 (numerics)
- selected special characters (symbols)

The format of the user-specified fd is:

Format:

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

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 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 through 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 3-18. Since this area is identical to bytes 8 through 23 of the SVC 7 parameter block, these bytes can be designated as the receiving area.

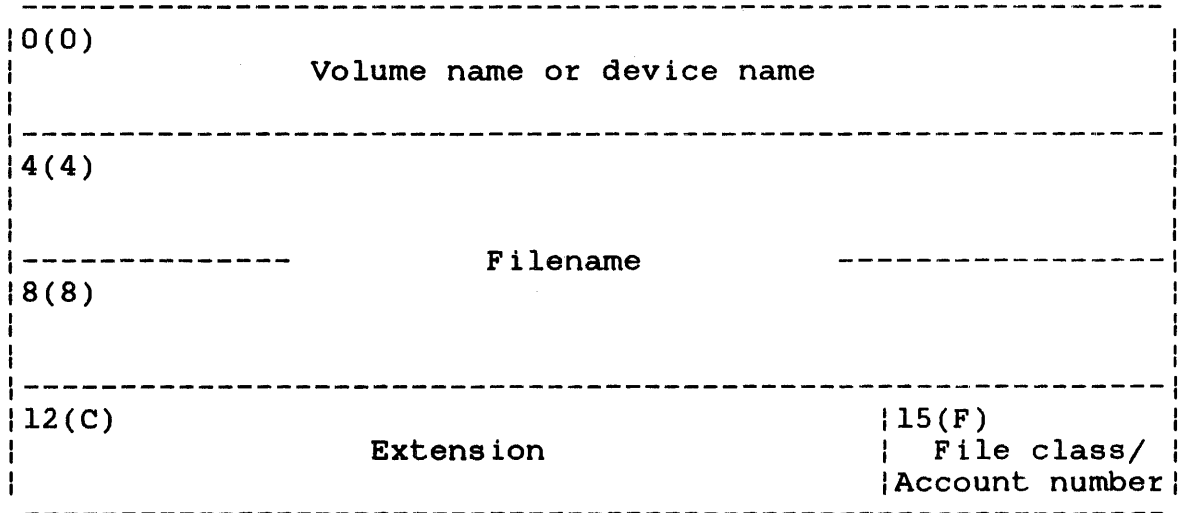


Figure 3-18 Packed File Descriptor Area

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 4 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 8 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 3 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 operating system. P is returned in the class field of the packed fd when running under MTM.

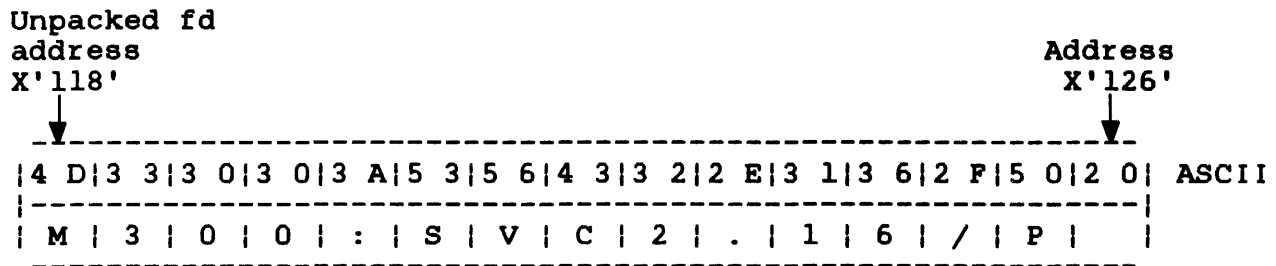
NOTE

If the SVC 2 code 16 options for privileged tasks are used, an account number is returned to this field. (See Section 3.14.9.)

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 SVC 2 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																									

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:

Unpacked fd
address
X'118'

Address
X'11D'

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

User-specified register before packing fd

|00|00|01|18| Hex

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	ASCII
M	3	0	0																											

User-specified register after packing fd

|00|00|01|18| Hex

Condition code

C	V	G	L
1	1	0	0

No extension present in unpacked fd;
syntax error present in unpacked fd

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

| Example 3:

Unpacked fd
 address
 X'118'

Address
 X'11F'

```

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

```

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|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|2 0|5 0| ASCII
-----
| M | 3 | 0 | 0 | S | V |   |   |   |   |   |   |   |   |   |
-----

```

User-specified register after packing fd

```

-----
| 00|00|01|18| Hex
-----

```

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

```

-----
| 4 3|5 2|4 4|5 2|2 E| ASCII
|-----|
| C | R | D | R | . |
|-----|

```

```

-----
| 4 0|3 3|3 0|3 0|4 3|5 2|4 4|5 2|2 0|2 0|2 0|2 0|2 0|2 0|2 0|5 0| ASCII
|-----|
| M | 3 | 0 | 0 | C | R | D | R | | | | | | | | P |
|-----|

```

Condition code

```

-----
| C | V | G | L |
|-----|
| 0 | 0 | 0 | 1 | No volume name present in unpacked format
|-----|

```

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

Example 5:

Unpacked fd

```

-----
| 5 0|4 3|4 2|3 3|3 2|2 F|5 3| ASCII
|-----|
| P | C | B | 3 | 2 | / | S |
|-----|

```

Packed fd

```

-----
| 4 D|3 3|3 0|3 0|5 0|4 3|4 2|3 3|3 2|2 0|2 0|2 0|2 0|2 0|2 0|5 3| ASCII
|-----|
| M | 3 | 0 | 0 | P | C | B | 3 | 2 | | | | | | | S |
|-----|

```

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 condition code to determine if a syntax error occurred.

3.14.1 SVC 2 Code 16, Option X'00'

If option X'00' 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 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

```
-----
| 2 0|4 D|3 3|3 0|3 1|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
|-----|
|   | M | 3 | 0 | 1 | C | H | 2 | P | R | M | . | 6 | 1 | 3 | / | G |
|-----|
```


Packed fd

```

-----
| 4 D|3 6|3 7|4 1|2 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|2 0|2 0|2 0| ASCII
-----
| M | 6 | 7 | A |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
-----

```

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.

3.14.2 SVC 2 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 system volume.

Example:

Unpacked fd

```

-----
| 2 0|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  
-----
```

| 3.14.3 SVC 2 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

```
-----  
| 5 3|5 6|4 3|3 2|2 E|3 1|3 6|2 F|5 0| 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|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.

3.14.4 SVC 2 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|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  
-----
```

| 3.14.5 SVC 2 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

```
-----  
| 5 3|5 6|4 3|5 4|5 3|5 4|2 E|4 3|4 1|4 C| ASCII  
|-----|  
| S | V | C | T | S | T | . | C | A | L |  
|-----|
```

Packed fd

```
-----  
| 5 3|3 3|3 0|3 0|5 3|5 6|4 3|5 4|5 3|5 4|2 0|2 0|4 3|4 1|4 C|5 0| ASCII  
|-----|  
| S | 3 | 0 | 0 | S | V | C | T | S | T | | | C | A | L | P |  
|-----|
```

| Condition code

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

| Example 2:

Unpacked fd

```
-----  
| 2 0|2 0|5 3|5 6|4 3|5 4|5 3|5 4|2 E|4 3|4 1|4 C| ASCII  
|-----|  
| | | S | V | C | T | S | T | . | C | A | L |  
|-----|
```

Packed fd

```

-----
| 5 3|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|2 0|2 0|2 0| ASCII
|-----|
| S | 3 | 0 | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-----|

```

Condition code

```

-----
| C | V | G | L |
|-----| No extension present in unpacked fd;
| 1 | 1 | 0 | 1 | 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.

3.14.6 SVC 2 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

```
-----  
| 2 0|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

```
-----  
| 4 D|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

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.

| 3.14.7 SVC 2 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

```

-----
| 4 F|5 0|5 4|3 8|3 0|2 E|4 3|4 1|4 C| ASCII
-----
| O | P | T | 8 | 0 | . | C | A | L |
-----

```

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
-----
| O | O | O | O | 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
-----
| O | O | O | O | 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 | O | 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

```
-----  
5 3|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|2 0| ASCII  
-----  
S | 3 | 0 | 0 |   |   |   |   |   |   |   |   |   |   |   |   |  
-----
```

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

```
-----  
2 0|4 F|5 0|5 4|3 8|3 0|2 E|4 3|4 1|4 C| ASCII  
-----  
  | O | P | T | 8 | 0 | . | C | A | L |  
-----
```

Packed fd location contents before pack fd operation

```
-----  
4 0|3 2|3 5|2 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  
-----  
M | 2 | 5 |   | W | H | A | T | E | V | E | R | E | L | S | E |  
-----
```

Packed fd after pack fd operation

```
-----  
4 D|3 2|3 5|2 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|2 0| ASCII  
-----  
M | 2 | 5 |   |   |   |   |   |   |   |   |   |   |   |   |  
-----
```


Condition code

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

3.14.8 SVC 2 Code 16, Option X'CO'

If option X'CO' 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
|-----
```

3.14.9 SVC 2 Code 16 Options for Privileged Tasks

Only privileged u-tasks, e-tasks, and privileged 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 0-65,535, 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 SVC 2 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 system volume
X'58'	Default volume is system volume; skip leading spaces
X'28'	Default volume is spool volume
X'68'	Default volume is spool volume; skip leading spaces
X'88'	No default volume
X'C8'	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, SVC 2 code 16 returns an account number to the resulting packed fd and sets the G bit in the condition code.

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).

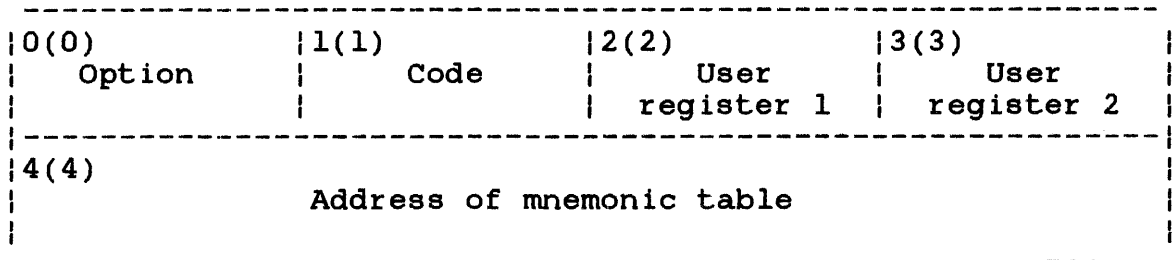
```

-----
|  SVC 2  |
| CODE 17 |
-----

```

| 3.15 SVC 2 CODE 17: SCAN MNEMONIC TABLE

SVC 2 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 SVC 2 code 17 parameter block is shown in Figure 3-19.



```

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

```

| Figure 3-19 SVC 2 Code 17 Parameter Block Format and Coding

This parameter block is 8 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 code 17 of SVC 2.
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 SVC 2 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 through 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 must	is a 4-byte field that must contain the starting address of the mnemonic table. This must be defined before executing SVC 2 code 17.

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

- A through Z (uppercase)
- a through z (lowercase)
- 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 lowercase characters that appear in the user-specified mnemonic character string are accepted as their uppercase equivalent.

3.15.1 Building a Mnemonic Table

The mnemonic table to be used in SVC 2 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 (uppercase alphabets)
- 0 through 9 (numerics 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; whereas, a bit setting of 0 indicates that it is not a required character.

3.15.2 Executing SVC 2 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 user register 1 specified in the parameter block.

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

Condition codes

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 SVC 2 code 17

Starting address
X'158'

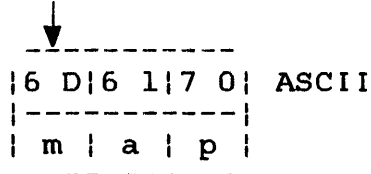


Table (after assembly) before and after execution of SVC 2 code 17.

```

-----
| 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 SVC 2 code 17

```

-----
|0 0|0 0|0 1|5 8| Hex
-----

```

User register 1 after execution of SVC 2 code 17

```

-----
|0 0|0 0|0 1|5 B| Hex
-----

```

User register 2 after execution of SVC 2 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 condition code 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 SVC 2 code 17

Starting
address
X'158'

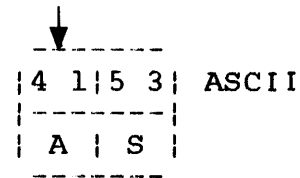
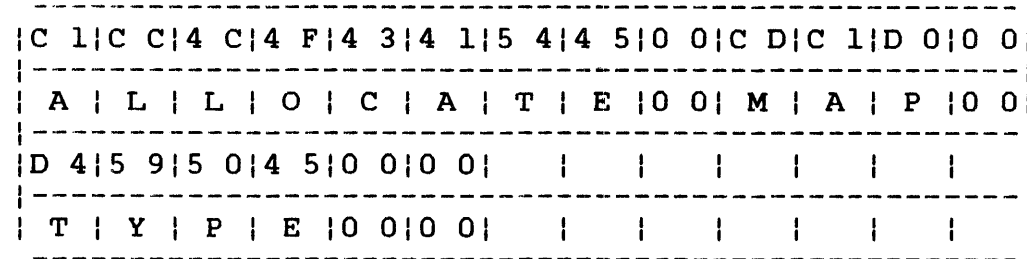
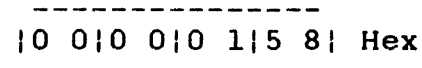


Table (after assembly) before and after execution of SVC 2 code 17



User register 1 before and after execution of SVC 2 code 17



User register 2 after execution of SVC 2 code 17

| F F | F F | F F | F F | Hex

Condition code

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

| 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 SVC 2 code 17

Starting
address

X'158' X'15A'

```
  ↓           ↓
-----
| 5 4|5 9|2 6|4 5| ASCII
-----
| T | Y | & | E |
-----
```

Table (after assembly) before and after execution of SVC 2 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 SVC 2 code 17

```
-----
| 0 0|0 0|0 1|5 8| Hex
-----
```

User register 1 after execution of SVC 2 code 17

```
-----
| 0 0|0 0|0 1|5 A| Hex
-----
```

User register 2 after execution of SVC 2 code 17

```
-----
| 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 condition code is set to 0.

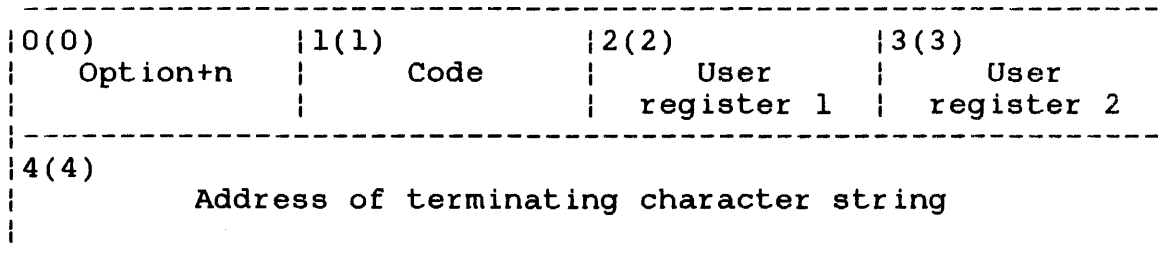
```

-----
|  SVC 2  |
| CODE 18 |
-----

```

| 3.16 SVC 2 CODE 18: MOVE ASCII CHARACTERS

SVC 2 code 18 moves a specified number of ASCII characters from a sending buffer to a receiving buffer in memory. The SVC 2 code 18 parameter block is shown in Figure 3-20.



```

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

```

| Figure 3-20 SVC 2 Code 18 Parameter Block Format and Coding

This parameter block is 8 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 through 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 code 18 of SVC 2.
- 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 SVC 2 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 SVC 2 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 SVC 2 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 condition code is also set after executing SVC 2 code 18. The possible condition code settings are:

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

3.16.1 SVC 2 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 SVC 2 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 condition code 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 SVC 2 code 18

```

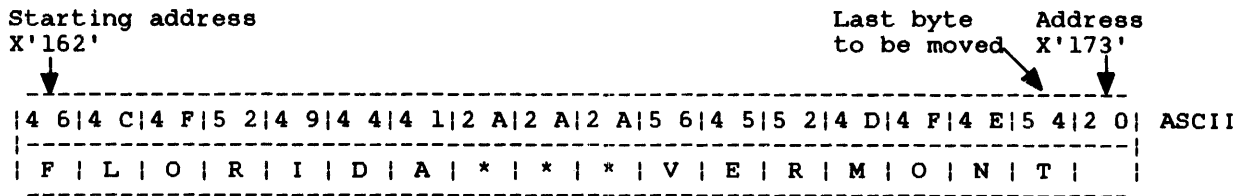
-----
|0 0|0 0|0 1|6 2| Hex
-----

```

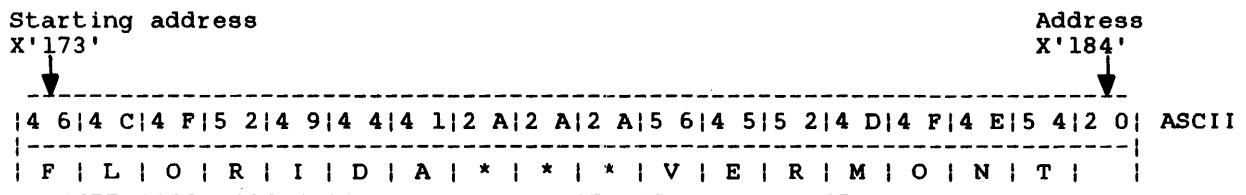
User register 2 before execution of SVC 2 code 18

| 0 0 | 0 0 | 0 1 | 7 3 | Hex

ASCII character string before and after execution of SVC 2 code 18



Receiving buffer after execution of SVC 2 code 18



User register 1 after execution of SVC 2 code 18

| 0 0 | 0 0 | 0 1 | 7 3 | Hex

User register 2 after execution of SVC 2 code 18

| 0 0 | 0 0 | 0 1 | 8 4 | Hex

Condition code

C	V	G	L
=====			
0	0	0	0

| 3.16.2 SVC 2 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 condition code 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.

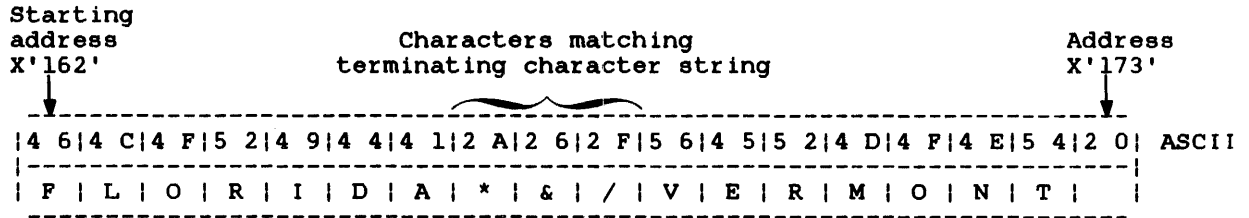
Example:

```

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 SVC 2 code 18



User register 1 before execution of SVC 2 code 18

```

-----
|0 0|0 0|0 1|6 2| Hex
-----

```

User register 2 before execution of SVC 2 code 18

```

-----
|0 0|0 0|0 1|7 3| Hex
-----

```

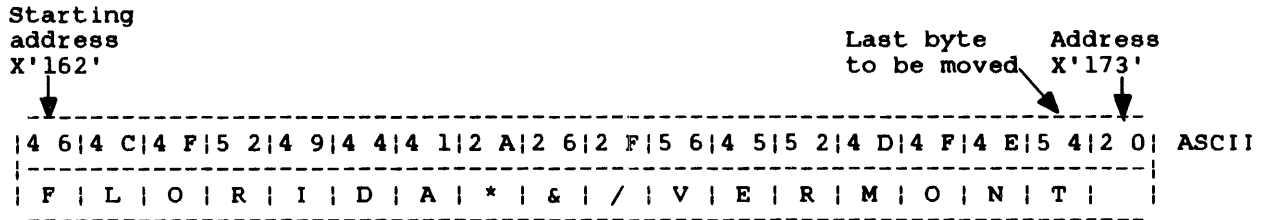

Example:

```

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 SVC 2 code 18



User register 1 before execution of SVC 2 code 18

```

-----
| 0 0 | 0 0 | 0 1 | 6 2 | Hex
-----

```

User register 2 before execution of SVC 2 code 18

```

-----
| 0 0 | 0 0 | 0 1 | 7 3 | Hex
-----

```

Receiving buffer after execution of SVC code 18

Starting

address

X'173'

Address

X'184'

| 4 6 | 4 C | 4 F | 5 2 | 4 9 | 4 4 | 4 1 | 2 A | 2 6 | 2 F | 5 6 | 4 5 | 5 2 | 4 D | 4 F | 4 E | 5 4 | 2 0 | ASCII
| F | L | O | R | I | D | A | * | & | / | V | E | R | M | O | N | T |

User register 1 after execution of SVC 2 code 18

| 0 0 | 0 0 | 0 1 | 7 4 | Hex

User register 2 after execution of SVC 2 code 18

| 0 0 | 0 0 | 0 1 | 8 4 | Hex

Terminating character string

| 2 7 | 2 4 | 3 A | ASCII

| , | \$ | : |

Condition code

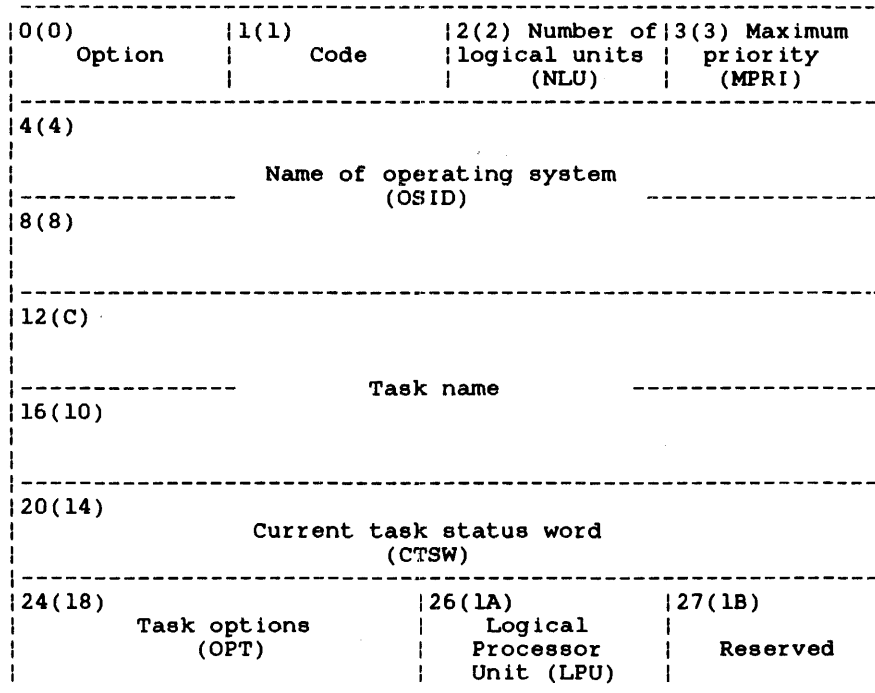
| C | V | G | L |
| 0 | 1 | 0 | 0 |

3.17 SVC 2 CODE 19: PEEK

SVC 2 code 19 provides four 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 task control block (TCB). Figures 3-21 through 3-25 illustrate the five peek parameter block option formats.

3.17.1 Parameter Block for Option X'00'

If SVC 2 code 19 is executed with option X'00' specified in the parameter block option field, use the parameter block format in Figure 3-21. This option is used to obtain task information.



```

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

```

Figure 3-21 SVC 2 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 code 19 of SVC 2.
Number of logical units (NLU)	is a 1-byte field that receives from the TCB the maximum logical unit number which can be assigned to a task. This hexadecimal number ranges from 0 through 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') through 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 SVC 2 code 19. Table 3-4 lists task options.
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' through X'09'; X'00' indicates the central processing unit (CPU).
Reserved	is a reserved 2-byte field that must contain zeros.

TABLE 3-3 TASK OPTIONS FROM THE TASK CONTROL BLOCK

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
0	Diagnostic task (d-task) (Y'8000 0000')	0 = task determined by bit 16 0 = task is a d-task
1	APU only (Y'4000 0000')	0 = task can run on CPU or APU 1 = task cannot run on CPU
2	APU mapping option (Y'2000 0000')	0 = no APU mapping allowed 1 = task can perform APU mapping functions
3	APU control option (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'04000000')	0 = MTM prompts disabled 1 = MTM prompts enabled
6	Vertical forms control (Y'02000000')	0 = except where specified, all I/O interpreted without forms control 1 = all I/O interpreted with vertical forms control
7	Extended SVC 1 parameter block (Y'01000000')	0 = SVC 1 extended parameter block not used (excludes communications I/O). 1 = extended SVC 1 parameter block used
8	Task event service (Y'00800000')	0 = new TSW for task event service 1 = no new TSW for task event service
9	Task event registers save (Y'00400000')	0 = all register contents saved and restored 1 = only contents of registers that contains task event data are saved and re-stored
10	Task event register save (Y'00200000')	0 = task event register not saved 1 = task event register saved

TABLE 3-3 TASK OPTIONS FROM THE TASK CONTROL BLOCK (Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
11	System group (Y'00100000')	0 = not in system group 1 = in system group
12	Console I/O intercept (Y'00080000')	0 = no console I/O interrupt 1 = console I/O interrupt enable (MTM)
13	Universal status report (Y'00040000')	0 = universal task status reports not allowed 1 = universal task status reports allowed
14	Executive task load (Y'00020000')	0 = allow e-task load 1 = prevent e-task load
15	Queued I/O (Y'00010000')	0 = queued I/O not purged on error 1 = queued I/O purged on error.
16	Executive task (Y'80000000')	0 = task is a u-task 1 = task is an e-task
17	Arithmetic fault (Y'40000000')	0 = task abnormally terminates on arithmetic fault 1 = task continues execution on arithmetic fault
18	Single precision floating point (Y'20000000')	0 = task does not support single precision floating point 1 = task does support single precision floating point
19	Memory resident (Y'10000000')	0 = task is nonresident 1 = task is resident in memory
20	SVC 6 control functions (Y'08000000')	0 = task can execute all SVC 6 control functions 1 = all SVC 6 control functions are prevented
21	SVC 6 communication functions (Y'04000000')	0 = task can execute all SVC 6 communication functions 1 = all SVC 6 communication functions are prevented

TABLE 3-3 TASK OPTIONS FROM THE TASK CONTROL BLOCK (Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
22	Illegal SVC 6 (Y'02000000')	0 = task abnormally terminates on an illegal SVC 6 1 = task continues execution on an illegal SVC 6
23	Double precision floating point (Y'01000000')	0 = task does not support double precision floating point 1 = task does support double precision floating point
24	Rollable (Y'00800000')	0 = task is not rollable 1 = task is rollable
25	Overlays (Y'00400000')	0 = task does not support the use of overlays 1 = task does support the use of overlays
26	Accounting facility (Y'00200000')	0 = disable accounting facility 1 = enable accounting facility
27	Intercept calls (Y'04000000')	0 = task cannot issue intercept calls 1 = task can issue intercept calls
28	Account number privileges (Y'00080000')	0 = task does not have file account number privileges 1 = task has file account number privileges
29	Bare disk I/O privilege (Y'00040000')	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'00020000')	0 = task is not a universal task 1 = task is a universal task
31	Executive task keys (Y'00010000')	0 = no keys are checked on an assign for an e-task. 1 = keys are checked on an assign for an e-task

Example:

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

Parameter block before execution of SVC 2 code 19

00	13	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

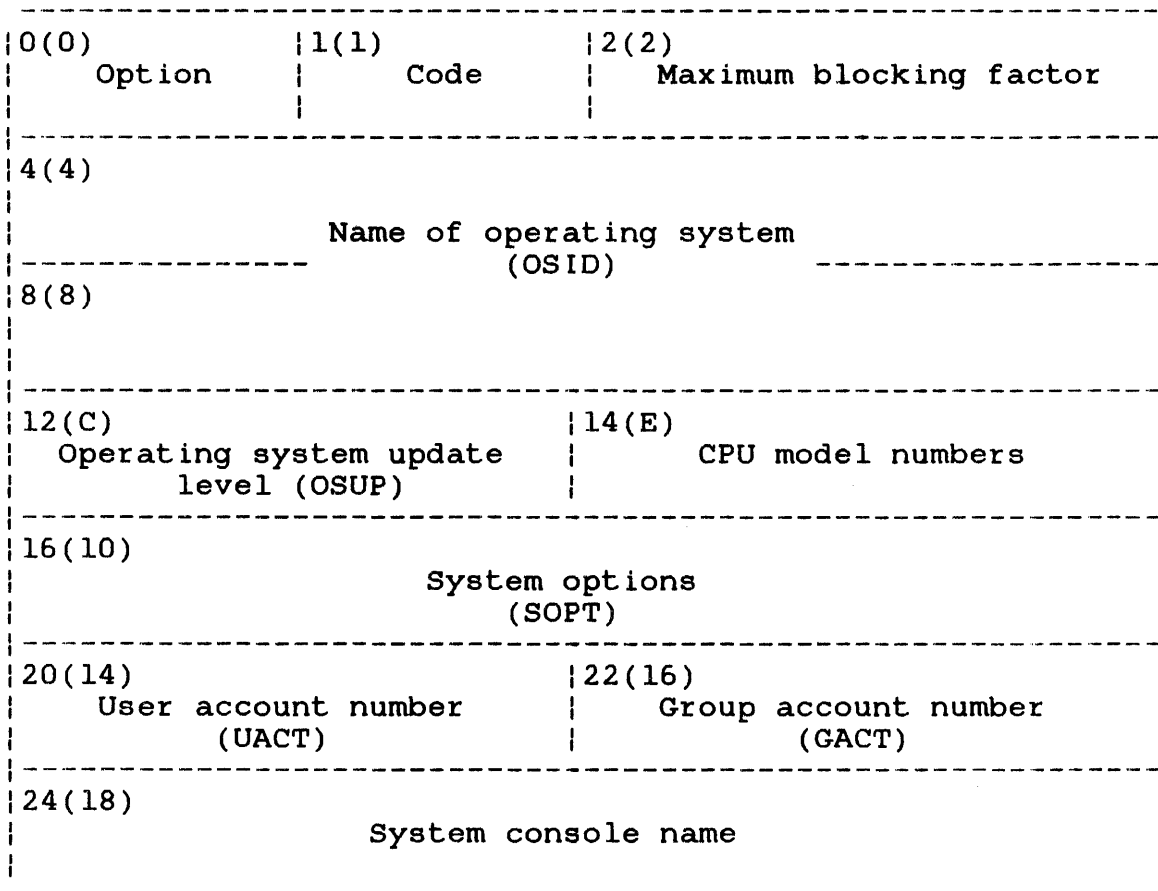
Parameter block after execution of SVC 2 code 19

00	13	0F	81
0	S	3	2
M	T	0	6
M	A	R	
00	00	00	00
0C	84	00	00

- 0C84= SVC 6 control call prevented.
- SVC 6 communication call prevented.
- Task is rollable.
- SVC 6 load of executive task prevented.

3.17.2 Parameter Block for Option X'01'

To execute SVC 2 code 19 with option X'01' specified in the parameter block option field, use the parameter block format in Figure 3-22.



```

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

Figure 3-22 SVC 2 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 code 19 of SVC 2.
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 SVC 7 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 from the SPT the model numbers in hexadecimal of the CPU used in the system. They are: <ul style="list-style-type: none">● Model 7/32 has a value of X'0007'.● Model 8/32 has a value of X'0008'.● Model 3200MPS has a value of X'0C80'● Model 3210 has a value of X'0C8A'● Model 3220 has a value of X'0C94'.● Model 3230 has a value of X'0C9E'.● Model 3240 has a value of X'0CA8'.● Model 3250 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 3-4 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 leftmost 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 leftmost portion.

System console name is a 4-byte field that receives the name of the device that is acting as the system console.

TABLE 3-4 SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
0	Single precision floating point (Y'8000 0000')	0 = system does not support single precision floating point 1 = system does support single precision floating point
1	Form date is displayed (Y'4000 0000')	0 = date is displayed in the form: mmddyy 1 = date is displayed in the form: ddmmyy
2	Time display (Y'2000 0000')	0 = time is displayed on output device specified by the user 1 = time displayed on panel
3	Double precision floating point (Y'1000 0000')	0 = system does not support double precision floating point 1 = system does support double precision floating point
4	Writable control store (WCS) (Y'0800 0000')	0 = system does not support WCS 1 = system does support WCS
5	Address alignment error checking (Y'0400 0000')	0 = hardware does not support address alignment error checking 1 = hardware supports address alignment error checking

TABLE 3-4 SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE
(Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
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 rollin, rollout 1 = system supports rollin, rollout
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
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

TABLE 3-4 SYSTEM OPTIONS FROM THE SYSTEM POINTER TABLE
(Continued)

BIT POSITION	BIT NAME AND MASK	BIT SETTING AND MEANING
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 3210, 3220, 3230, 3240, 3250, 3200MPS processors
19	MAT hardware (Y'0000 1000')	1 = system has MAT hardware 0 = system does not have MAT hardware
20	Single precision floating point traps (Y'0000 0800')	1 = single precision floating point software traps present 0 = single precision floating point software traps not present
21	Double precision floating point traps (Y'0000 0400')	1 = double precision floating point software traps present 0 = double precision floating point 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 SVC 2 code 19

01	13	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

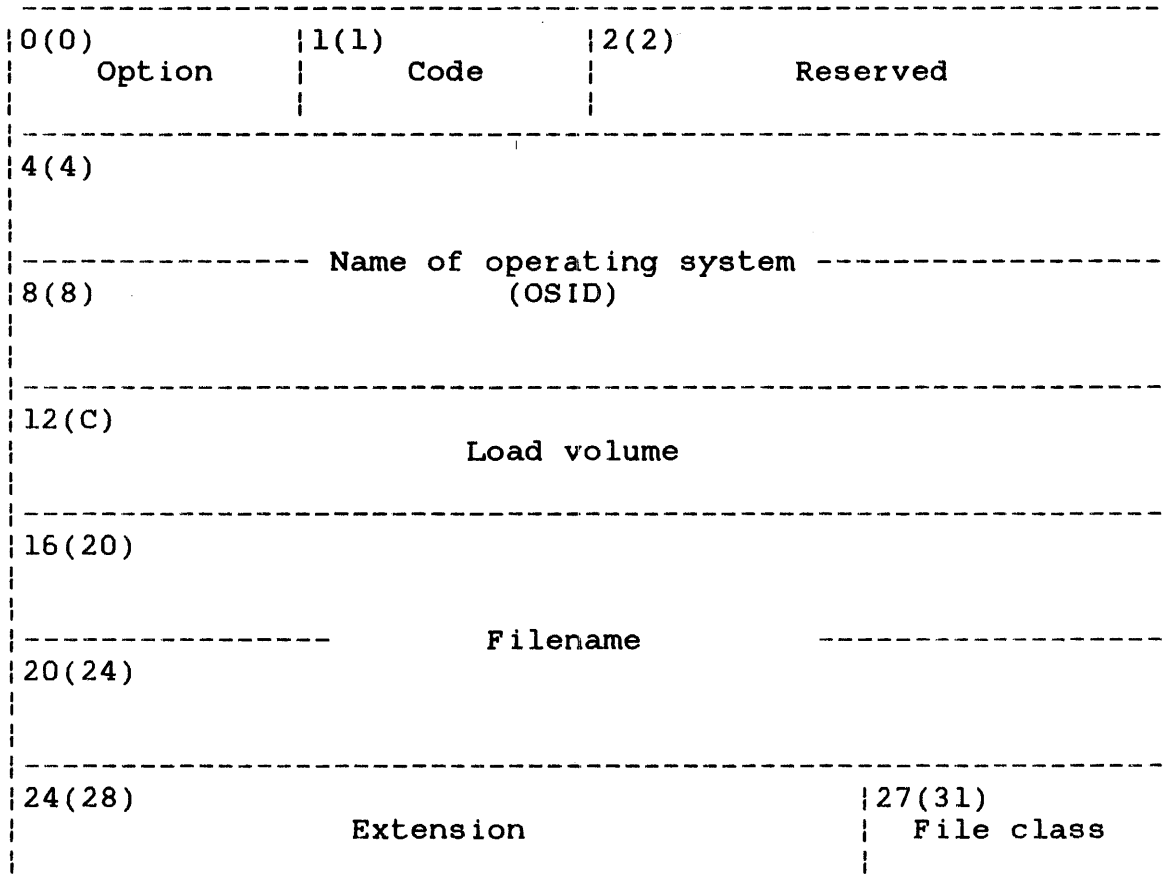
Parameter block after execution of SVC 2 code 19

01	13	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= Single precision floating point
- Time display on hexadecimal display panel
- Double precision floating point
- Direct access support
- Spooler option
- Roll option
- Temporary file support
- Multiple register set support
- Load read address support

3.17.3 Parameter Block for Option X'02'

If SVC 2 code 19 is executed with option X'02' specified in the parameter block option field, use the parameter block format in Figure 3-23.



```

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

```

Figure 3-23 SVC 2 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 SVC 2.
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	is the fd from which the 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 SVC 2 code 19

02	13	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 SVC 2 code 19

02	13	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

3.17.4 Parameter Block for Option X'03'

To execute SVC 2 code 19 with option X'03' specified in the parameter block option field, use the parameter block format in Figure 3-24. This option is used to obtain extended information on a task.

0(0)	Option	1(1)	Code	2(2)	Number of logical units (lu)	3(3)	Maximum priority (MPRI)
4(4)	Taskid (TID)						
8(8)	Task name						
12(C)	Current task status word (CTSW)						
16(10)	Task options (OPT)						
20(14)	Task waits						
24(18)	User account number						
28(1C)	Group account number (GACT)						
32(20)	Load volume						
36(24)	Filename						
40(28)	Extension					51(33)	File class
44(2C)	Monitor task name						
48(30)	Originating user console device (legacy)						
52(34)	Task Priority						
56(38)	65(41) Reserved			66(42) LPU		67(43) Reserved	
60(3C)	Task Priority						

```

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

```

Figure 3-24 SVC 2 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 code 19 of SVC 2.
Number of logical units (lu)	is a 1-byte field that receives from the TCB the maximum number of logical units which can be assigned to a task. This hexadecimal number ranges from 0 (X'00') through 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') through 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 Filename Extension 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' through X'09'; X'00' indicates the CPU.
Reserved	is a 1-byte reserved field that must contain zeros.

TABLE 3-5 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

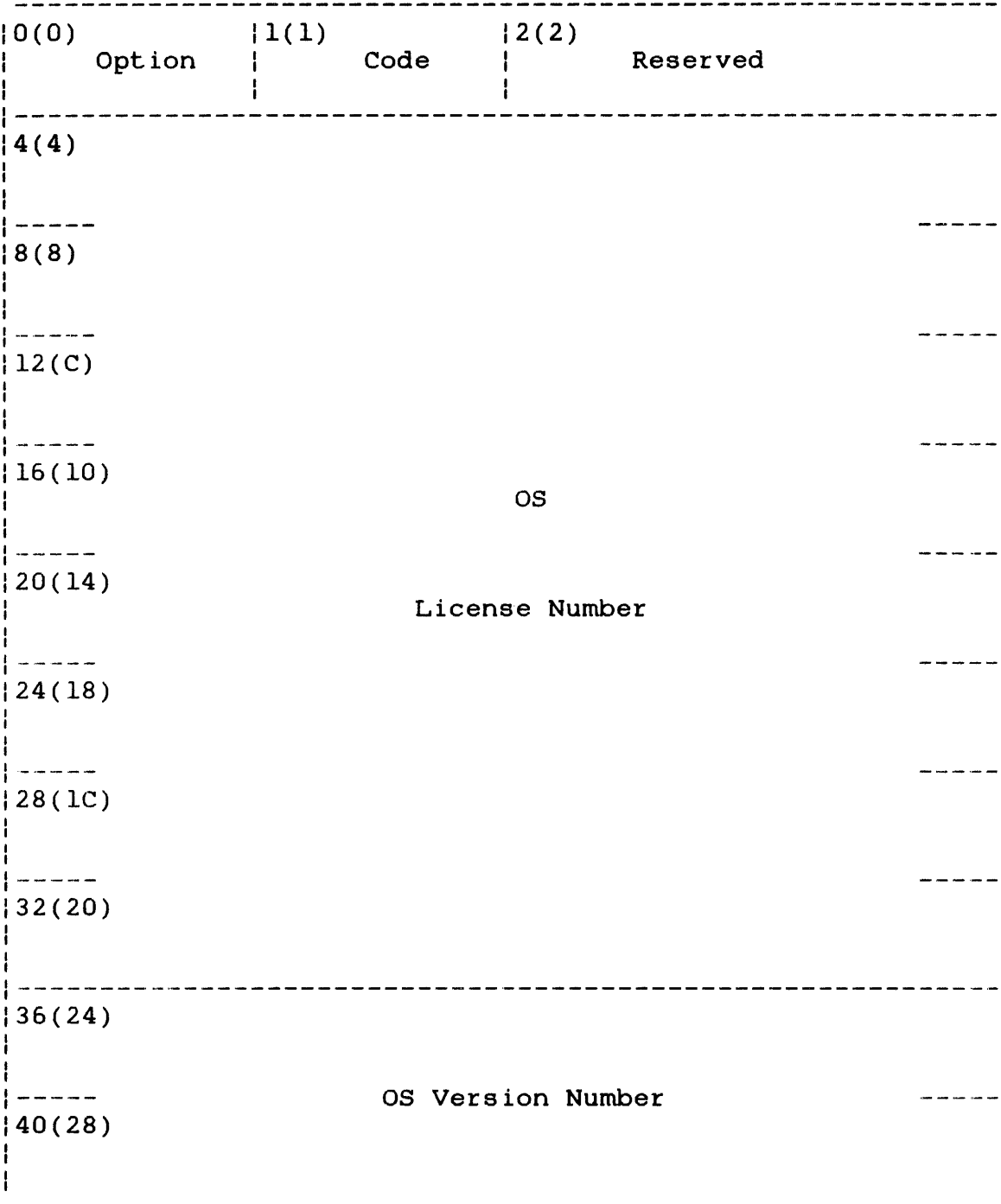
If bits 0 to 30 are set to 0, the task is active.

Parameter block after execution of SVC 2 code 19

03	13	0F	81
00	00	00	00
P	E	E	K
P	E	E	K
00	00	00	00
0C	03	08	A1
00	00	00	00
00	00	00	91
00	00	00	91
M	3	0	0
S	U	P	R
V	I	S	P
O	V	Y	S
.	M	T	M
Ø	Ø	Ø	Ø
D	I	1	8
81	00	01	00

3.17.5 Parameter Block for Option X'04'

Option X'04' accesses the license number and current sysgen version of the operating system that is currently running on the system. To execute option X'04' of SVC 2 code 19, use the parameter block format shown in Figure 3-25.



```

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

```

Figure 3-25 SVC 2 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 SVC 2 code 19.
Code	is a 1-byte field that contains the decimal number 19 indicating code 19 of SVC 2.
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., License 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').

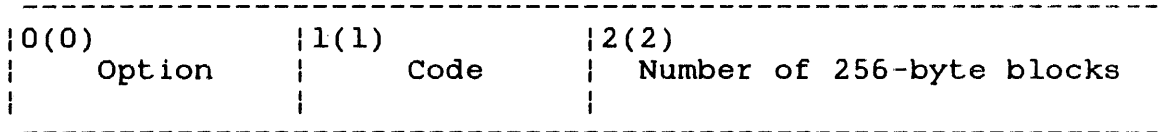
```

-----
|  SVC 2  |
| CODE 20 |
-----

```

| 3.18 SVC 2 CODE 20: EXPAND ALLOCATION

SVC 2 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 3-26.



```

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

```

| Figure 3-26 SVC 2 Code 20 Parameter Block Format and Coding

This parameter block is 4 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 code 20 of SVC 2.
- Number of 256-byte blocks is an unused 2 byte field.

The condition code is set after executing SVC 2 code 20.
Possible condition codes are:

Condition codes

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

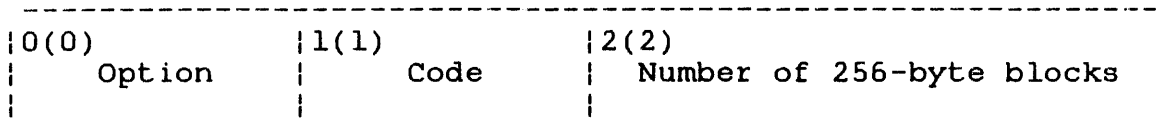
```

-----
|  SVC 2  |
| CODE 21 |
-----

```

| 3.19 SVC 2 CODE 21: CONTRACT ALLOCATION

SVC 2 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 3-27.



```

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

```

| Figure 3-27 SVC 2 Code 21 Parameter Block Format and Coding

This parameter block is 4 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 21 of SVC 2.
- Number of 256-byte blocks is an unused 2-byte field.

3.20 SVC 2 CODE 23: TIMER MANAGEMENT

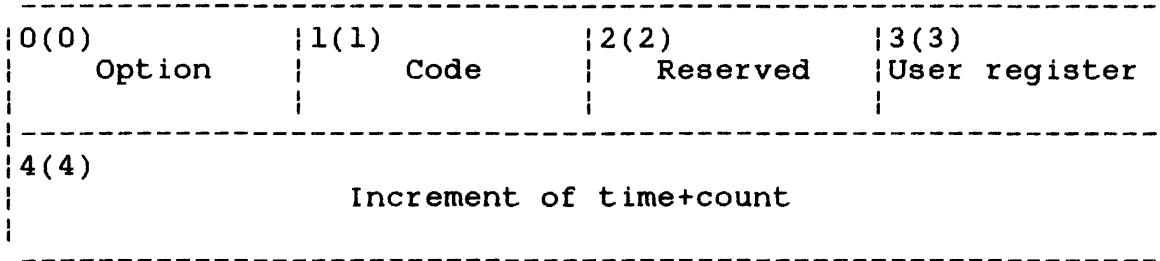
SVC 2 code 23 performs five timer management functions used in coordination with real time operations:

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 SVC 2 code 23 parameter blocks shown in Figures 3-28 through 3-32.

3.20.1 SVC 2 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 SVC 2 code 23 parameter block for option X'00' shown in Figure 3-28. 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 3-28 SVC 2 Code 23 Parameter Block Format and Coding for Option X'00'

This parameter block is 8 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 code 23 of SVC 2.
- 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 condition code is set, the task continues processing or enters a trap wait state. Possible condition codes are:

Condition codes

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'
```

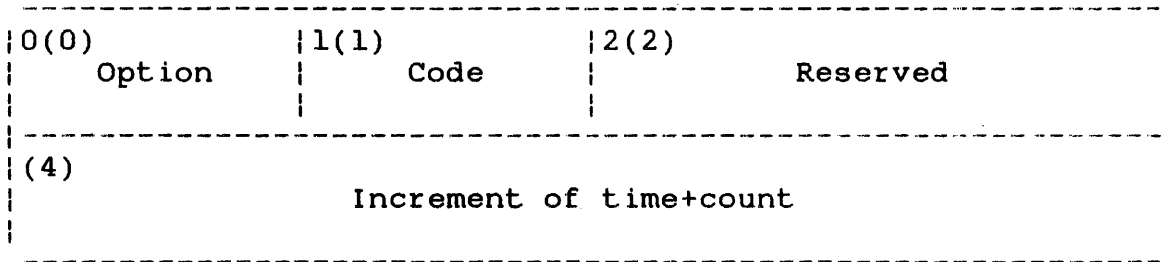
If this call is executed and insufficient system space is available, no time period elapses, no item is added to the task queue, and the condition code is set to 4 (V bit set). If this call is executed and the task is unprepared to handle this trap, no item is added to the task queue and the task has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

3.20.2 SVC 2 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 SVC 2 code 23 parameter block for option X'80' shown in Figure 3-29.



```

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

```

Figure 3-29 SVC 2 Code 23 Parameter Block Format and Coding for Option X'80'

This parameter block is 8 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 code 23 of SVC 2.
- 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 SVC 2. The possible condition codes are:

Condition codes

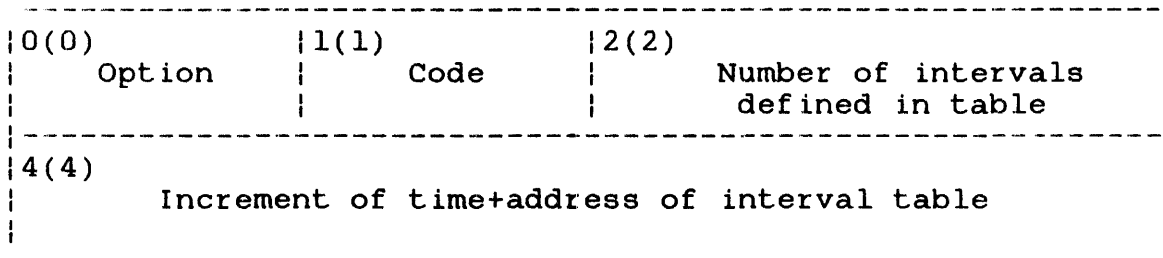
C	V	G	L	
0	0	0	0	Internal started; normal termination
0	1	0	0	Insufficient system space available; no wait occurred

If this call is executed and insufficient system space is available, no interval elapses, no item is added to the task queue, and the condition code 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.

3.20.3 SVC 2 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 SVC 2 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 SVC 2 code 23 parameter block for option X'40' shown in Figure 3-30.



```

                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 3-30 SVC 2 Code 23 Parameter Block Format and Coding for Option X'40'

This parameter block is 8 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 code 23 SVC 2.

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 elapse. 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 3 days.

Example:

	ALIGN 4	
INTABLE	DC F'54000'	1500 hours of current day
	DC F'1'	
	DC F'140399'	1500 hours of second day
	DC F'2'	
	DC F'227798'	1500 hours of third day
	DC F'3'	
	DC F'231498'	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 54000 ms. The
| time period is repetitively executed until the task cancels the
| time interval request via SVC 2 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 condition code is set to these possible settings:

Condition codes

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 condition code 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.

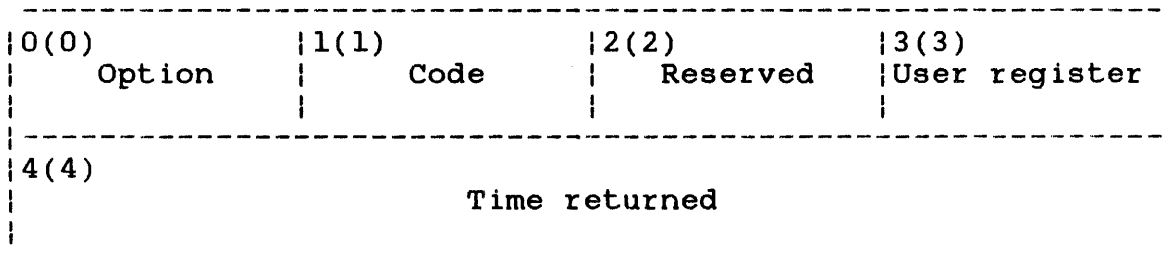
If queue overflow occurs after one of the specified intervals elapses, the end of task code is set to 1000 and the task terminates abnormally.

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.

3.20.4 SVC 2 Code 23 Parameter Block for Option X'20'

SVC 2 code 23 reads the time remaining until the interval previously established with option X'00' or X'40' elapses. This is accomplished through the SVC 2 code 23 parameter block for option X'20' shown in Figure 3-37.



```

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

Figure 3-31 SVC 2 Code 23 Parameter Block Format and Coding for Option X'20'

This parameter block must be 8 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 code 23 of SVC 2.

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 will be 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 SVC 2 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 SVC 2 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 condition code is set to these possible condition codes:

Condition codes

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:

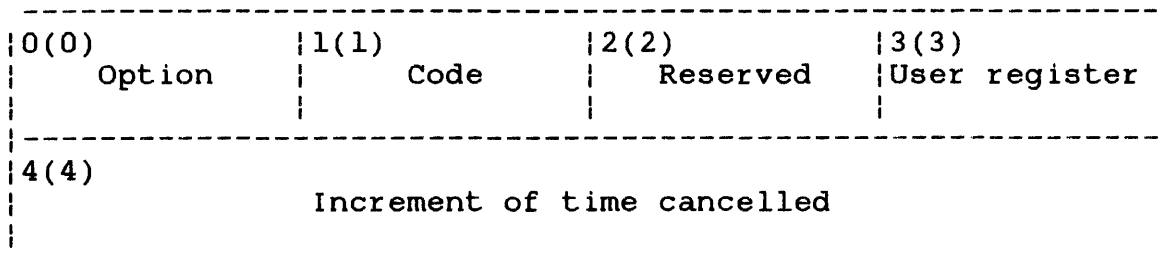
```

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'

```

3.20.5 SVC 2 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 SVC 2 parameter block for option X'10' shown in Figure 3-32.



```

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

```

Figure 3-32 SVC 2 Code 23 Parameter Block Format and Coding for Option X'10'

This parameter block is 8 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 code 23 of SVC 2.
- 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 SVC 2 code 23, the condition code is set to these possible conditions:

Condition codes

C	V	G	L
0	0	0	0
0	1	0	0

Normal termination

No previous interval request exists that matches the parameter provided

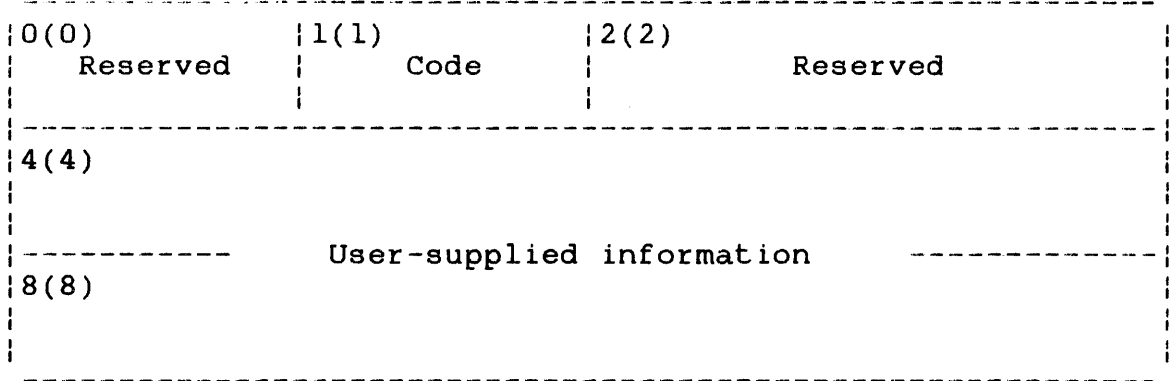
```

-----
|   SVC 2   |
|  CODE 24  |
|-----|

```

| 3.21 SVC 2 CODE 24: SET ACCOUNTING INFORMATION

| SVC 2 code 24 stores eight bytes of user-supplied information in
| the ATF task completion or data overflow account records of the
| accounting transactions file (ATF). This is accomplished through
| the SVC 2 code 24 parameter block shown in Figure 3-33.



```

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

```

| Figure 3-33 SVC 2 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 code 24 of SVC 2.

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 SVC 2 code 24 is executed by a task, the user-supplied information in the last SVC 2 code 24 executed is stored in the ATF. The condition code is always set to 0.

```

-----
|  SVC 2  |
| CODE 25 |
-----

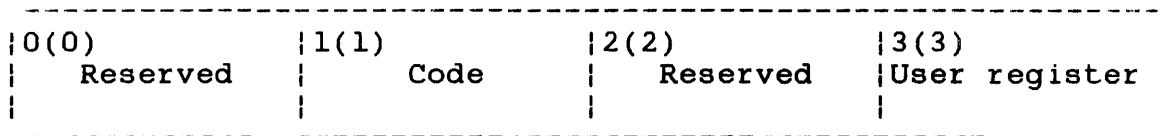
```

| 3.22 SVC 2 CODE 25: FETCH ACCOUNTING INFORMATION

SVC 2 code 25 fetches task accounting information and stores it into a user-specified area. The accounting information accessed is:

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

| This is accomplished through the SVC 2 code 25 parameter block shown in Figure 3-34.



```

                SVC    2,parblk
                .
                .
                .
parblk         ALIGN 4
                DB     0,25
                DB     0
                DB     user register

```

| Figure 3-34 SVC 2 Code 25 Parameter Block Format and Coding

This parameter block is 4 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 25 to indicate code 25 of SVC 2.
- Reserved is a reserved 1-byte field that must contain a 0.
- 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 3-35. The condition code is always set to 0.

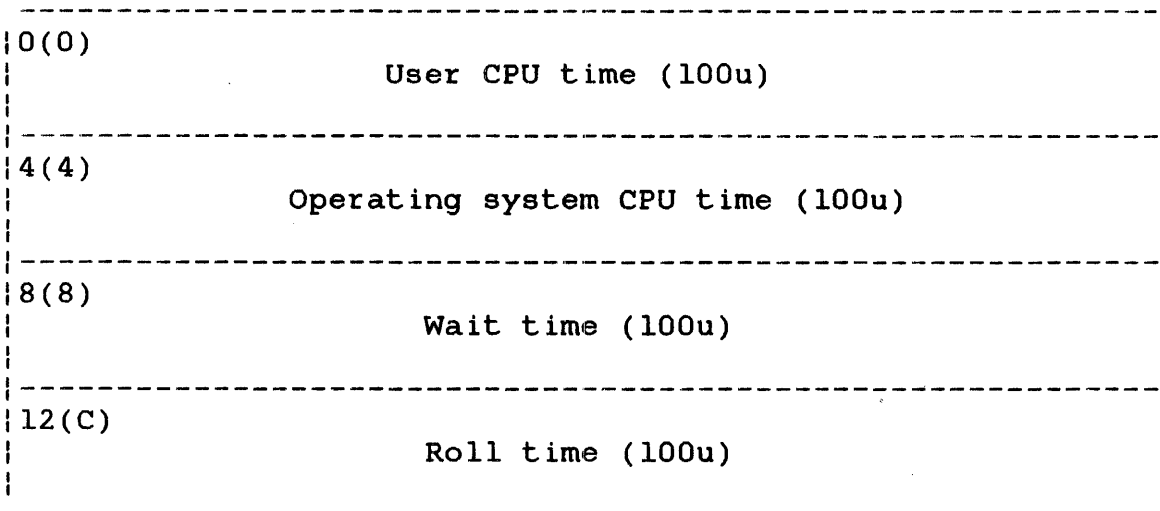


Figure 3-35 Area Receiving Accounting Information

```

-----
| SVC 2 |
| CODE 29 |
-----

```

3.23 SVC 2 CODE 29: UNPACK FILE DESCRIPTOR

SVC 2 code 29 converts a packed file descriptor from the file directory or an SVC 7 parameter block to its unpacked format. The format for the SVC 2 code 29 parameter block is shown in Figure 3-36.

```

-----
|0(0)          |1(1)          |2(2) Source   |3(3) Destina-|
| Option       | Code         | Register     |tion Register|
| (UPFD.OPT)  | (UPFD.COD)  | (UPFD.SRC)  | (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
                .
                .
                .
parblk         ALIGN 4
                DB     option,29
                DB     source register
                DB     destination register
                DAC    A(packed fd)
                DAC    BUFFADR

```

Figure 3-36 SVC 2 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 SVC 2 code 29 option codes:

OPTION CODE	FUNCTIONAL EQUATE	MEANING
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 condition code is set.

NOTE

If neither X'80' nor X'40' is specified, the fd will be 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, SVC 2 code 29 will unpack the fd contained in the SVC 7 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	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	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	The source address of the packed fd is specified by the UPFD.SAD field of the SVC 2 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 code 29 of SVC 2.
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 SVC 7 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 SVC 7 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 SVC 2 code 29.

Example 1:

```

SVC229  PROG  SVC 2,29 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 SVC 2,29 EXAMPLE - UNPACKED FD

NLSTM
\$SVC7

LA 1,SVC7PBLK
LA 2,DEST
SVC 2,UFD
SVC 2,PAUSE

ADDRESS OF SOURCE
ADDRESS OF DESTINATION
UNPACK FD

PAUSE ALIGN 4
DB 0,1,0,0
UFD DB X'58',29,1,2
DAC 0
DAC 0

PAUSE
PGS, SVC7, BLANKS, BLANKS, REG

SVC7PBLK DS SVC7.
ORG SVC7PBLK+SVC7.VOL
DC C'MTM '
DC C'TEST '
DB C'CSS '
DB C'G '

INPUT PACKED FD

DEST DS 24
END

OUTPUT UNPACKED FD

CHAPTER 4
END OF TASK SUPERVISOR CALL 3 (SVC 3)

4.1 INTRODUCTION

SVC 3 terminates task execution. This is accomplished through the SVC 3 format.

4.2 SVC 3: END OF TASK

Format:

SVC 3,n

Fields:

SVC is the mnemonic used as an operation code indicating that it is a supervisor call.

3 is a decimal number indicating it is SVC 3.

n is a decimal number ranging from 0 through 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 mapping error in impure segment during rollin.
- 1101 indicates mapping error in pure segment during rollin.
- 1102 indicates pure segment was not found during rollin.
- 1105 indicates I/O error on roll file for impure segment.
- 1106 indicates I/O error on roll file for pure segment.

- 1200 indicates termination caused by expiration of CPU time limit.
- 1210 indicates termination caused by expiration of 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 SVC 3 is executed, write operations continue until completed and then terminate normally; read operations terminate immediately.

For all logical units, read operations and SVC 15 operations are halted via an SVC 1 halt I/O. If the task is resident, an SVC 7 checkpoint is executed. If the task is nonresident, an SVC 7 close is executed.

For more information on using end of task codes in CSS, refer to the OS/32 Operator Reference Manual.

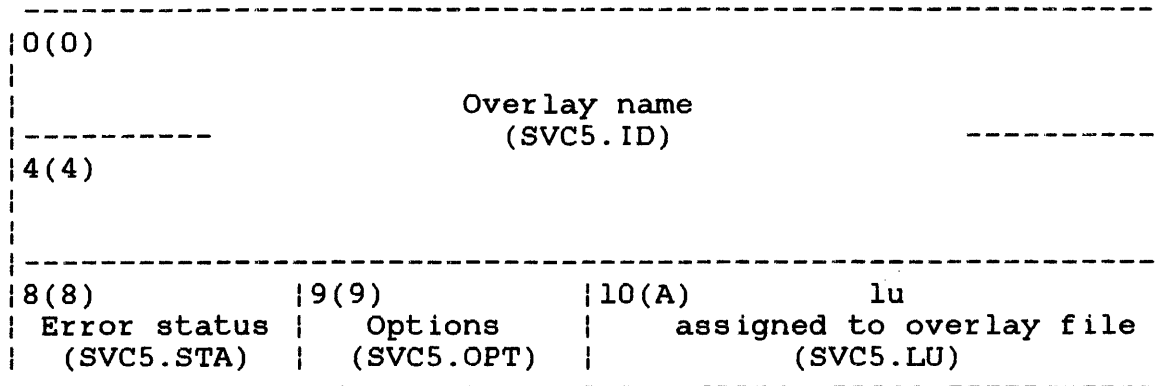
CHAPTER 5
FETCH OVERLAY SUPERVISOR CALL 5 (SVC 5)

5.1 INTRODUCTION

SVC 5 permits user-controlled loading of overlays generated by Link or TET. Loading of overlays is accomplished through the SVC 5 parameter block in Figure 5-1. The SVC 5 parameter block is 12 bytes long, fullword-boundary aligned, and must be in a task writable segment.

```
-----
| SVC 5 |
-----
```

5.2 SVC 5: FETCH OVERLAY



```

                SVC    5,parblk
                .
                .
                .
parblk        ALIGN 4
                DC     C'8 character overlay name'
                DS     1
                DB     'option'
                DC     H'lu'
```

Figure 5-1 SVC 5 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 in the overlay descriptor table (ODT) of the task image file. The ODT contains the information needed by Link to process the overlay. If the overlay name is found in the ODT, 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 SVC 5. 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 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 logical unit (lu) without positioning.
- Option X'04' indicates load from lu after rewind.

The option byte is not required for overlays generated by Link.

Logical unit
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.

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
                .
                .
                .
parblk          ALIGN 4
                DC     C'MARIANNE'
                DB     0           Initialize status to 0
                DB     1           Load without print
                DC     H'2'        Overlay assigned to lu 2
```

CHAPTER 6
INTERTASK COMMUNICATIONS SUPERVISOR CALL 6 (SVC 6)

6.1 INTRODUCTION

SVC 6 provides a task with the ability to communicate with and control another task. The task that issues an SVC 6 is known as the calling task. An SVC 6 can be directed to any task within the calling task's execution environment, including the calling task itself. The task to which the SVC 6 is directed is called the directed task.

Before a calling task can issue an SVC 6, that task must be linked with one of the following task options:

- COMMUNICATE - this option allows a calling task to perform SVC 6 intertask communication functions. See Section 6.2.1.
- CONTROL - this option allows a calling task to perform SVC 6 intertask control functions. See Section 6.2.1.

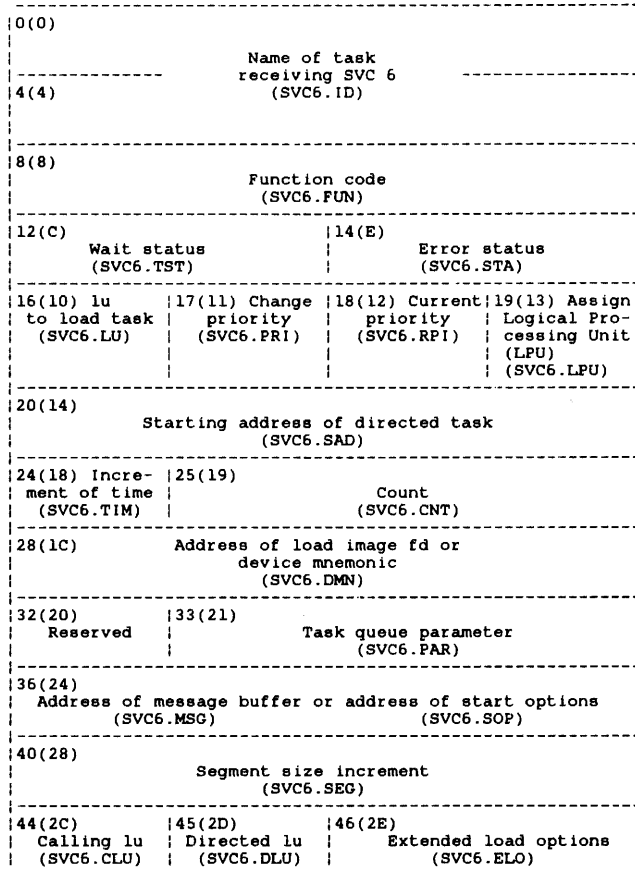
In an OS/32 real-time environment, only foreground tasks can issue an SVC 6. If a background (BG) task attempts to issue this call, the OS will treat the call as an illegal call or NOP, depending on the SVCPAUSE task option in effect. See the OS/32 Link Reference manual for more information on the task options that apply to SVC 6.

NOTE

SVC 6 cannot be executed in a multi-terminal monitor (MTM) environment unless specified as an MTM option at system generation (sysgen).

6.2 SVC 6: INTERTASK COMMUNICATIONS

Communication and control between tasks are accomplished through the SVC 6 parameter block shown in Figure 6-1.



```

SVC 6,parblk
.
.
ALIGN 4
parblk DC C'8-byte name of task receiving SVC 6'
        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 logical processing unit
        DC A(start)
        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 SVC 6 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, refer to the appropriate section in this chapter. A brief description of each field in the parameter block follows.

Fields:

Name of task receiving SVC 6 (SVC6.ID)	is an 8-byte field that contains the task name to which SVC 6 is directed. If SVC 6 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 SVC 6 is executed. If the calling task wants to check the wait status of the directed task at any time, an SVC 6 can be issued with bits 0 and 1 of the function code set to 10 or 11 and the remaining bits set to 0.
Error status (SVC6.STA)	is a 2-byte field that receives the appropriate error code when an error occurs during execution of the SVC 6. 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 must contain the user-specified hexadecimal number indicating the logical unit 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 through 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 SVC 6 is executed. If the calling task wants to check the current priority of the directed task at any time, an SVC 6 can be issued with bits 0 and 1 of the function code set to 10 or 11 and the remaining bits set to 0.

Assign 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:

- Option X'00' indicates seconds from midnight (time of day).
- Option 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 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 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 options listed in Table 6-3.

6.2.1 Function Code (SVC6.FUN)

SVC6.FUN has 21 functions for intertask communication 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.

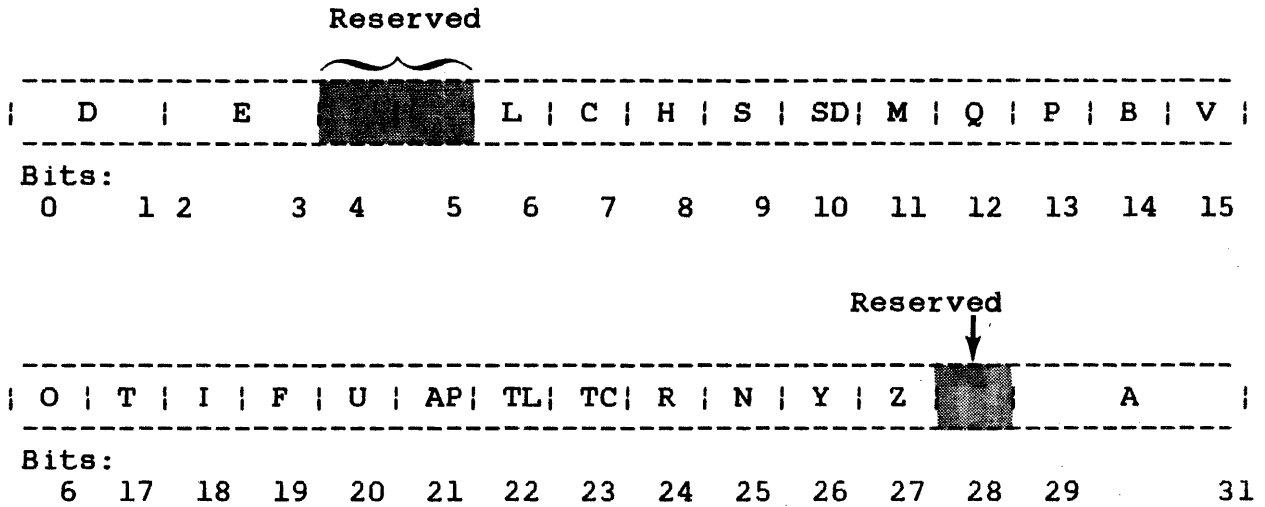


Figure 6-2 SVC 6 Function Code Field

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC 6 CALLS

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
0 (D) 1	Direction (SFUN.DOM=10) (SFUN.DSM=11)	The task to which SVC 6 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

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC 6 CALLS
(Continued)

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
4 5	N/A	Reserved	00 = reserved
6 (L) 7 (C)	Load task (SFUN.LM=10 SFUN.LXM=01)	Load 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 directed task resident in memory.	0 = no function requested 1 = make task resident
9 (S)	Suspend (SFUN.SM)	Put 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 directed task.	0 = no function requested 1 = send message
11 (M)	Send message (SFUN.MM)	The calling task sends a 64-byte message the directed task.	0 = no function requested 1 = send message
12 (Q)	Add to task queue (SFUN.QM)	Add parameter to directed task queue.	0 = no function requested 1 = add to task queue
13 (P)	Change priority (SFUN.PM)	Change priority of directed task.	0 = no function requested 1 = change the priority

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC 6 CALLS
(Continued)

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
14 (B)	Send lu (SFUN.XSM=10)	Calling task's lu is assigned to the directed task.	00 = no function requested 10 = send lu
15 (V)	Receive lu (SFUN.XRM=01)	Directed task's lu is assigned to calling task.	01 = receive lu 11 = illegal
16 (O)	Connect (SFUN.OM)	A trap generating device is connected to directed task.	0 = no function requested 1 = connect device to task
17 (T)	Thaw (SFUN.TM)	Enable interrupts on trap generating device connected to directed task.	0 = no function requested 1 = enable interrupts
18 (I)	Sint (SFUN.IM)	Simulate interrupt on trap generating device to directed task.	0 = no function requested 1 = simulate interrupt
19 (F)	Freeze (SFUN.FM)	Disable interrupts on trap generating device connected to directed task.	0 = no function requested 1 = disable interrupts
20 (U)	Unconnect (SFUN.UM)	Disconnect specified trap generating device from directed task.	0 = no function requested 1 = disconnect device from task
21 (AP)	Assign LPU (SFUN.LPM)	Assign LPU to directed task	0 = no function requested 1 = assign LPU

TABLE 6-2 DESCRIPTION OF FUNCTION CODE FIELD FOR SVC 6 CALLS
(Continued)

BIT POSITIONS	FUNCTIONS AND MASK NAMES	MEANING	BIT SETTINGS
22 (TL)	Transfer to LPU (SFUN.XLM)	Make task LPU-directed	0 = no function 1 = set LPU-directed status in TCB
23 (TC)	Transfer to CPU (SFUN.XCM)	Make task CPU-directed	0 = no function requested 1 = reset LPU-directed status in TCB
24 (R)	Release (SFUN.RM)	Remove directed task from wait state.	0 = no function requested 1 = remove task from state
25 (N)	Nonresident (SFUN.NM)	Make directed task nonresident.	0 = no function requested 1 = make task nonresident
26 (Y)	Rollable (SFUN.RLM)	Make directed task rollable.	0 = no function requested 1 = make task rollable
27 (Z)	Nonrollable (SFUN.NRM)	Make directed task nonrollable.	0 = no function requested 1 = make task nonrollable
28	N/A	Reserved	0 = reserved
29 30 (A) 31	Start (SFUN.SIM=010) (SFUN.SOM=011) (SFUN.SDM=100) or =110)	Start execution of 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)

| The direction function identifies the task to be affected by the
| SVC 6 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
- Bits 0 and 1 of the function code field

If the bit setting equals 10 (SFUN.DOM), the call is directed to any system task whose name must be specified in the task name field. If the bit setting equals 11 (SFUN.DSM), the call is self-directed or 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)

The end task function abnormally terminates or 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.
- The task remains in memory.
- All 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.
- The task is removed from memory.
- All the task's assigned files and devices are closed.

When the bit setting equals 10 or 11 (SFUN.EDM) and the directed task is resident, these operations occur:

- Task execution is cancelled.
- The task is made nonresident.
- The task is removed from memory.
- All the task's assigned files and devices are closed.

When the bit setting equals 10 or 11 (SFUN.EDM) and the directed task is nonresident, these operations occur:

- Task execution is cancelled.
- The task is removed from memory.
- All the task's assigned files and devices are closed.

If this call is self-directed, SVC 6 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)

The required parameter block fields for bit setting 10 (SFUN.LM) are:

- Task name field
- Bits 0, 1, 6, and 7 of the function code field
- lu to load task field

| Before executing this call, the lu specified in the parameter
| block must be assigned to the file or device containing the
| directed task image file. 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 byte following the loaded task. If the same task is to be reloaded from other than a direct access device with the same lu assigned, the lu must be rewound by using SVC 1 prior to each subsequent load. For direct access devices the load task function initializes the start address to zero.

If the following error conditions occur, SVC 6 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)

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.

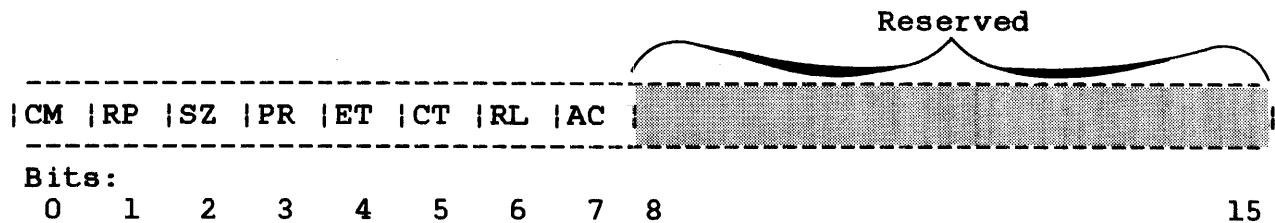


Figure 6-3 Extended Load Options Field

The required parameter block fields for bit setting 7 (SFUN.LXM) are:

- Task name
- Bits 0, 1, 6, and 7 of the function code field
- lu to load task field (required when the load wait extended option is specified)
- Address of load image fd or device mnemonic field (required when load and proceed extended option is specified)
- Extended load options field
- Segment size increment field. This field is required only when the task to be loaded needs a memory area larger than the task image size. The extended load option SELO.SZM must be set.

When a task is loaded with the SFUN.LXM enabled, any options specified in the extended load options field are in effect during execution 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 SVC 6 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 byte following the last byte of the task image. If the task is again loaded from the same lu, an SVC 1 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)	<p>If bit 0 equals 1, the directed task that was loaded into memory can execute the SVC 6 communication functions.</p> <p>If bit 0 equals 0 and the loaded receiving task issues an SVC 6 communication function, the call is rejected, and an error code is stored in the error status field of the parameter block.</p>
1 (RP)	Subtask reporting (SELO.RPM)	<p>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 monitor the task through task traps.</p> <p>If bit 1 equals 0, the directed task is not a subtask. No subtask status changes are reported.</p>
2 (SZ)	Segment size Increment (SELO.SZM)	<p>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.</p> <p>If bit 2 equals 0, the workspace set by the WORK= parameter of the LINK OPTION command is used.</p>

TABLE 6-3 EXTENDED LOAD OPTIONS FIELD BIT DEFINITIONS
(Continued)

BIT POSITION	OPTION AND MASK NAME	MEANING
3 (PR)	Load and proceed (SELO.PRM)	<p>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 SVC 6 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 X'7' and the SVC 6 parameter block address are added to the task queue.*</p> <p>If bit 3=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.</p>
4 (ET)	Prevent e-task or d-task (SELO.ETM)	<p>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 SVC 6 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 ACPRIVILEGE, DISC, or the INTERCEPT task option, the task will be loaded but these options will be disregarded.</p>
5 (CT)	Intertask control (SELO.CTM)	<p>If bit 5 equals 1, the directed task that was loaded into memory can execute the control functions of SVC 6.</p> <p>If bit 5 equals 0 and the directed task that was loaded issues an SVC 6 control function, the call is rejected, and an error code is stored into the error status field of the parameter block.**</p>

TABLE 6-3 EXTENDED LOAD OPTIONS FIELD BIT DEFINITIONS
(Continued)

BIT POSITION	OPTION AND MASK NAME	MEANING
6 (RL)	Roll (SELO.RLM)	<p>If bit 6 equals 1, the directed task is forced to be a rollable task regardless of the roll option established by Link.</p> <p>If bit 6 equals 0, the directed task uses the roll option established by Link.</p>
7 (AC)	Accounting (SELO.AEM)	<p>If bit 7 equals 1, the directed task that was loaded into memory is given the accounting option. This setting will override the NOACCOUNT option established by Link.</p> <p>If bit 7 equals 0, the directed task uses the accounting option established by Link.</p>

* When bit 3 equals 1, all other SVC 6 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 trap generating device (TGD) functions can be executed if bit 5 equals 1.

6.2.5 Task Resident (SFUN.HM)

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
- Bits 0, 1, and 8 of the function code field

6.2.6 Suspend (SFUN.SM)

The suspend function places the directed task into a wait state. The required parameter block fields are:

- Task name receiving SVC 6 field
- Bits 0, 1, and 9 of the function code

The directed task remains in the wait state until an SVC 6 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 APU active or ready tasks. See the OS/32 System Level Programmer Reference Manual for more information on using SVC 6 in a Model 3200MPS multiprocessing system.

6.2.7 Send Data (SFUN.DB)

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 Buffers 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 SVC 6.

Hence, two data structures are required by the calling task, the SVC 6 parameter block and the send data message buffer. The required SVC 6 parameter block fields for this function are:

- Task name field
- Bit 10 of the function code field
- Address of the buffer containing the message to be sent (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.

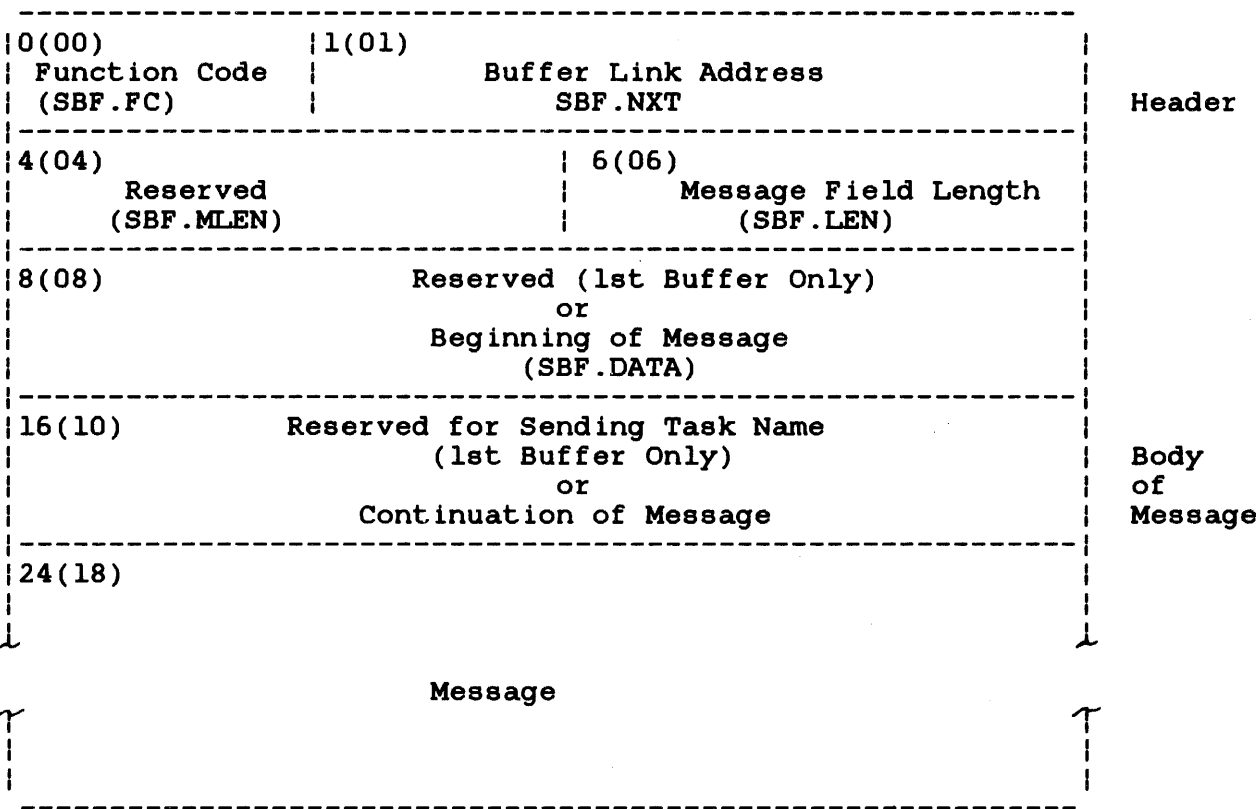


Figure 6-4 Send Data Message Buffer Format for Calling Task

Each send data message buffer can be variable in length provided that the buffer is aligned on a fullword boundary and its total length is equal to an integer 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 SVC 6.

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 8 bytes of the message body of the directed task buffer are filled with zeros, the next 8 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 buffer indicating the address of the next buffer in the chain. The OS 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 indicating the length of the message body for that buffer.

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

- 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 free send data message buffers should be sufficient to hold the entire message transferred to those buffers by the calling task. The format of the free send data message buffers is shown in Figure 6-5.

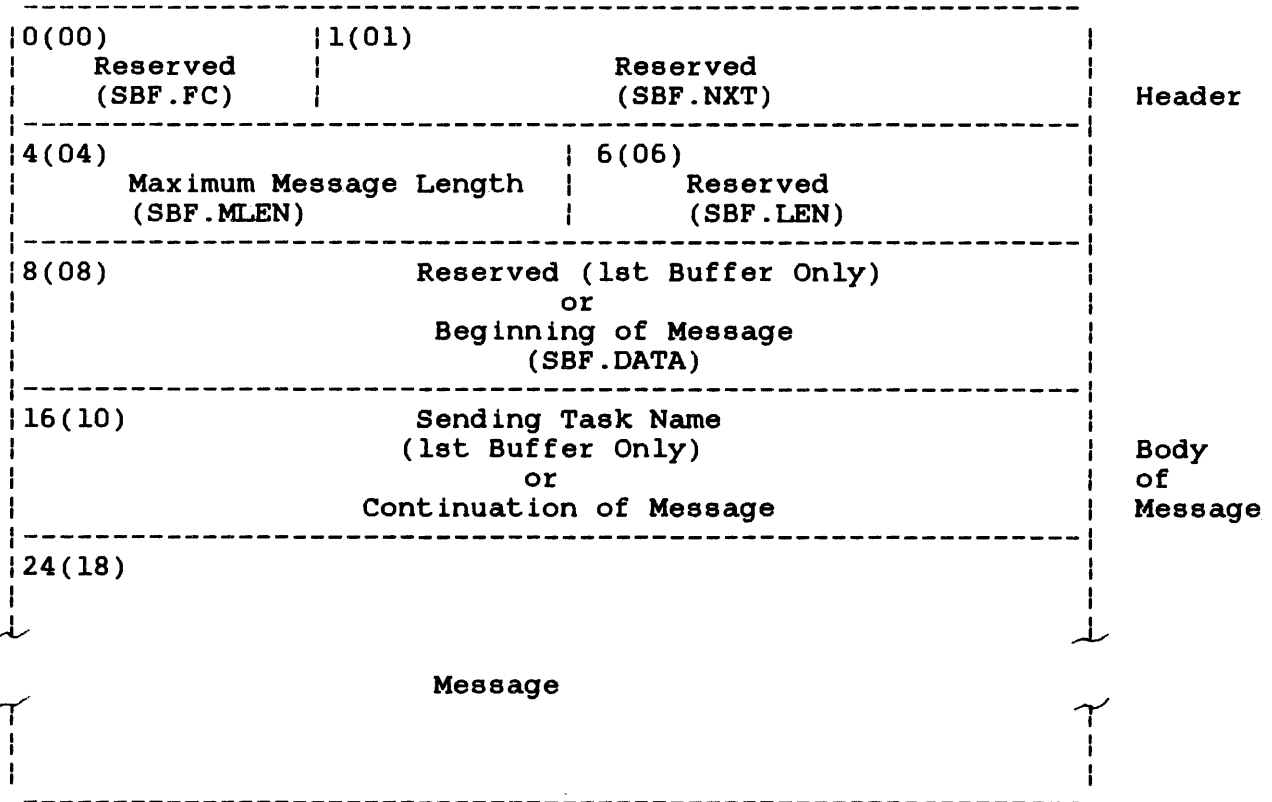


Figure 6-5 Send Data Message Buffer Format for Directed Task

Like the calling task's message buffers, each free message buffer can be variable in length as long as the buffer is aligned on a fullword boundary and the total length in bytes is an even number.

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 SVC 6 to send a message, the OS 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 OS 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 OS 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 OS 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 OS continues to fetch and fill the directed task's free buffers until the entire message is transferred 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 OS outputs an error message indicating no message was sent.

If the entire message is successfully transferred, the OS places reason code X'06' 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 will be returned to the calling task.

6.2.7.3 Sample Programs Using SVC 6 Send Data Function

The following sample programs demonstrate the data structures used to send a message via the SVC 6 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+SVCL.SAD	STORE THE STARTING ADDR
	ST	9,WRITE+SVCL.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+SVCL.SAD	STORE THE STARTING ADDR
	ST	9,WRITE+SVCL.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+SVCL.SAD	STARTING ADDR
	ST	9,WRITE+SVCL.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+SVCL.SAD	STARTING ADDR
	ST	9,WRITE+SVCL.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	SVCL.	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)

The send message function allows calling task to send a 64-byte message to the directed task. SVC 6 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 on to the directed task queue.

The required SVC 6 parameter block fields are:

- Task name field
- Bits 0, 1, and 11 of the function code field
- Address of message buffer field

To prepare a directed task to accept the message sent by the SVC 6 send message function:

- Allocate message buffers to receive the message. (Use 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.

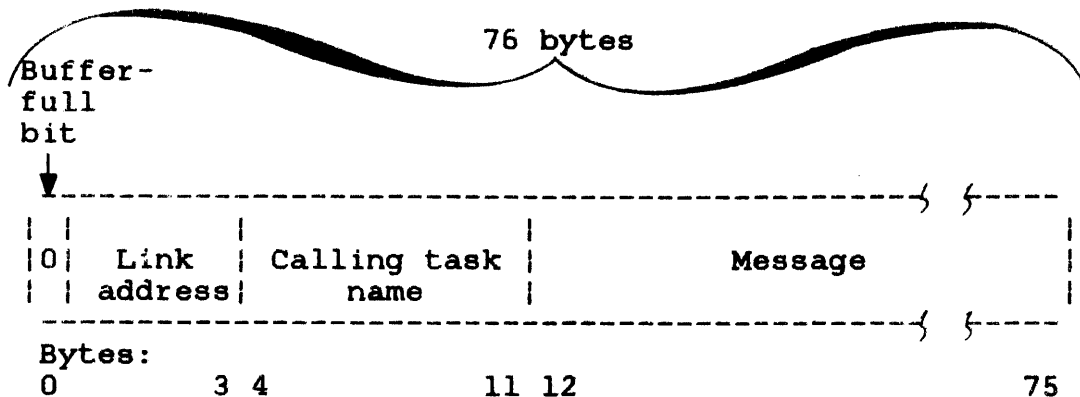


Figure 6-6 Message Buffer Format for Directed Task

This message buffer must be 76 bytes long and aligned on a fullword boundary. 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:

1. Single buffer ring
2. Single buffer chain
3. Multiple buffer ring
4. 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.

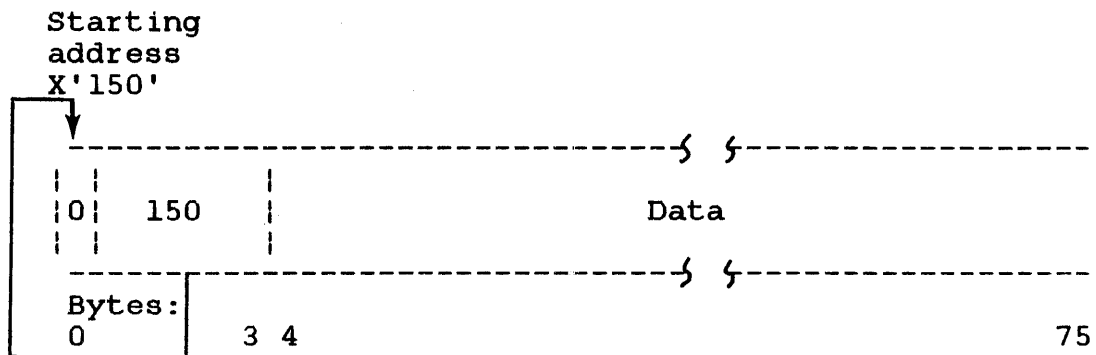


Figure 6-7 Single Buffer Ring

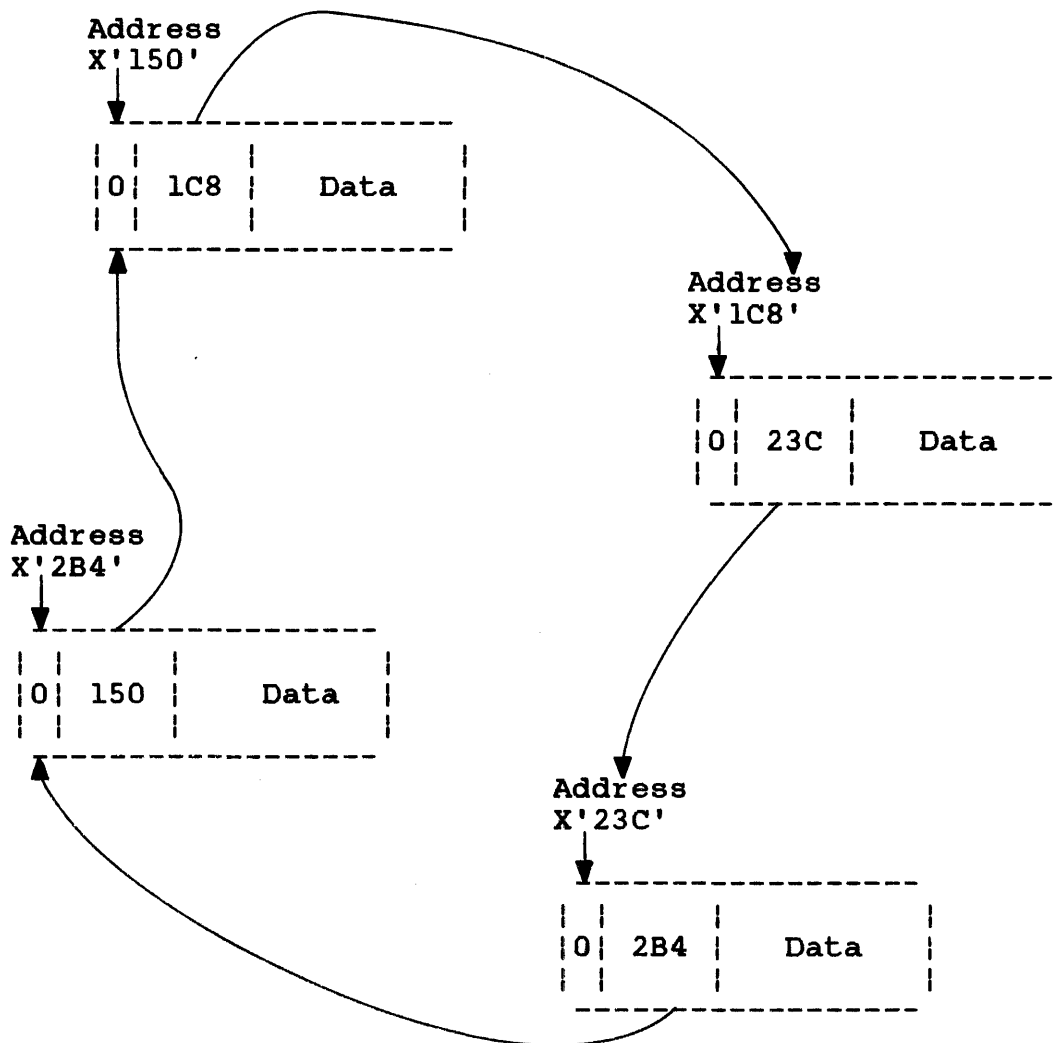


Figure 6-9 Multiple Buffer Ring

A multiple buffer chain consists of a variable, unlimited number of buffers. Each buffer-full bit initially should 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.

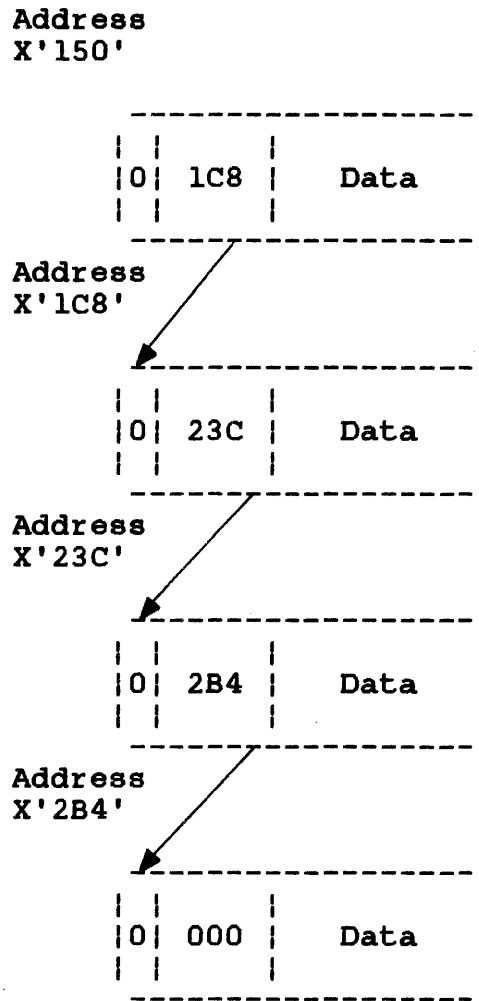


Figure 6-10 Multiple Buffer Chain

The following sample programs demonstrate the data structures used to send a message via the SVC 6 send message function.

Sample send message application: Sending task

```
SEND    PROG  SVC 6 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
```

```
SVC6    ALIGN  4  
        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  SVC 6 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(R01)	SAVE TASK QUEUE NEW TSW
	LA	R15,TSKQ	ADDRESS OF TASK QUEUE
	ST	R15,UDL.TSKQ(R01)	
	LA	R15,MESSQ	ADDRESS OF MESSAGE BUFFER RING
	ST	R15,UDL.MSGR(R01)	
	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	*	
	RTL	R02,TSKQ	A ITEM ON THE TASK QUEUE?
	BO	QEMPTY	NO - ENTER TRAP WAIT
	LA	R15,0(R02)	GET PARAMETER
	SRL	R02,24	ISOLATE REASON CODE
	CLHI	R02,6	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	R00,0(R15)	RESET MESSAGE ACTIVE FLAG
	NI	R00,Y'7FFFFFFF'	
	ST	R00,0(R15)	
	L	R00,12(R15)	GET FIRST FOUR BYTES OF MESSAGE
	CLI	R00,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
TRAPENA		ALIGN 4	
	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)

The queue parameter function adds the user-specified parameter, located in the task queue parameter field of the parameter block, to the directed task queue. The required parameter block fields are:

- Task name field
- Bits 0, 1, and 12 of function code field
- Task queue parameter field

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 queue is full, the call is rejected, the parameter is lost, and an error code is stored into the error status field of the parameter block.

6.2.10 Change Priority (SFUN.PM)

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 change priority value at which the directed task is now executing into the current priority field of the parameter block. The required parameter block fields are:

- Task name field
- Bits 0, 1, and 13 of the function code field
- Change priority field

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 (SFUN.XSM)

| The send logical unit 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
- Bits 0, 1, and 14 of the function code field
- Calling lu field
- Receiving lu field

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 SVC 6.

6.2.12 Receive Logical Unit (SFUN.XRM)

| 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
- Bits 0, 1, and 15 of the function code field
- Calling lu field
- Receiving lu field

Before the calling task accepts the lu that 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 SVC 6.

6.2.13 Connect (SFUN.OM)

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
- Bits 0, 1, and 16 of the function code field
- Device mnemonic field
- Task queue parameter field

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.

6.2.14 Thaw (SFUN.TM)

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
- Bits 0, 1, and 17 of the function code field
- Device mnemonic field

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 affect.

6.2.15 Sint (SFUN.IM)

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 affect. The required parameter block fields are:

- Task name field
- Bits 0, 1, and 18 of the function code field
- Device mnemonic field

When the simulate 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.

6.2.16 Freeze (SFUN.FM)

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
- Bits 0, 1, and 19 of the function code field
- Device mnemonic field

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 affect.

6.2.17 Unconnect (SFUN.UM)

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
- Bits 0, 1, and 20 of the function code field
- Device mnemonic field

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.

6.2.18 Assign LPU (SFUN.LPU)

The assign LPU function assigns a logical processing unit number to the directed task. 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
- Bits 0, 1, and 21 of the function code field
- LPU assignment field

6.2.19 Transfer to LPU (SFUN.TL)

The transfer to LPU function sets the LPU-directed status field in the TCB of the directed task. As a result, the directed task is transferred 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 and the calling task is executing on an APU, no transfer occurs. If a calling task executing on the CPU directs this function to itself, the LPU-directed status field in the TCB is set 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
- 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 SVC 6 parameter block, only bit 23 is recognized when the call is complete.

6.2.20 Transfer to CPU (SFUN.TC)

The transfer to CPU function resets the LPU-directed status field in the TCB. 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 processing.

If a calling task executing on the CPU directs this function to itself, no transfer occurs. 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 SVC 6, or
- the task is assigned to a different APU through an LPU assignment, or

- the task is cancelled or goes to end of task, or
- 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
- 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 SVC 6 parameter block, only bit 23 is recognized when the call is completed.

6.2.21 Release (SFUN.RM)

The release function releases a directed task currently suspended by a previous SVC 6 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
- Bits 0, 1, and 24 of the function code field

6.2.22 Nonresident (SFUN.NM)

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
- Bits 0, 1, and 25 of the function code field

6.2.23 Rollable (SFUN.RLM)

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
- Bits 0, 1, and 26 of the function code field

6.2.24 Nonrollable (SFUN.NRM)

The nonrollable function prevents the directed task from being rolled. The required parameter block fields are:

- Task name field
- Bits 0, 1, and 27 of the function code field

If both rollable and nonrollable functions are specified, only the nonrollable function is recognized.

6.2.25 Start (Bit Positions 29, 30, 31)

The start function starts execution of the directed task. This call is rejected if it is self-directed. Four methods of starting are:

1. Start (bit setting equals 010)
2. Start with start options (bit setting equals 011)
3. Delay start (bit setting equals 100 or 110)
4. Delay start with start options (bit setting equals 101)

The required parameter block fields are:

- Task name field
- Bits 0, 1, 29, 30, and 31 of the function code field
- Address of start options field (only required when start with start options or delay start with start options is specified in the function code)

- Increment of time field (only required when delay start or delay start with start options is specified)
- Count field (only required when delay start or delay start with start options is specified)
- Starting address of directed task field

Before the start function is executed, the directed task must be:

- loaded or present in memory, and
- in a dormat or console wait state.

6.2.26 Start Function for SVC 6 (SFUN.SIM)

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 SVC 6 (SFUN.SOM)

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 system generation (sysgen) time through the CMDLENGTH option. If the length of the start options is greater than that specified at sysgen time or a carriage return is present within the start options, only those characters up to the maximum number or the carriage return 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 SVC 6 (SFUN.SDM)

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 OS for SVC 6 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 SVC 6 (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 OS for SVC 6 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 carriage return is present within the start options, only those characters up to the maximum number or the carriage return 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 SVC 6 is executed.

If the calling task wants to check the wait status of the directed task, an SVC 6 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 SVC 6 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 indicated 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 through 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 SVC 6 error codes.

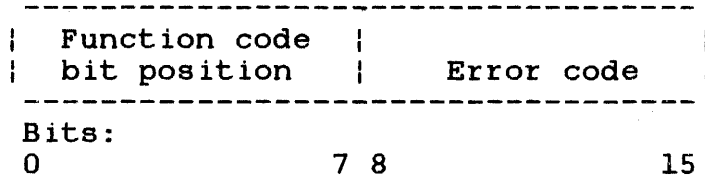


Figure 6-11 Error Status Field

TABLE 6-5 SVC 6 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.

TABLE 6-5 SVC 6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
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 the range of 10 through 249.
6	6 (L)	The directed task requires floating point facilities that are not sysgened into the system.
7	9 (S)	The specified directed task is dormant, paused, or suspended.
	14 (B)	The specified directed task is not dormant, paused, or suspended.
	15 (V)	
	29	
	30 (A)	
31		
8	11 (M)	The message is not aligned on a fullword boundary or an invalid starting address was specified for a directed task.
	29	
	30 (A)	
	31	
9	All	The calling task cannot execute SVC 6 control or communication functions.
A (10)	29	The values specified for the increment of time and count fields are invalid.
	30 (A)	
	31	
B (11)	11 (M)	The calling task message was not sent to the directed task.
	10 (SD)	
C (12)	10 (SD)	Task queue service in the directed task TSW is disabled. The directed task queue is full. The directed task has no queue.
	11 (M)	
	12 (Q)	

TABLE 6-5 SVC 6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
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.
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.

TABLE 6-5 SVC 6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
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 will exceed 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 RTL or a TCOM required by the directed task is not present at load time.
43 (67)	6 (L)	The calling task specified load options and the directed task specified link options are not the same.
44 (68) 45 (69)	6 (L) 6 (L) 14 (B) 15 (V) 29 30 (A) 31	The LIB format is invalid. 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 overlay descriptor table (ODT) of a tree structured overlay is invalid.

TABLE 6-5 SVC 6 ERROR CODES (Continued)

ERROR CODE HEXADECIMAL (DECIMAL)	FUNCTION CODE BIT POSITIONS CAUSING THE ERROR	MEANING
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)	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.
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 SVC 1 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 (SVC 7)

7.1 INTRODUCTION

SVC 7 provides file and device handling functions supported by the file manager and the data communications subsystem. These functions are accomplished through the SVC 7 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.

SVC 7

7.2 SVC 7: FILE HANDLING SERVICES

0(0)	Function code (SVC7.OPT)	2(2)	Error status (SVC7.STA)	3(3)	lu (SVC7.LU)
4(4)	Write key (SVC7.WKY)	5(5)	Read key (SVC7.RKY)	6(6)	Logical record length (SVC7.LRC)
8(8)	Volume name or device mnemonic (SVC7.VOL)				
12(C)					
16(10)	Filename (SVC7.FNM)				
20(14)	Extension (SVC7.EXT)			23(17)	File class/account (SVC7.ACT)
24(18)	File size (SVC7.SIZ)				

```

SVC    7,parblk
      .
      .
      .
parblk ALIGN 4
      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 SVC 7 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 SVC 7. 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 SVC 7 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 hexadecimal number indicating a file logical record length or a device physical record length.

Volume name or device mnemonic (SVC7.VOL) is a 4-byte field that contains ASCII code indicating the volume name of a direct access device, the device mnemonic of a nondirect access device or 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 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 or class to which the file is allocated. If SVC 7 is issued by an e-task or a u-task that was link-edited with the ACPRIVILEGE option, an account number can be specified in this field.

NOTE

To allocate a file with an account number, the file descriptor (fd) must be packed into the SVC 7 parameter block using SVC 2 code 16 (See Section 3.14.9.) The account number can range from 0 through 65,535.

If SVC 7 is issued by a u-task that was link-edited with the NACPRIVILEGE option, the file class is specified as follows:

- P indicates file is allocated under a private account.
- G indicates file is allocated under a group account.
- S indicates 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)

SVC 7 has nine functions specified by the first byte of the function code, called the command byte, and has three 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, and
- 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.

TABLE 7-1 SVC 7 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 communications line (SVC 15) or terminal (SVC 1)	0=no function requested 1=change filename
4 (RP)	Reprotect	Changes the files current read/write protection keys to new protection keys	Changes the read/write protection keys of the communications line (SVC 15) or terminal (SVC 1) 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 particular line driver or terminal manager	0=no function requested 1=close an lu
6 (DL)	Delete	Releases reserved space on a direct access device	Releases a reserved line control block	0=no function requested 1=release reserved space
7 (CK)	Checkpoint	Copies buffered file data to a direct access device	Copies buffered file data to a logical terminal	0=no function requested 1=copy buffered file data

TABLE 7-1 SVC 7 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 VFC for devices that support VFC	Indicates file access method for data communications	00=terminal level (SVC 1) access 01=terminal level (SVC 1) access with vertical forms control 10=reserved 11=line level (SVC 15) access

TABLE 7-1 SVC 7 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 magnetic tape density being used	Indicates if buffered terminal or line access is being used	000=contiguous files or enable manual density selection on Telex magnetic tape drives. No action on other magnetic tape drives 001=extendable contiguous files 010=indexed files 011=nonbuffered indexed files 100=select 800 BPI NRZI density (STC and Telex drives only) 101=Select 1600 BPI PE density (STC and Telex drives only)

TABLE 7-1 SVC 7 FUNCTION CODE BIT DEFINITIONS (Continued)

BIT POSITION	FUNCTION	MEANING		BIT SETTING
		DEVICE/FILE	DATA COMMUNICATION	
13 14 15 (Continued)				110=select 6250 BPI GCR density (STC and Telex drives only) 111=communi- cations 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	Vertical forms control (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 time and date disk file was created and last written to.		X'FF00= re- turns time and date in sysgen format X'FF01'= re- turns time and date in mm/dd/yy; hr:min:sec format X'FF02'= re- turns time and date in dd/mm/yy; hr:min:sec format

TABLE 7-1 SVC 7 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 line control block (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 and 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
- Logical unit field
- 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 vertical forms control 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 function 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
- Logical unit field
- Logical record length field
- File size field

To allocate a temporary file, specify an allocate or an 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 TEMPFIL system 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 existing access privileges of the file remain unchanged. For example, if the device is assigned with read only privileges, a change to write access privileges is not permitted. Table 7-2 illustrates allowable access privilege requests relative to current access privileges, exclusive of other assignments to the same fd. The required parameter block fields for this function are:

- Bits 2, and 8 through 10 of the function code field
- Logical unit field

TABLE 7-2 ALLOWABLE ACCESS PRIVILEGE CHANGES

CHANGE FROM	CHANGE TO							
	SRO	ERO	SWO	EWO	SRW	SREW	ERSW	ERW
SRO	x	x						
ERO	x	x						
SWO			x	x				
EWO			x	x				
SRW	x	x	x	x	x	x	x	x
SREW	x	x	x	x	x	x	x	x
ERSW	x	x	x	x	x	x	x	x
ERW	x	x	x	x	x	x	x	x

LEGEND

x indicates allowable change

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. The required parameter block fields for this function are:

- Bits 3, and 8 through 10 of the function code field
- Logical unit 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:

- Bits 4, and 8 through 10 of the function code field
- Logical unit 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
- Logical unit field

When the lu is closed, all data in file 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 SVC 1 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)
- Logical unit 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 and/or important data has been written to a buffered file.

7.2.1.9 Fetch Attributes Function

The fetch attributes function sends to the SVC 7 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 SVC 7 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.

0(0)	1(1)	2(2)	3(3)
Command byte	Device codes	Error status	lu
4(4)	6(6)		
Device attributes	Physical record length		
8(8)	Volume name or device mnemonic		
12(C)			
	Filename		
16(10)			
20(14)	23(17)		
Extension	File class		
24(18)	File size		

```

SVC    7,parblk
.
.
.
ALIGN 4
parblk DB    0,0
        DS    1
        DB    lu
        DS    24 bytes for device attributes

```

Figure 7-3 SVC 7 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 (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 7-3 lists all supported attributes and corresponding masks.

TABLE 7-3 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 forwardspace filemark
14	X'0002'	Supports backspace filemark
15	X'0001'	Device dependent function

* Indicates the current access privilege.

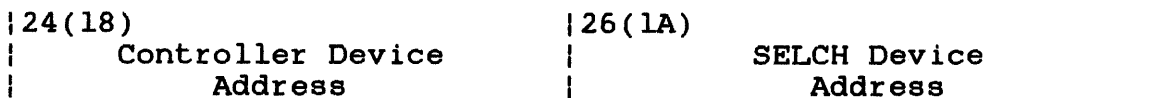
The physical record length field receives the logical record length of the file or physical record length of the device assigned to the specified lu; e.g., 80-byte record for card readers and 120- or 132-byte record for line printers. If the device has variable length records, a value of 0 is returned 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.

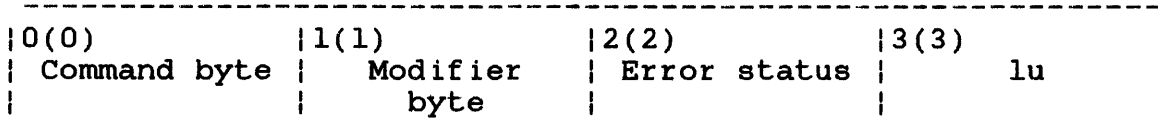
| For bare disk devices, the first two bytes of the file size field contain the controller device address if the bare disk is attached to a controller. If the bare disk is not attached to a controller, the first two bytes equal zero. The remaining two bytes of this field contain the SELCH device address 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)

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 SVC 7 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 SVC 7 Parameter Block Format and Coding for VFC Function

This parameter block must be fullword-boundary aligned and located in a task writable segment.

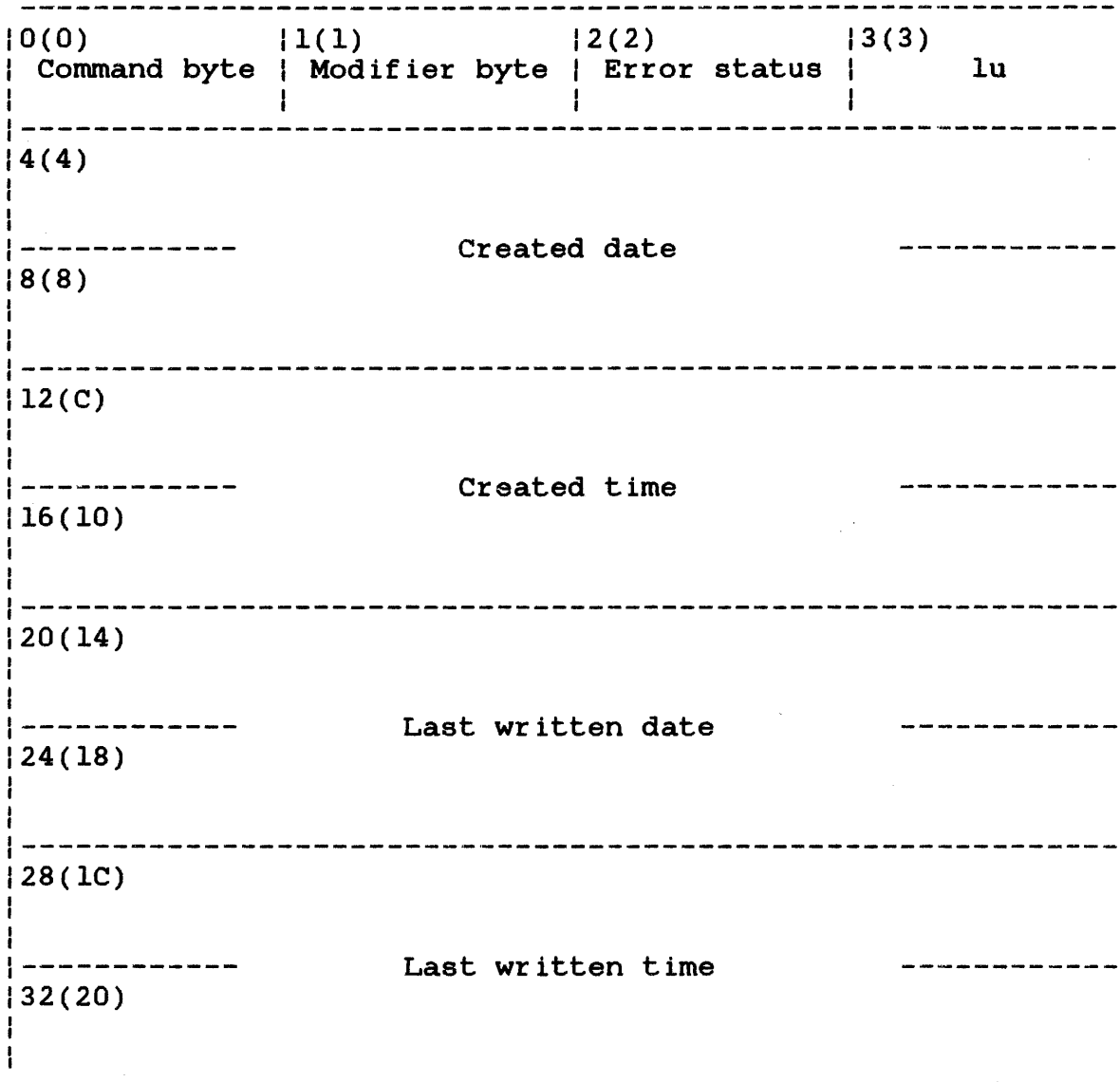
To turn on the use of the VFC function for a particular device, set the modifier byte to X'20'. To turn the function off, set the modifier byte to X'21'. The error status and lu fields are the same as for all SVC 7 services.

7.2.1.11 Fetch Time and Date Attributes from Disk Directory

The fetch time and date attributes function returns to the SVC 7 parameter block the date and time the disk file was created and last written to.

FUNCTION CODE	FORMAT
X'FF00'	Defined by 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'.



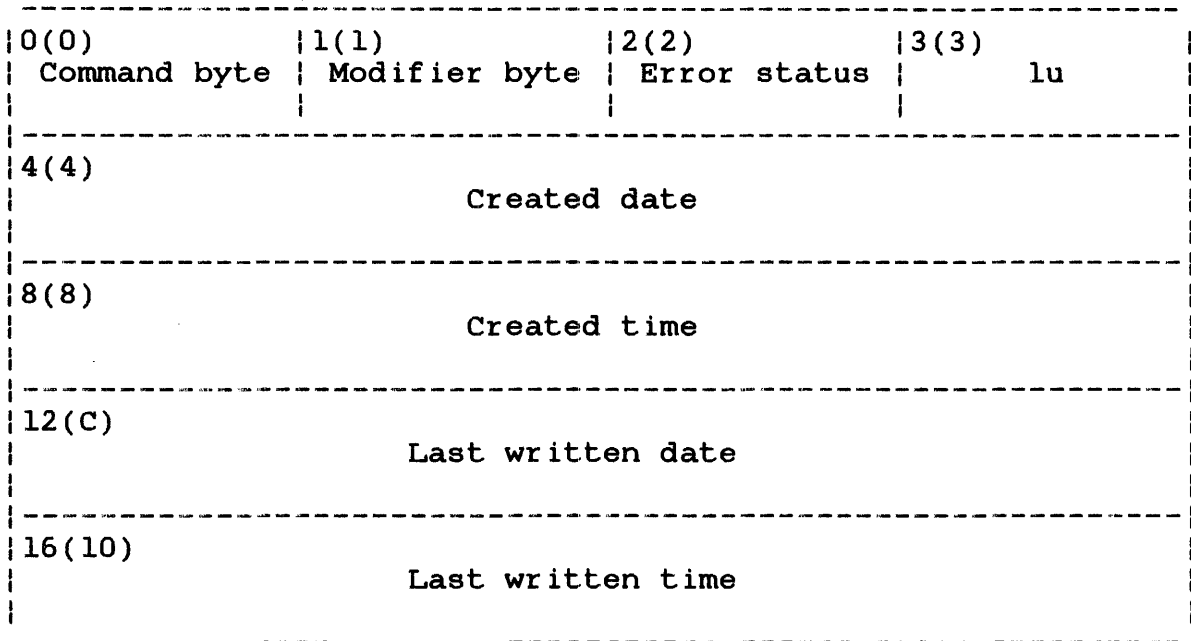
```

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 SVC 7 X'FF00', X'FF01', or X'FF02' Parameter Block Format and Coding for Fetch Time and Date Attributes Function

The SVC 7 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.



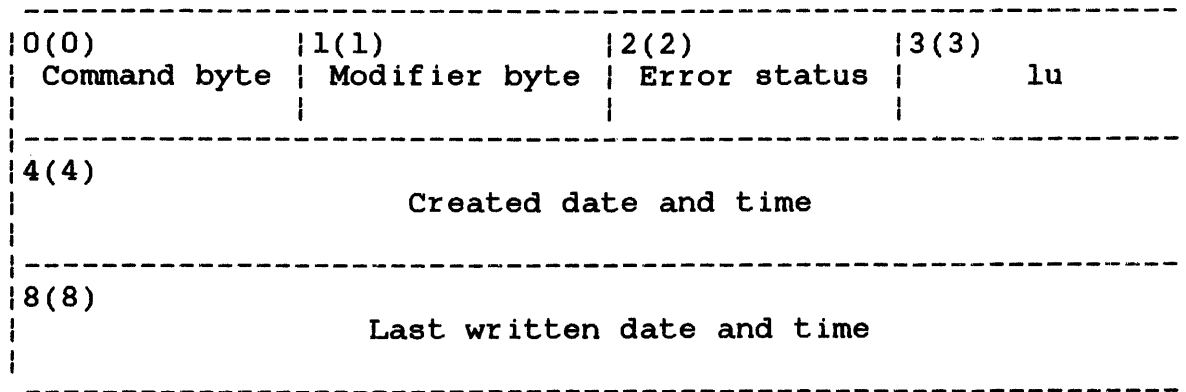
```

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 SVC 7 X'FF03' Parameter Block Format and Coding for Fetch Time and Date Attributes Function

The SVC 7 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 SVC 7 X'FF04' Parameter Block Format and Coding for Fetch Time and Date Attributes Function

7.2.1.12 Fetch Logical Attributes of Open File

The fetch logical attributes of open file function returns the following attributes for an open file to the SVC 7 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, and
- data blocksize of indexed, nonbuffered indexed, and extendable contiguous open files.

The function code for this SVC 7 is X'FF0A'. Figure 7-8 shows the SVC 7 parameter block fields for receiving the logical attributes of an open file.

0(0) Command byte	1(1) Modifier byte	2(2) Error status	3(3) LU
4(4) Total Logical Records			
8(8) Current Logical Records			
12(C) Index Blocksize		14(E) Data Blocksize	

```

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 SVC 7 X'FF0A' Parameter Block Format and Coding for the Fetch Logical Attributes of Open File Function

7.2.1.13 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 7-4 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 7-4 ACCESS PRIVILEGES DEFINITIONS

ACCESS PRIVILEGE	MEANING	BIT SETTING
Shared read only (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
Exclusive read only (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
Shared write only (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

TABLE 7-4 ACCESS PRIVILEGES DEFINITIONS (Continued)

ACCESS PRIVILEGE	MEANING	BIT SETTING
Exclusive write only (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
Shared read write (SRW)	Tasks can read from and write to the assigned file. This is the default.	100=SRW
Shared read exclusive write (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
Exclusive read shared write (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
Exclusive read write (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

7.2.1.14 Data Communications Access Methods

This 2-bit modifier field contains the access methods used by data communications. The access methods are listed in Table 7-5. See the OS/32 Basic Data Communications Reference Manual.

TABLE 7-5 DATA COMMUNICATIONS ACCESS METHODS 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 SVC 1.	00 =terminal level access 01 =terminal level access with vertical forms control
Line level access	This is the device dependent support of a communications line through the data communications driver. This is accomplished through SVC 15.	11 =line level access

7.2.1.15 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
- 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 SVC 7 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 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 will be 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 will not be assigned and status code X'09' will be returned in the SVC 7 parameter block.

7.2.1.16 Read/Write Key Fields (SVC7.RKY/SVC7.WKY)

The read/write key fields should contain the hexadecimal number indicating a file or the 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 7-6 lists the read/write protection keys.

TABLE 7-6 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 user task (u-task) can assign for protected access mode.

7.2.1.17 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.

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.

24(18)	Physical block size (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.

24(18)	26(1A)
Controller Device Address	SELCH Device Address

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.

7.2.1.18 SVC 7 Error Codes

If an error occurs during execution of an SVC 7 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 7-7 for the list of SVC 7 error codes.

TABLE 7-7 SVC 7 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.
6	Assign	Read/write protection keys do not match.
7	Allocate	Entire disk is currently assigned as ERW.
	Assign	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 specified new privileges are not compatible with existing ones. See Table 7.2.
		File not assigned ERW
	Reprotect	File not assigned ERW
	Delete	File assigned to another task (not closed).
	Rename	Read/Write Protection Keys do not match. File not assigned ERW.

TABLE 7-7 SVC 7 ERROR CODES (Continued)

ERROR CODE	FUNCTIONS AFFECTED	MEANING
8	Assign	Insufficient space for file control block (FCB) and buffers System space pointer or pointers have become corrupted.
	Close	
	Delete	
9	Assign	The lu is already assigned or device is offline. Magnetic tape drive does not support software density selection.
	Rename Reprotect Close Fetch attributes Change access privileges VFC	The lu is not assigned.
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.
E-7F	N/A	Reserved
80-FF	N/A	SVC 1 I/O error. See Tables 2-3 and 2-4.

CHAPTER 8
LOAD TASK STATUS WORD (TSW) SUPERVISOR CALL 9 (SVC 9)

8.1 INTRODUCTION

SVC 9 sets the initial TSW or replaces the current TSW located in the task control block (TCB) with a new user-specified TSW. The SVC 9 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.

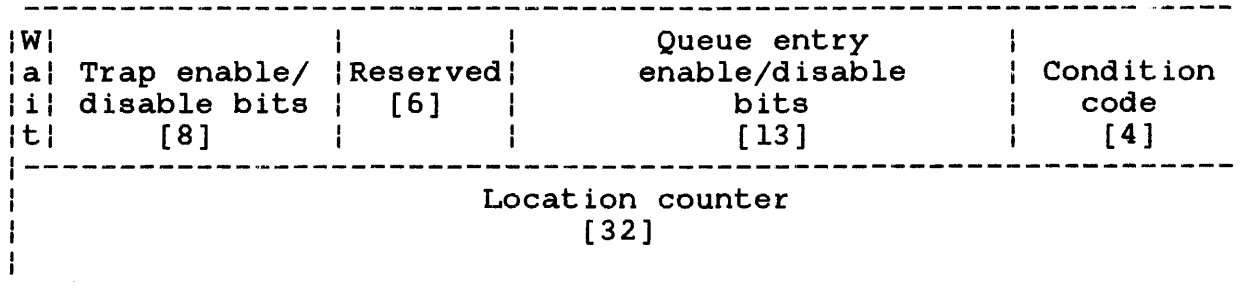
Storing TSW values into the user dedicated location (UDL) does not change the current TSW.

```

-----
| SVC 9 |
-----

```

8.2 SVC 9: 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 SVC 9 Parameter Block Format and Coding

This parameter block must be 8 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 a 7-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 7-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	is a 4-bit field stored in the processor condition code. For an explanation of the condition code, see the appropriate processor user manual. This field corresponds to the condition code 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 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, and
- condition code bits.

The second fullword of the TSW contains the location counter.

SVC 9 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 condition code setting in the status portion and the location counter in the location counter portion of the TSW. See Table 8-1 for the TSW bit definitions.

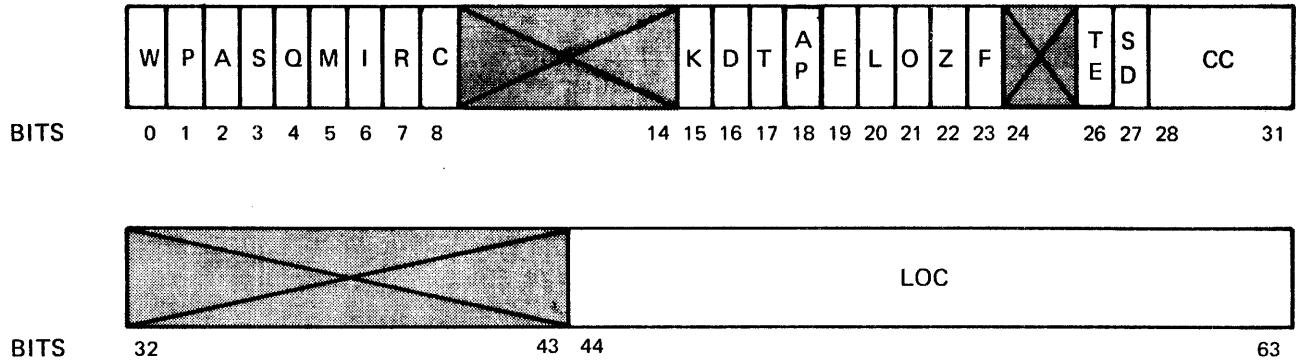


Figure 8-2 Task Status Word

TABLE 8-1 TASK STATUS WORD 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)	SVC 14 execution trap enable/disable (TSW.S14M) (Y'10000000')	Allows execution of SVC 14.

TABLE 8-1 TASK STATUS WORD 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.
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)	CPU-override status (TSW.CPOM) (Y'00800000')	Task is executed on the CPU and cannot be transferred to an APU for processing. (This bit applies only to tasks running on the Model 3200MPS.)
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 SVC 6 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 SVC 6 queue parameter function is directed to this task.

TABLE 8-1 TASK STATUS WORD BIT DEFINITIONS (Continued)

BIT POSITION	BIT NAME AND MASK	MEANING
18 (AP)	Queue entry on signal from APU (TSW,APTMM) (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 SVC 6 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 SVC 6 load and proceed function is executed and the load is completed.
21 (O)	I/O completion entry enable/disable (TSW.IOM) (Y'00000400')	An item is added to the task queue when an SVC 1 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 SVC 2 code 23 is executed and the interval has elapsed.
23 (F)	SVC 15 function SVC 1 buffer transfer completion (TSW.ITM) (Y'00000100')	An item is added to the task queue when an SVC 15 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 TASK STATUS WORD 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 SVC 6 send message function is directed to the task.
28-31 (CC)	Condition code	The condition code following SVC 9 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 SVC 9 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 SVC 9 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 SVC 9 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 location counter 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 SVC 9 loads a TSW that has zeros in the location counter field, execution resumes with the instruction following the SVC 9.

When SVC 9 loads a new TSW, the condition code of the new TSW becomes the current condition code. Any value ranging from 0 through 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 condition code is restored.

CHAPTER 9
OVERLAY LOADING SUPERVISOR CALL 10 (SVC 10)

SVC 10

9.1 SUPERVISOR CALL 10 (SVC 10)

SVC 10 is an internal call that provides for the automatic loading of overlays generated by Link. SVC 10 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 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 (ORT). 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, SVC 10 cannot perform the load function.

The overlay descriptor table entry (ODTE) is part of the ORT and represents the position in the ODT that contains processing information for the overlay to be loaded. Both the ORT and the ODT are OS system data structures and are defined in the system macro library.

9.2 MESSAGES

This 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)

Fields:

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 SVC 10 handler.

zzzzzz is the physical address of yyyyyy.

This 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)

Fields:

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 SVC 10 handler.

zzzzzz is the physical address of yyyyyy.

If the overlay load failure resulted from a malfunction of SVC 10, the task is paused with the current program status word (PSW) pointing to the SVC 10 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	User space violation of overlay start address
21	User space violation of overlay end address
22	Highest level OVL required by this SVC was not found.
23	OVL size is less than 10 bytes. Eight bytes for 2 fullword ORT entry pointers plus a 2-byte instruction (BR) is the minimum size for an overlay.
30	User space violation of ORT table address in ODT entry
*31	Pointers to ORT entries are unreliable. The address difference between these pointers must be 0 or an even multiple of 8 bytes.
*32	User space violation of ORT 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 (SVC 13)

10.1 INTRODUCTION

SVC 13 provides tasks running on the Perkin-Elmer Model 3200MPS System with the ability to:

- map APUs into the logical processor mapping table (LPMT),
- control the processing of all APUs within the system, and
- obtain processing status information on each APU in the system.

Table 10-1 lists the SVC 13 function codes that allow the task to access OS/32 system services for the Model 3200MPS multiprocessing system.

TABLE 10-1 SVC 13 FUNCTION CODES

FUNCTION CODE	MEANING
SVC 13 code 0	Fetch LPMT
SVC 13 code 1	Fetch APU status
SVC 13 code 2	Execute APU mapping option
SVC 13 code 3	Execute APU control option

The following sections outline the functions provided by each of the SVC 13 function codes. For more information on how to use SVC 13 in a multiprocessing environment, see the OS/32 System Level Programmer Reference Manual.

10.2 FETCHING LOGICAL PROCESSOR MAPPING TABLE (LPMT)

SVC 13 function code 0 can be used by all tasks in a Model 3200MPS System to retrieve information about the APUs within the system. This information includes:

- the maximum number of logical processing unit (LPUs),
- the maximum number of APUs, and
- the LPMT.

The LPMT contains one entry for each LPU. This entry specifies the APU to which the LPU is mapped.

10.3 FETCHING AUXILIARY PROCESSING UNIT (APU) STATUS INFORMATION

SVC 13 function code 1 can be used by all tasks in a Model 3200MPS System to retrieve information about the status of a specified APU within the system. This information includes:

- the number of LPUs mapped to the specified APU,
- the number of tasks in the ready queue of the specified APU,
- the status of the specified APU, and
- the names of the tasks associated with the specified APU. These tasks include the task actively executing on the APU, the task with control rights over the APU, the task with mapping rights over the APU, and all the tasks waiting in the APU's ready queue.

10.4 AUXILIARY PROCESSING UNIT (APU) MAPPING FUNCTIONS

Using SVC 2 code 2, a task linked with the auxiliary processing unit mapping privileges (APM) option can request mapping privileges for a specified APU in the system. These privileges allow a task to:

- mark the APU on,
- map the APU into the LPMT,
- remove all references to the APU from the LPMT, and
- mark the APU off.

10.5 AUXILIARY PROCESSING UNIT (APU) CONTROL FUNCTIONS

Using SVC 13 code 3, a task linked with the auxiliary processing unit control privileges (APC) option can request control privileges over a specified APU. These privileges allow a task to:

- initialize an APU that is waiting for power-up link check,
- stop APU execution,
- start normal execution (if stopped), and
- preempt the currently active task on an APU.

CHAPTER 11
USER SUPERVISOR CALL 14 (SVC 14)

SVC 14

11.1 SUPERVISOR CALL 14 (SVC 14)

SVC 14 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 SVC 14 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, SVC 14 is enabled; otherwise, SVC 14 is considered an illegal SVC.

When SVC 14 is executed, the operating system stores the effective program address of the SVC 14 second argument into the SVC 14 address pointer location in the task user dedicated location (UDL). A TSW swap then occurs, using the SVC 14 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 SVC 14; consequently, a task should not use SVC 14 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 SVC 14 task traps.

CHAPTER 12
DATA COMMUNICATIONS DEVICE DEPENDENT INPUT/OUTPUT (I/O)
SUPERVISOR CALL 15 (SVC 15)

SVC 15

12.1 SUPERVISOR CALL 15 (SVC 15)

SVC 15 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.

D	
Data communications	
access methods	7-27
buffered terminal manager	7-28
buffered terminals	7-31
device dependent I/O subsystem	12-1
subsystem	2-22
wait I/O	7-1
wait only	2-1
Data transfer requests	
conditional proceed	2-16
I/O proceed	2-15
queuing I/O requests	2-15
test and set	2-9
unconditional proceed	2-16
wait I/O	2-17
wait only	2-17
DATE command	3-39
Default task workspace	3-8
Delay start for SVC 6	6-42
Delete	7-5
Delete function	7-16
Density selections	7-5
manual	7-29
software	7-29
Device attributes	7-18
Device dependent status	2-21
Device independent status	2-19
Directed task	6-1
Direction SFUN.DOM,SFUN.DSM	6-10
DLIST xx command	2-39

E	
End of task	4-1
End of task codes	4-2
End task SFUN.ECM,SFUN.EDM	6-10
Error codes SVC6.STA	6-44
Executing SVC 2 code 17	3-76
EXPAND ALLOCATION	3-114
Extendable contiguous files	7-12
	7-13
	7-20
	7-28
	7-30
Extended function codes	
control operations	2-24
data transfer operations	2-26
gapless operations	2-43
local and remote communications	2-23
Extended options	
communication dependent	2-22
device dependent	2-22
device independent	2-22
field	2-42
magnetic tape devices	2-24
nonmagnetic tape devices	2-22
status codes for mag tape use	2-29

F	
FETCH ACCOUNTING INFORMATION	3-134
Fetch attributes	7-5
Fetch attributes function	7-17
FETCH DATE	3-38
Fetch logical attributes of open file	7-24
Fetch overlay	5-1
FETCH POINTER	3-20
Fetch time and date attributes	7-21
Fetching APU mapping functions	10-2
Fetching logical processor mapping table	10-2
File and device handling functions	7-1
File handling services. See SVC 7.	
File manager	7-1
File size field SVC7.SIZ	7-30
File types	7-5
contiguous	2-12
data communications	7-28
buffered terminal manager	7-28
extendable contiguous	2-12
indexed	2-12
nonbuffered indexed	7-28
Free buffer list queue	2-12
Free send data message buffers, receiving task	6-21
Freeze SFUN.FM	6-19
Function code SVC6.FUN	6-37
Function codes	6-5
command function requests	2-18
data transfer requests	2-5
file handling services	7-5
gapless mode data transfer	7-5
general service functions	2-35
intertask communications	3-1
	6-5

G	
Gapless I/O operations	2-32
buffer queues	2-39
device dependent status	2-38
device independent status	2-38
logical unit	2-37
standard function code format	2-35
Gapless mode parameter block format	2-32
General service functions. See SVC 2.	
GET STORAGE	3-7

Q
 Queue parameter SFUN.QM 6-33
 Queuing I/O requests 2-15

R
 Read key 7-3
 Read/write key fields
 SVC7.RKY/SVC7.WKY 7-29
 Read/write protection keys 7-29
 Receive logical unit SFUN.XRM 6-34
 Record lock bit 2-9
 Release SFUN.RM 6-39
 RELEASE STORAGE 3-13
 Rename 7-5
 Rename function 7-15
 Reprotect 7-5
 Reprotect function 7-15
 Rollable SFUN.RLM 6-40
 RTL. See run-time library.
 Run-time library 1-2
 3-7
 instruction 2-41

S
 Sample applications
 send data, receiving task 6-23
 send data, sending task 6-22
 send message, receiving task 6-31
 send message, sending task 6-31
 test and set 2-14
 unpack file descriptor 3-139
 3-140
 SCAN MNEMONIC TABLE 3-74
 Send data message buffers, sending task 6-17
 Send data SFUN.DB 6-17
 Send logical unit SFUN.XSM 6-34
 Send message SFUN.MM 6-25
 SET ACCOUNTING INFORMATION 3-123
 SET STATUS 3-16
 Single buffer chain 6-27
 Single buffer ring 6-27
 Sint SFUN.IM 6-36
 Software enabling of manual density selection 7-29
 SPT. See system pointer table.
 Standard function code format, gapless mode 2-35
 Start bit positions 29, 30, 31 6-40
 Start function for SVC 6 6-41
 Status codes
 device dependent 2-21
 2-29
 2-38
 device dependent, gapless 2-38

Status codes (Continued)
 device dependent, magnetic tape 2-29
 device independent 2-19
 2-38
 device independent, gapless 2-38
 fetch overlay 5-3
 file handling services 7-31
 intertask communications 6-44
 overlay loading 9-3
 supervisor calls 1-8
 SVC drives 7-29
 Structure macro 1-3
 Suspend SFUN.SM 6-17
 SVC 1-1
 error messages 1-6
 errors 1-6
 parameter block 1-2
 status codes 1-8
 SVC 1: I/O REQUESTS 2-1
 command function requests 2-17
 data transfer 2-5
 gapless I/O operations 2-32
 parameter block, gapless 2-32
 parameter block, standard 2-3
 SVC 2: GENERAL SERVICE FUNCTIONS 3-1
 contract allocation 3-116
 convert ASCII to binary 3-46
 convert binary to ASCII 3-23
 expand allocation 3-114
 fetch accounting information 3-135
 fetch date 3-38
 fetch pointer 3-20
 get storage 3-7
 interrogate clock 3-33
 interval wait 3-44
 log message 3-27
 move ASCII characters 3-82
 pack file descriptor 3-53
 pause 3-5
 peek 3-91
 release storage 3-13
 scan mnemonic table 3-74
 set accounting information 3-132
 set status 3-16
 time of day wait 3-41
 timer management 3-117
 unpack file descriptor 3-136
 SVC 2 code 1: PAUSE 3-5
 SVC 2 code 2: GET STORAGE
 option X'00' 3-9
 option X'80' 3-11
 SVC 2 code 3: RELEASE STORAGE 3-13
 SVC 2 code 4: SET STATUS 3-16
 option X'00' 3-16
 3-18
 option X'80' 3-16
 3-19
 SVC 2 code 5: FETCH POINTER 3-20
 SVC 2 code 6: CONVERT BINARY TO ASCII
 option X'00'+n 3-25
 option X'40'+n 3-25
 option X'80'+n 3-26
 option X'CO'+n 3-26

SVC 2 code 7: LOG MESSAGE		SVC 2 code 29: UNPACK FILE	
option X'00'	3-29	DESCRIPTOR	3-137
option X'20'	3-30	SVC 3: END OF TASK	4-1
option X'40'	3-30	SVC 5: FETCH OVERLAY	5-1
option X'60'	3-30	SVC 6: INTERTASK	
option X'80'	3-30	COMMUNICATIONS	6-1
option X'A0'	3-32	assign LPU SFUN.LPU	6-37
option X'C0'	3-32	change priority SFUN.PM	6-33
option X'E0'	3-32	connect SFUN.OM	6-35
SVC 2 code 8: INTERROGATE		delay start functions	6-42
CLOCK		direction	
option X'00'	3-34	SFUN.DOM,SFUN.DSM	6-10
option X'40'	3-36	end task	
option X'80'	3-35	SFUN.ECM,SFUN.EDM	6-10
option X'C0'	3-37	error codes SVC6.STA	6-44
SVC 2 code 9: FETCH DATE	3-38	freeze SFUN.FM	6-37
SVC 2 code 10: TIME OF DAY		function code SVC6.FUN	6-5
WAIT	3-41	load task functions	6-11
SVC 2 code 11: INTERVAL WAIT	3-44	nonresident SFUN.NM	6-39
SVC 2 code 15: CONVERT ASCII		nonrollable SFUN.NRM	6-40
TO BINARY		parameter block	6-3
option X'00'	3-46	queue parameter SFUN.QM	6-33
	3-47	receive logical unit	
option X'40'	3-46	SFUN.XRM	6-34
	3-49	release SFUN.RM	6-39
option X'80'	3-47	rollable SFUN.RLM	6-40
	3-50	send data SFUN.DB	6-17
option X'C0'	3-47	send logical unit	
	3-52	SFUN.XSM	6-34
SVC 2 code 16: PACK FILE		send message SFUN.MM	6-25
DESCRIPTOR		sint SFUN.IM	6-36
option X'00'	3-62	start bit positions 29,	
option X'10'	3-64	30, 31	6-40
option X'20'	3-66	start function, SFUN.SIM	6-41
option X'40'	3-63	start options, SFUN.SOM	6-41
option X'50'	3-65	suspend SFUN.SM	6-17
option X'60'	3-67	task resident SFUN.HM	6-16
option X'80'	3-68	thaw SFUN.TM	6-36
option X'C0'	3-71	transfer to CPU SFUN.TC	6-38
privileged task options	3-72	transfer to LPU SFUN.TL	6-38
SVC 2 code 17: SCAN MNEMONIC		disconnect SFUN.UM	6-37
TABLE	3-74	wait status field	
SVC 2 code 18: MOVE ASCII		SVC6.TST	6-43
CHARACTERS		SVC 7: FILE HANDLING SERVICES	7-1
option X'00'+n	3-83	access privileges	7-26
	3-84	allocate	7-11
option X'80'+n	3-83	assign	7-12
	3-86	change access privileges	7-14
SVC 2 code 19: PEEK		checkpoint	7-17
option X'00'	3-91	close	7-16
option X'01'	3-97	data communication	
option X'02'	3-103	access method	7-27
option X'03'	3-105	delete	7-16
option X'04'	3-111	error codes	7-31
SVC 2 code 20: EXPAND		fetch attributes	7-17
ALLOCATION	3-115	fetch logical attributes	7-24
SVC 2 code 21: CONTRACT		fetch time and date	
ALLOCATION	3-117	attributes	7-21
SVC 2 code 23: TIMER		function code field	
MANAGEMENT		SVC7.OPT	7-5
option X'00'	3-117	rename	7-15
option X'10'	3-129	reprotect	7-15
option X'20'	3-127	temporary	
option X'40'	3-123	allocation/assignment	7-13
option X'80'	3-120	vertical forms control	7-20
SVC 2 code 24: SET		SVC 9: LOAD TASK STATUS WORD	8-1
ACCOUNTING INFORMATION	3-133	SVC 10: OVERLAY LOADING	9-1
SVC 2 code 25: FETCH		SVC 13: APU CONTROL	10-1
ACCOUNTING INFORMATION	3-135	SVC 14: USER SUPERVISOR CALL	
		14	11-1
		SVC 15: DEVICE DEPENDENT I/O	12-1

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, ect.

1. Publication number _____

2. Title of publication _____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if necessary.

4. Was the publication easy to understand? If not, why?

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