DATABOARD 4680®

OS.8 MT PROGRAMMING MANUAL

SATTCO AB DALVÄGEN 10 S-171 36 SOLNA SWEDEN PHONE 08/ 734 00 40 TLX 11588

FOREWORD

This manual describes OS.8, the Multi-tasking Real-time Operating System, and deals specifically with revision 2.50 and higher revisions. It is designed as a reference manual for programmers in the process of designing and programming tasks to run under the Operating System, or involved in System maintenance.

This publication is divided into two separate manuals and four appendices:

OS.8 PROGRAM REFERENCE MANUAL (PRM)

provides information for the user programmer to make full use of the wide range of capabilities provided by the Operating System when constructing and maintaining user programs.

OS.8 SYSTEM LOGIC MANUAL (SLM)

describes the internal structure of the Operating System, and contains information necessary to write device handlers.

Appendix A contains glossary and shortenings.

Appendix B holds the crash codes.

Appendix C specifies the error codes.

Appendix D describes the differances from previous revisions.

OS.8

PROGRAM REFERENCE MANUAL (PRM)

CONTENT

CHAPTER 1

INTRODUCTION

1 INTRODUCTION

This manual is designed as a reference manual for programmers in the process of designing and programming tasks to run under OS.8.

Use of this manual requires that the reader is familiar with the features, functions and conventions of the DataBoard-4680 system from the user´s point of view as documented in:

- DataBoard-4680 System Manual.
- DataBoard-4680 Software Catalog.
- DataBoard-4680 Assembler Manual.
- DataBoard-4680 OS.8 Operator´s Manual.

The major features in this revision, R2.50, are:

- Dynamic memory management and support of discontiguous memory.
- Disc management, including directory pre-allocation and OS images in files.
- On line resource handling.
- Overlay handling.
- Symbiont task and task devices.

This publication is divided into six more chapters as folows:

Chapter-2 gives a general system overview of OS.8 including hardware requirements.

Chapter-3 describes shortly the system operation.

Chapter-4 discusses task structure, priorities, task-handled events.

Chapter-5 is a guide to the services available through supervisor calls (SVCs).

Chapter-6 describes the functional aspects of the device and the direct-access file support.

Chapter-7 is a guide for using SVC 2 calls for command processing functions.

CHAPTER 2

SYSTEM OVERVIEW

2.1 INTRODUCTION

OS.8 is a Real-Time Multi-Tasking Operating System for the DataBoard-4680 System that increases programming and operation efficiency, yet requires a minimum of storage and computing time. It can monitor both batch processing and real-time applications such as process control, production control, plant supervising, communication protocols, program development etc.

OS.8 is a disc-based operating system, but may be used as a memory only system located in PROM. It supports up to 64 kilobyte of main memory (256KB in R3.00), both contiguous and discontiguous.

Built-in functions include system control via the Terminal Management, interrupt handling and I/O servicing. Data file management features are provided for any system equipped with direct-access storage media. The file management technics includes two types of file structures, and a disc directory structure such that each disc contains complete information concerning all its existing files.

Facilities are provided for supporting up to 255 tasks running concurrently. Tasks can reside permanently in memory or on a mass storage device, such as disc and discettes, and be brought into memory for execution, which is carried out on priority basis. A task of higher priority will interrupt a task of lower priority either through a hardware interrupt or via a programmed request. When the task of higher priority is finished, the operation of the lower task is resumed.

The operating system has a modular design and only the modules which the user needs have to be included in a software installation, thus saving memory space. The system can easily be generated together with user programs, and be placed in PROM, for simple handling of final applications. The system has a very low overhead and reentrant code is used to a very large extend.

The programmer communicates with the system through standardized requests and via commands from the terminal device. Input and output functions are carried out by driver programs, one for each type of device. Devices can easily be added and deleted on line.

2.2 MINIMUM HARDWARE REQUIREMENTS

DataBoard-4680 system equipped with a Z80-CPU.

Minimum 48KB main memory.

Interval Clock.

Optional peripherals are:

- Terminal device together with the Terminal Manager.
- Discs and drums from 80 Kb up to 400 Mb.
- Magnetic tapes and cassettes.
- High speed line printer.
- High speed paper tape reader and punch.
- Card reader.
- Modems and dial-up units.
- Digital and analog inputs and outputs.

2.3 FEATURES

Outstanding features of OS.8:

KERNEL:

PROMmable, all data structures are created at cold start (64KB version)

Priority allocation of the whole computer system, including devices, execution time, memory allocation, etc.

Dynamic memory handling that supports up to 64KB (256KB in R3.00) of main memory.

Calendar and time-of-day are normally maintained by the system, and interval timing is available to user tasks.

The number of tasks in memory at any time may be as great as 255, limited only by the amount of memory available and by system generation considerations.

Tasks are scheduled by priority with 255 distinct levels. An optional time slice scheduler allows tasks of equal priority to share processor time.

Low overhead. A minimum of searching is required to find the highest priority task.

There are 16 system priority levels, 8 for hardware and 8 for software services. The system will support up to 255 interrupt driven devices. Response time for higher level interrupts is less than 200 uS.

INPUT/OUTPUT:

Input and output operations are device independent, allowing re-assignment without having to alter existing software.

Devices and drivers can easily be added and deleted on line wich allows the user to dynamically add new devices without a system generation.

TASKS:

Tasks and overlays are loaded into any free memory area by a relocating loader.

Multiple applications programs can operate concurrently through the use of interleaving techniques.

Task need not to be totally memory resident, but may be segmented and overlaid from any mass storage device.

Task queue. Event-related information is maintained for each task within its own task queue.

Task may request the activation and execution of other tasks, and may pass parameters to one another.

Task may take traps on the reciption of these parameters and upon completion of all type of requests.

FILE MANAGEMENT:

Comprehensive file management facilities are provided on direct-access devices; contiguous and indexed file structures are provided for save and efficient use of the disc.

No hardware dependent file organization, any combination of disk types may be used at the same time.

Theoretical unlimited file size, disk addresses are 32 bits.

Hashed directory structure for faster directory search.

User readable directories.

Twelve character file names plus file type designator.

Cache memory type of sector buffering, to keep down the number of disk accesses.

Element File capabilities, keeps down the number of entries in Master File Directory. Element Directories speeds up directory searching.

Element Files used for example in source module collection. Module name is then expressed as PROJECT.MODULE. Backup of an Element File Directory will backup all elements in the directory which simplifies bookeeping of source modules since only one name is required.

Byte random access, a file may be treated as a stream of bytes.

Variable or fixed length files.

File access lockout in multi user environment.

Files can be accessed logical (byte access) or physical (256 bytes blocks).

Files can be assigned by more than one program at each time unless the access attributes aren´t violated.

This figure shows the principal interactions between the major groupings of the Operating System. For clarity, many minor interactions between these module groupings are not shown. For a more detailed explanation of system interactions, the reader is referred to the DataBoard-4680 OS.8 System Logic Manual.

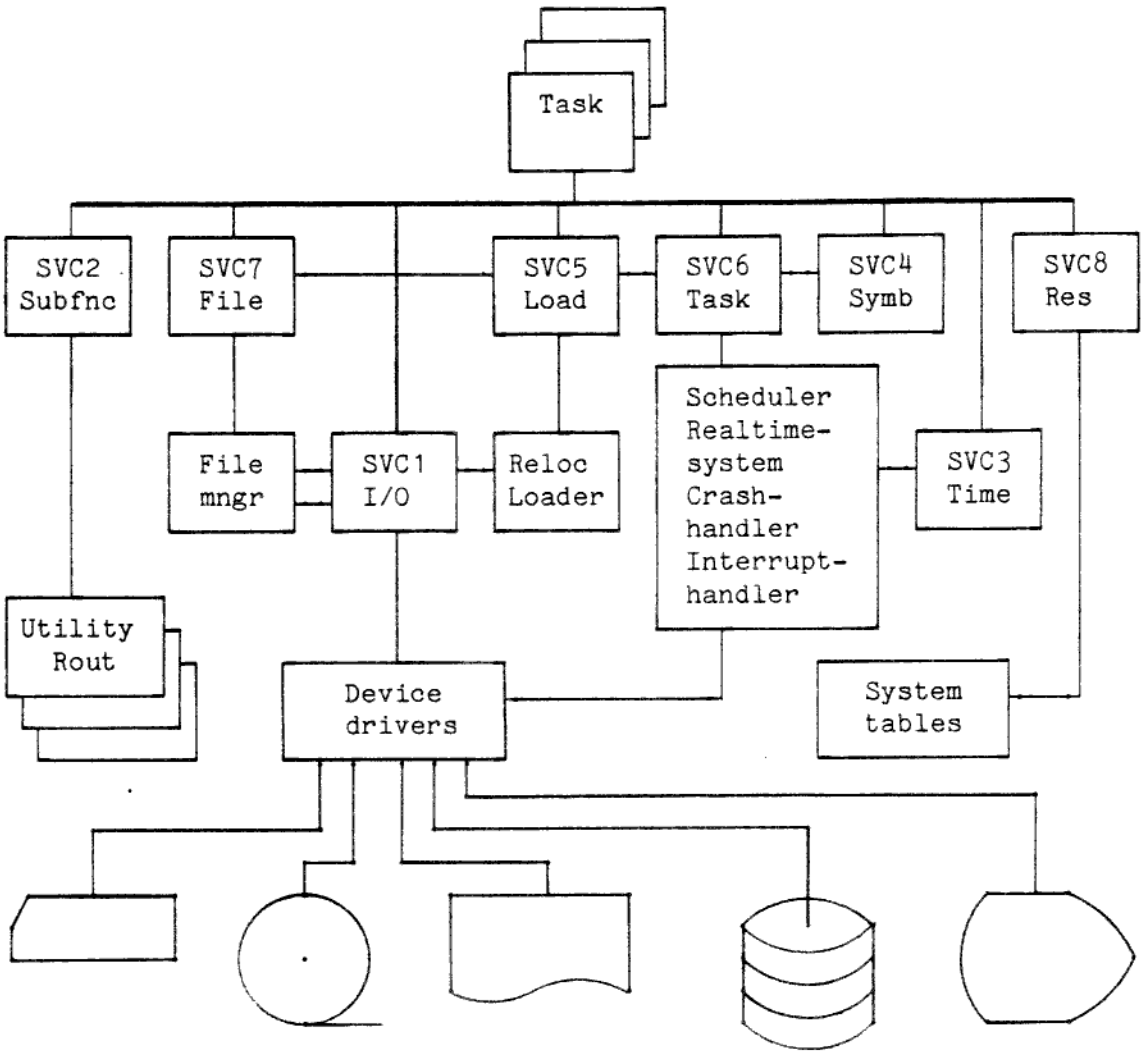


Figure: OS.8 Functional Block Diagram

CHAPTER 3

SYSTEM OPERATION

3.1 SYSTEM GENERATION

OS.8 ought to be tailored to the specific configuration it is to support. This is normally done through the command stream, when linking the the libraries of system object modules, to produce an absolute OS.8 load module. It is also possible to generate an OS.8 together with user application programs, that could be placed in PROM. The OS.8 System Logic Manual contains more detailed information about system generation.

3.2 SYSTEM START UP

OS.8 is normally loaded from a mass storage device by the Bootstrap Loader, but can also be loaded from a paper tape reader, or be resident in PROM. On completion of the load, control is normally transferred to the Terminal Manager Task which prints

```
DOS.8MTM Rx.yz
```

on the terminal device, where 'x' is the release number and 'yz' is the update number, and issues the command promt '-'. The system is now ready to accept commands. If the OS.8 is generated without the Terminal Management, control is transferred to a task specified by the user.

3.3 SYSTEM SHUT DOWN AND RESTART

In a disc-based system it is necessary that the system should be shut down or restarted in an orderly fashion, to assure the integrity of the discs in use. Before shutting down or restarting the system, the operator should cancel and delete all tasks, and close all discs. The system may now be restarted. If the system crashes, it should NOT be restarted, it must be reloaded.

3.4 SYSTEM CRASH

When the system determines that futher execution may cause system or user data to be destroyed, the system crash handler is entered. As an option, a crash dump sequence may be included into the system. Some of the conditions causing a system crash are:

- Invalid data in system structures.
- Illegal interrupts.

For a complete list of the system crash codes and their meanings refer to Appendix D. The OS.8 must be reloaded after a crash !

CHAPTER 4

TASKS

4.1 INTRODUCTION

The fundamental unit of work in OS.8 is the task. A task may consist of a single program, or it may include a main program and a number of subroutines and overlays. Tasks may be permanently resident in memory, or they may be loaded as required. Tasks are referred to by TASKID which is associated with the task at load time. The number of tasks allowed in memory at one time is limited at system generation time. Each task is controlled through a Task Control Block (TCB).

4.2 PREPARATION

A task must be prepared by processing the object module with the OS.8 Task Establisher. Once established, the task is loaded by the resident loader via the LOAD operator command or SVC6.

The process of establishing a User task produces a relocatable load module of the task. The task references data and instructions as if the task were loaded at location 0 in memory. When the task is loaded into the OS.8 system, the relocating loader is used to provide automatic relocation from task space addresses to real physical addresses, thus allowing a task to be loaded into any memory segment large enough.

When creating a PROMmed installation, the user tasks may be linked together with the OS.8, or linked separate, to produce the final application.

4.3 PROTECTION

Since there are neither any Memory Management nor any Memory Protect hardware present, any task can access memory outside their boundaries. The only protection present is a software check of each task's stack.

4.4 STATUS

A task in memory may be in any of five states. These are:

- Current, the task executing instructions.
- Ready, to become the Current task.
- Waiting, for an event.
- Paused
- Dormant, not started.

Additionally, task are classified as either resident or non-resident. By definition, a task which is resident is not deleted when it completes execution. A non-resident task which goes to end-of-task (EOT) is deleted from the system. A task may be made resident at task establish time or at run time by the operator or a task.

The Current task is the task executing instructions. Only one task may be in this state at any given instant in time. All other tasks in memory are in one of the other four states, and may become the Current task depending on circumstances.

A Ready task is one which has no obstacles to become the Current task. It is eligible to be dispatched (i.e., become Current) whenever it becomes the highest priority Ready task.

A task in the Wait state is one which may not become Ready until some specific circumstance has occurred. Among the possible Wait states are:

Connection wait	Waiting for I/O to start
I/O wait	Waiting for I/O completion
Time wait	Waiting for an interval or time of day
Trap wait	Waiting for a task-handled event
Task wait	Waiting to be released by another task

A Paused task is one which may not execute until explicitly continued either by the console operator or by another task.

A Dormant task is one which may not execute until it has been explicitly started, either by the console operator or by another task. When a resident task goes to EOT it enters the Dormant state. When any task is loaded, it enters the Dormant state after load-complete, and remains in this state until started.

4.5 PRIORITY AND SCHEDULING

OS.8 recognizes 256 priority levels from a high of 0 to a low of 255. Of these levels, 1-255 are available to user tasks while 0 is reserved for system's use. Each task has two priorities associated with it:

- Task Priority
- Dispatch Priority

Task priority is the priority currently assigned to the task; it is set at task establish time and may be modified by operator command or SVC-6.

Dispatch priority is the priority set up by the system to determine the order in which ready tasks are serviced. Normally, a task's dispatch priority is the same as its task priority, but may be raised temporarily if the task is using a system resource required by a higher priority task.

Two types of scheduling algorithm are available. Tasks may be scheduled in strict priority order or time-sliced, with either a global slice limit or an own limit, within priority. In the former case, if two tasks of equal priority are started, a task remains active until it relinquish control of the processor. Care should be taken in assigning priorities so that tasks which do not frequently relinquish control of the processor do not inadvertently lock out other tasks. A task may relinquish control in one of the following ways:

It is Paused or Cancelled by the operator or by another task.
A higher priority task becomes ready because of some external event.
It executes an SVC that places it in Wait, Pause or Dormant state.

Rather than scheduling on strict priority basis, task may be time-sliced within priority. This option allows the user to ensure that tasks of equal priority receive shares of processor time.

The time-slicing option may be enabled and disabled by an operator command. Refer to DataBoard-4680 OS.8 Operator´s Reference Manual.

When a task becomes ready, it is queued on a round-robin basis behind all ready tasks of equal priority.

4.6 TERMINAL MANAGER

The terminal operator interface is provided by a task called Terminal Manager. The Terminal Manager interprets and executes all commands; it also performs all I/O requests to the console device. The Terminal Manager task is normally included as a task of OS.8 and cannot be cancelled. When the Terminal Manager is the only task in the system, except for a dormant task, the system is said to be quiescent.

The OS.8 Operator´s Manual describes in detail the commands and procedures related to communication with the Terminal Manager.

4.7 SUPERVISOR CALLS (SVC´s)

The program interface to the operating system is provided through Supervisor Call (SVC) instructions. SVC instructions are executed by programs to request OS.8 services. The parameters associated with the request are passed to the OS.8 in a parameter block. Most of the services provided by the Console Manager are performed with SVC instructions, thus making these services available to user tasks. Chapter 5 describes the individual SVC instructions and their associated parameter blocks in detail.

4.8 EVENT QUEUE

OS.8 provides a facility at the task level known as the task-handled trap facility. This allows a task to handle asynchronus events. Event related information is maintained for each task within its own event queue. An event queue is a linked list of nodes.

A trap service occurs whenever the task is in Trap-wait state. If a task is in any Wait state other than trap wait, a trap does not actually occur until the task has left that wait state. Several trap-causing conditions may occur before the first trap is handled by the task. Therefore, the event queue facility is provided to allow for queuing of event information during periods when the task is unable to service a trap.

The following trap-causing conditions cause an item to be added to the event queue:

- Addition of a parameter to the event queue.
- Completion of any queued no-wait request.
- Requests to own task-device.
- SVC 1 requests to own TCB.

4.9 SYMBIONT AND TASK DEVICES

The OS.8 recognizes special "imaginary" devices, Task Devices, that may be added to the system. These devices have all the characteristics of a real interrupt driven device, being able to be assigned to any task and having I/O requests queued for them. Whenever these requests are queued, another task called a Symbiont is "activated" instead of a device driver. In this case the symbiont task "initiates" the operation and handles the completion of the request just as a device driver would do. Normally, the symbiont operates at a task priority level higher than any task that may call it.

However, being an ordinary task, the symbiont has more freedom than a device driver has, since it can execute SVC instructions. Since the symbiont simulates a real device driver, it can be written in such a way that it processes requests of a single task device, or it processes requests of several simular or unrelated devices.

Symbionts may be used to simulate devices, to spool devices (impose intermediate disc buffering), or to impose special formatting, communication protocols, or data conversions on existing interrupt driven devices. These techniques can be implemented in such a way that they are "transparent" to the task which uses them.

CHAPTER 5

SUPERVISOR CALLS

5 SVC - SUPERVISOR CALLS

Supervisor calls (SVC) are used in programs to request the operating system to perform operations such as data transfer, file handling, task manipulation, timer coordination, device control and subfunctions like text-processing.

All SVC´s are connected to a parameter block where all parameters are specified to perform the requested function. The parameter block must be in a writable segment.

SVC´s are written in assembly language and they always consist of a restart 7 instruction followed by a 1 byte argument, and a 2 bytes parameter. The argument specifies the request type, and the parameter is normally an address to a parameter block. Parameter values lower than 256, means that a value in CPU-register pair should be used.

When the SVC-instruction is executed, all CPU-registers are stored on the task´s stack. This because of the fact that the operating system needs the registers, and to allow the task to keep the register contents at return after a SVC. During a SVC the system operates as a subroutine of the calling task, mostly with the callers priority.

At return from the SVC all CPU-registers are unchanged, except the current ´A´-register and the condition codes. The ´A´-register contains the return status, and the condition code is set to true zero and false carry when no error conditon exists, and set to false zero, and true carry on error.

When an illegal SVC is done, a message is logged on the system console and the task will be paused. To resume the execution, the SVC must be corrected, or the task must be canceled.

If the task has specified system recovery, then a message is logged and the task will be paused for every abnormal return status. When the task is told to continue, the SVC that produced the error is repeted. When system recovery has been specified, all error handling is done in a system dependent routine inside the operating system.

The SVC-sequence for most requests are:

```
        SVC   ARG,PARBLK      Parameter block ptr in the instruction.
         -                    Return point.
         -
        SVC   ARG,(rp)        Parameter block ptr in a register pair.
         -                    Return point.
         -
PARBLK  EQU   *               Parameter block.
        DB    0               Function code.
        DB    0               Return status.
        DB    0,0...          Depends on the type of request.
```

PARAMETER BLOCK

The parameter block contains the detailed information required to perform the requested function. All parameter blocks always contains a field that holds the function code and another field where the return status is stored by the system. The content of the following fields depends on the service requested. The general structure of a parameter block is:

(0) S0.FC Function code	(1) S0.RS Return status	(2-) Depends on the SVC-function

Each field within a parameter block is identified by its name and a descriptive title. Offsets are given in the form (DD), where DD is the offset in decimal. The name of the field is in the form Sn.FFFFF, where <n> is the SVC-argument and <FFFFF> is the field name.

Certain fields contain flag bits to denote information. These bits are defined by a name of the form SnF.BBBB, where <Sn> refers to the name of the parameter block, <F> refers to the field and <BBBB> identifies the function of the flag bit.

In order to make a program more readable, the programmer should use these symbolic definitions. The definitions are available as an object library to be used at program linking time.

FUNCTION CODE, S0.FC

In general, the request is defined by the logical 'OR' of the function code bits. The '0' setting of each bit is valid for all requests. If any invalid '1' setting of a bit is specified, the request is rejected as an illegal function. There are 3 common function codes:

S0F.WAIT	0	Wait for completion.
S0F.TST	40Q	Test request.
S0F.CAN	41Q	Cancel all previous requests.

There is also 2 common bits in the function code:

S0F.NW 100Q Wait-proceed bit. Indicates the action to be taken after the request has been initiated.
0-Wait. The task is to be put into wait until the request is complete.
1-Proceed. The control is to be return to the task after initiation of the request.

S0F.PRO 200Q Unconditional proceed bit.
0-The task is to be put into connection wait until the requested resource is free. At that time the request is processed.
1-The request is to be rejected if the requested resource is not free.

RETURN STATUS, S0.RS

The system returns the status of the requested function in the return status byte. This status byte is set to indicate the general type of error that occured. If there was no error, it is set to zero. The return status is presented in the form NNX, where NN is the SVC-function and X is the error code. The status has 9 common values:

S0S.OK	0	No error.
S0S.EON	1	End of nodes in the task.
S0S.IFC	2	Invalid function.
S0S.PRO	3	Can't connect to the resource.
S0S.OFFL	4	Resource off line.
S0S.PRES	5	Not present in this system.
S0S.NYET	6	Not yet implemented function.
S0S.CAN	7	Request is canceled.
S0S.SVC	8	Invalid SVC function.

WAIT FOR COMPLETION, S0F.WAIT

A wait for completion request (function code 0) causes the task to be placed into wait until the completion of a previous proceed request to the specified resource. If there is no outstanding request by the task to the specified resource, control is returned immediately. This call make use of the function code and status fields of the parameter block. Illegal resource is the only error status returned by this call. The status of the request being tested is returned in the parameter block associated with the original proceed call and not in the wait for completion call's parameter block.

CANCEL REQUEST, S0F.CAN

A cancel command request (function code 41Q) is used to terminate a request wich has previously been issued. This is especially useful on an interactive device. If cancel request is not used, an outstanding request must be satisfied before any other request can be started on a resource.

When a cancel command is issued, the operating system schedules the previus request for termination. The actual termination is asynchronous to the cancel request. When the request completes, the task receives a trap, if enabled. The parameter added to the trap queue is the address of the original parameter block, not the address of the cancel parameter block. Alternatively, the task may sense the completion of the request with test request or wait for completion request.

It is possible that a previous proceed request has gone to completion, at the time the cancel call is done, without being serviced, i.e. has been added to the event queue. In that case, the return status in the proceed request will not contain cancel return code.

Two parameter blocks are involved in the cancel processing. The first is the original parameter block specified by the user when the request was initiated. The second is the command function parameter block which is requesting the cancel. These should not be the same

parameter block. As the result of cancel command, status is returned to both of these parameter blocks as indicated below:

1. Cancel parameter block:
   000 The requested termination has been scheduled.
   007 No request on-going for the task on this resource.
   011 Resource not assigned.

2. Original parameter block:
   007 Request is canceled.

TEST REQUEST, SOF.TST

A test request (function code 40Q) returns with a return status of 0 if there is no outstanding proceed request to the specified resource by the task. If there is an outstanding proceed request, the call returns a status of 377Q.

WAIT/PROCEED, SOF.NW

A wait call requests the operating system to suspend the calling task until completion of the requested operation.

Once a request has been initiated (that is, the specified resource is free), any proceed request causes control to be returned to the task so that the task may execute concurrently with e.g. the data transfer. The return status is not set until completion of the request, except for illegal function, and illegal resource which are rejected before initiation. Every no-wait request to a queued resource will be added to the event queue of the task, if enabled. The return status of the request may be checked by:

- Monitoring the return status field in the parameter block.
- Issuing a wait for completion request to the same resource.
- Taking a task handled trap on completion.

UNCONDITIONAL PROCEED, SOF.PRO

Unconditional proceed is used when a task does not wish to wait for the requested operation. Requests are coordinated by the system so that only one request may access an exclusive resource at a time. If unconditional proceed is not requested, and the specified resource is in use at the time of the request, the calling task is suspended by the operating system until the resource is free. At that time, the request is initiated.

If Unconditional proceed is specified and the resource is in use, the request is rejected. Return status is set to '3' and the condition code is set to nonzero. The calling task may then retry the request at later time. If the specified resource is not in use, the setting of unconditional proceed has no effect on the request.

5.1 SVC1 - INPUT/OUTPUT REQUEST

SVC1 is used by a task to perform all general purpose I/O requests.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S1.LU Logical unit	(3) S1.TS Term. status
(4) S1.BAD Buffer start address		(6) S1.BSZ Buffer size in bytes	
(8) S1.BCNT Byte count at completion		(10) S1.RND Random address -	
(12) - if applicable.			

S0.FC, Function Code

	..0...xx	Read-write bits.
S0F.WAIT	..0...00	Wait for completion.
S1F.READ	..0...01	Read request.
S1F.WRIT	..0...10	Write request.
S1F.WRD	..0...11	Write with read check (device dependent).
	..0.xx..	Format bits. These bits indicates the type of data formatting requested.
S1F.IASC	..0.00..	Image ASCII.
S1F.FASC	..0.01..	Format ASCII.
S1F.IBIN	..0.10..	Image binary.
S1F.SPEC	..0.11..	Special.
	..0x....	Sequential-random bit.
	..00....	Sequential. The next logical record is to be accessed.
S1F.RND	..01....	Random. The logical record specified by the random address field is to be accessed.
	..1xxxxx	Command codes. Codes not specified are resource dependent.
S0F.TST	..100000	Test request.
S0F.CAN	..100001	Cancel request.
S1F.FR	..100010	Forward record.
S1F.FF	..100011	Forward file.
S1F.WF	..100100	Write filemark.
S1F.BR	..100101	Back record.
S1F.BF	..100110	Back file.
S1F.RW	..100111	Rewind.
S1F.ATTN	..101000	Attention.
S1F.FEOF	..101001	Fetch end-of-file position.
S0F.NW	.x.....	Wait-proceed bit.
S0F.PRO	x.....	Unconditional proceed bit.

S0.RS, Return Status

	0-9	Common codes.
S1S.LU	10	Illegal or unassigned LU.
S1S.AM	11	Access mode missmatch.
S1S.TOUT	12	Time-out.
S1S.DWN	13	Device off line.
S1S.EOF	14	End of file.
S1S.EOM	15	End of media.
S1S.RER	16	Recoverable or parity error.
S1S.UNR	17	Unrecoverable, read-write failed.
S1S.RND	18	Invalid random address.
S1S.NRND	19	Non-existent random address.

ASCII, Definition
Seven bit data, with the most significant bit in the byte cleared. Data values between 40Q and 177Q. All other are control information.

BINARY, Definition
Eight bit data.

S1F.FASC, Formatted ASCII
ASCII data with space compress, the most significant bit in the byte is set, and seven bit space counter. The termination byte in the buffer is zero.

S1F.IASC, Image ASCII
ASCII data without space compress, and no termination character in the buffer.

S1F.IBIN, Image Binary
Transfer of eight bit data bytes, without any formatting.

S1F.SPEC, Special
Depends on driver.

S1.LU, Logical Unit
In order to provide device independet I/O, all I/O requests are directed to a logical unit. LU is a number from 0 up to 255. The particular resource desired must be assigned to the specified LU prior to executing the SVC1 call. If an invalid or unassigned LU is specified, the call is rejected, unless reference to the system device numbers is allowed. If no operation is desired, the specified LU should be assigned to the NULL device.

S1.TS, Termination Status
This status byte may contain information unique to the specific type of device.

S1.BAD, Buffer Address
The buffer is specified by the buffer start address, and points to the first byte in the buffer. All buffers must be fully contained in the same logical segment of the task address space. Buffers used in read requests must be in a writable segment, since the memory locations are changed by the read operation.

S1.BSZ, Buffer Size
The buffer size specifies the number of bytes to be written, and the maximum number of bytes to recieve.

S1.BCNT, Byte Count
This field is used to return the actual number of bytes transferred during a request. This field is most useful when dealing variable length record devices, such as magnetic tape. This field is undefined on error status.

S1.RND, Random Address
Used when the function code specifies random (S1F.RND). It is interpreted in two different ways depending on the format specified in function code.

On Image ASCII/Binary, it specifies the logical record number (starting at 0) to be accessed for data transfer.

On Formatted ASCII, it is divided into two 16-bit fields which contains positioning information at data transfer. These fields are input in Read-function and output in Write-function. The first field (most significant part) contains vertical tabulation and the second horisontal. Positioning is absolute, when the most significant bit is set, the remaining 15-bit specifies an absolute position. Positioning is relative, when the most significant bit is off, the remaining 15-bit forms a signed integer for relative positioning.

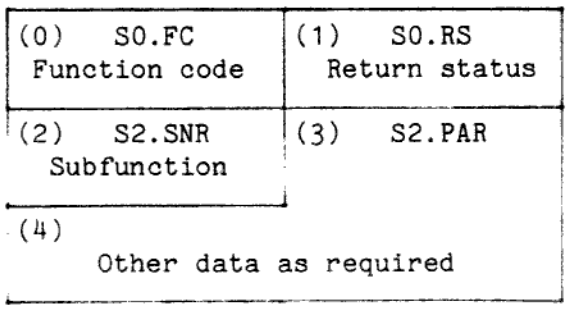
Example: Data shall start on first position of next line, S1.RND contents in hex is 00018000.

5.2 SVC2 - SUBFUNCTIONS

This request type provides a number of general service functions. These functions are related to the task's communication with the console operator, to memory allocation, to text processing and to command processing functions.

The function code is generally used to futher modify the conditions of the call, and is used individually by the subfunctions. For those subfunctions for which no function code are defined, the content is ignored by the subfunction, but check by the operating system as a common function code. That means that no wait for completion and unconditional proceed are supported. All requests require a parameter block.

PARAMETER BLOCK



S0.FC, Function Code
See each subfunction.

S0.RS, Return Status
Contains the return status, see each subfunction. There is one common value:

S2S.ISB	20	Invalid subfunction number.

S2.SNR, Subfunction
Shall contain the requested subfunction, and they are:

EV2.1MEM	1	Memory allocating.
EV2.2MSG	2	Logg message.
EV2.3PFD	3	Pack file descriptor.
EV2.4PNU	4	Pack numeric data.
EV2.5UNP	5	Unpack binary number.
EV2.7DAT	7	Fetch or set date and time.
EV2.8CMD	8	Scan mnemonic table.
EV2.12OC	12	Open/close device.

S2.PAR, Other Data
The content of this field depends on each subfunction.

5.2.1 SVC2.1 - MEMORY HANDLING

This request is used to allocate and deallocate memory. The storage is allocated in system memory.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status
(2) S2.SNR Subfunction 1	(3) S2.PAR Reserved
(4) S2.1ADR Memory address	
(6) S2.1SIZ Memory size	

S0.FC, Function Code

S2F.1ALO	..000001	Allocate memory.
S2F.1MAX	..000010	Reserved.
S2F.1REL	..000011	Release memory.
S2F.1TCB	..000100	Allocate a TCB, only internal use.
S2F.1CAN	..000101	Remove callers TCB, only internal use.

S0.RS, Return Status

S2S.1PAR	21	Illegal parameter.
S2S.1EOM	22	End of memory.

S2.1ADR, Memory Address

Shall contain the memory address at deallocation and will return the memory address at allocation.

S2.1SIZ, Memory Size

Specifies the memory size in bytes to allocate.

5.2.2 SVC2.2 - LOGG MESSAGE

This request is used to logg a message on the terminal device, or system log device, regardless of Logical Unit assignments in force at the time of the request.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status
(2) S2.SNR Subfunction 2	(3) S2.2TS Term. status
(4) S2.2BAD Buffer address	
(6) S2.2BSZ Buffer size	

(8) ~~S2.~~ Reserved

S0.FC, Function Code
Refer to SVC1 function code about data formatting.

S0.RS, Return Status
Always good, S0S.OK !

S2.2TS, Termination Status
Reserved.

S2.2BAD, Buffer Address
Is the address of the buffer to write on the system console.

S2.2BSZ, Buffer Size
Specifies the number of bytes to write.

5.2.3 SVC2.3 - PACK FILE DESCRIPTOR

This request permits the user to process a File Descriptor in standard OS.8 syntax. Leading spaces are ignored, and the scan terminates either when a syntax error (characters that couldn't be a part of the FD) is detected, or it proceeds until it has satisfactorly processed each field. Note that some kind of termination character must exist if string size is unspecified.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S2.SNR Subfunction 3	(3) S2.3TS Term. status
(4) S2.3ADR ASCII-string address		(6) S2.3BUF Address of receiving area	
(8) S2.3PNT Terminating string address		(10) S2.3CNT String size	

S0.FC, Function Code

S2F.3FN	..00...1	Unpack as filename, if not specified.
S2F.3KEP	..00..1.	Keep non-modified fields.
S2F.3CNT	..00.1..	String size specified.
S2F.3PMO	..001...	Pack modifier.

S0.RS, Return Status

S2S.3IFD	21	Invalid file descriptor, syntax error.

S2.3TS, Termination Status

S2T.3NEL	0000...1	Element name not found.
S2T.3NFN	0000..1.	File name not found.
S2T.3NVO	0000.1..	Volume name not found.
S2T.3NMO	00001...	Modifier not found (only set if S2F.3PMO is requested, and no modifier found).

S2.3ADR, String Address

Is a pointer to a string that contains the file descriptor to be packed. The length of this string can be limited by setting the S2F.3CNT-bit in S0.FC-field, and give the size in S2.3CNT-field.

S2.3BUF, Receiving Area

This is a pointer to a 29-byte area. Not that the modifier field is on the negative side of the area, and must only be present if the function S2F.3PMO is requested.

(-1)	FD.MOD File modifier
(0)	FD.VOL Volume name
(4)	FD.FILE File name
(16)	FD.ELMT Element name

Höger el. vänster justerat?

S2.3PNT, Terminating String Address

This field is returned pointing to the first byte that is not part of the file descriptor.

S2.3CNT, String Size

This is an optional field, specifing the string length.

5.2.4 SVC2.4 - PACK NUMERIC DATA

This request translates ASCII hexadecimal, decimal or octal character strings to binary 8/16/24/32-bit numbers. Leading spaces are ignored, and the conversion continues until a character not conforming to the base is found.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S2.SNR Subfunction 4	(3) S2.4SIZE Size
(4) S2.4ADR String address		(6) S2.4PNT Updated string address	
(8) S2.4RES Result			

S0.FC, Function Code

	..00..xx	Conversion base.
S2F.4DEC	..00..00	Decimal.
S2F.4OCT	..00..01	Octal.
S2F.4HEX	..00..10	Hexadecimal.
	..00.x..	Sign handling.
	..00.0..	No signed input allowed.
S2F.4SGN	..00.1..	Input may be signed.
	..00x...	Destination.
	..000...	In parameter block.
S2F.4IND	..001...	Address specified.

S0.RS, Return Status

S2S.4OFL	21	Overflow.
S2S.4NCV	22	Nothing converted.

S2.4SIZE, Size

Describes the size of the binary result, and gives the possibility of an auto-incrementing result pointer.

S2Z.4BIN	.xxx....	Specifies the size of result field.
S2Z.4INC	1.......	Makes the result-pointer auto-incrementable.

S2.4ADR, String Address

This is a pointer to the first character of the ASCII string to be converted.

S2.4RES, Result

The result is placed either in this field, or at the address specified by this field.

S2.4PNT, Updated String Address

This is the updated string pointer at return, and it is pointing at the first byte in the string that was not converted.

5.2.5 SVC2.5 - UNPACK BINARY NUMBER

This request translates an 8/16/24/32-bit binary number into ASCII hexadecimal, decimal or octal format.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S2.SNR Subfunction 5	(3) S2.5SIZE Size
(4) S2.5ADR Destination address		(6) S2.5PNT Updated string address	
(8) S2.5VAL Source			

S0.FC, Function Code

	xx	Converting base.
S2F.5DEC	00	Decimal.
S2F.5OCT	01	Octal.
S2F.5HEX	10	Hexadecimal.
	.....x..	Sign handling.
	.....0..	Unsigned conversion.
S2F.5SGN	.....1..	Signed conversion.
	....x...	Source description.
	....0...	Number in parameter block.
S2F.5IND	....1...	Address specified.
	...x....	Space flag.
	...0....	Leading zeros.
S2F.5SP	...1....	Leading spaces.
	..x.....	Field justifieing.
	..0.....	Right justify.
S2F.5LFT	..1.....	Left justify.

S0.RS, Return Status
Always good, S0S.OK !

S2.5SIZE, Size
Describes the size of both binary and ASCII fields, and gives the possibility of an auto-incrementing binary pointer. If the number to be converted exceeds the buffer length, most significant bytes are lost. If supression of leading zeros is requested, the number is stored in the buffer, and the remaining characters, if any, are filled with spaces. If the number is to be left justified, only the number of bytes required for the number will be used, and S2.5PNT will point at the next position after the number.

```
S2Z.5ASC  ....xxxx  Number of bytes in ASCII-string.
S2Z.5BIN  .xxx....  Number of bytes in binary number.
S2Z.5INC  1.......  Auto-increment of binary pointer.
```

S2.5VAL, Source

The binary number is placed either in this field, or at the address specified by this field.

S2.5ADR, Destination Address

This is a pointer to the first location of a buffer in memory where the converted number is to be stored. This buffer must be in a writable logical segment.

S2.5PNT, Updated Destination Address

This is the updated buffer address pointer at return, and it is pointing at the first byte in the buffer after the converted number.

5.2.7 SVC2.7 - FETCH/SET DATE/TIME

This request is used either to interrogate the time-slice value, or to fetch and set the current date and time of day in the system. The system maintains a calendar and a 24-hour clock.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status
(2) S2.SNR Subfunction 7	(3) S2.PAR Reserved
(4) S2.7BUF Buffer address	

S0.FC, Function Code

S2F.7GET	..0...01	Fetch function.
S2F.7SET	..0...10	Set function.
S2F.7SLC	..0.00..	Slice handling.
S2F.7DAT	..0..1..	Date handling.
S2F.7TIM	..0.1...	Time handling.
	..00....	ASCII data, not at slice handling.
S2F.7BIN	..01....	Binary data, not at date and time handling.

S0.RS, Return Status

S2S.7DAT	21	Invalid date.
S2S.7TIM	22	Invalid time.

S2.7BUF, Buffer Address

This field holds the value at slice handling, or contains the address to a buffer, within user´s program, that receives or sends the values at date and/or time handling.

If ASCII format is selected, the buffer must be ten bytes long for date and eight bytes long for time handling, and nineteen bytes long for both date and time handling. On set date/time the buffer must be terminated by a binary zero. The format is:

Date: YYYY-MM-DD
Time: HH.MM.SS

Date and time: YYYY-MM-DD HH.MM.SS

5.2.8 SVC2.8 - SCAN MNEMONIC TABLE

This request permits the user to decode command mnemonics. A command can consist of more than one word. Leading spaces are always ignored.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status
(2) S2.SNR Subfunction 8	(3) S2.8INX Index
(4) S2.8ADR String address	
(6) S2.8LIST Mnemonic table address	
(8) S2.8PNT Updated string address	

S0.FC, Function Code
None.

S0.RS, Return Status

S2S.8CMD 21 Undefined command mnemonic.

S2.8INX, Index
This field returns the number in the table of the matched mnemonic, starting with zero. Thus, if a match is found on the third item in the table, the index returned is 2.

S2.8ADR, String Address
This field shall contain the address to the source string, within the user´s program space, to be scanned.

S2.8LIST, Mnemonic Table Address
This field shall contain the address of a mnemonic table within the user´s program space. A mnemonic table is composed of a string of mnemonics, separated from each other by a byte containing binary zero. The end of the table is signified by the occurrence of two consecutive bytes of binary zeros.

The first byte of a mnemonic specifies the abbreviation size. The abbreviation must begin with the first character in the mnemonic, and must also be contiguous. Thus, the mnemonic TWO WORDS, in wich letters TW and W are required, is coded as:

```
CMDTABLE  DB      2                       Abbreviation size.
          DB      'ONEWORD'               Command in ASCII.
          DB      0                       End of command.
          DB      2,'TWO',1,'WORDS',0
          DB      -1                      Flag to indicate that table
          DA      NEXTABLE                continues at this address.
*
NEXTABLE  DB      2,'TREE',1,'WORDS',1,'COMMAND',0
          DB      0                       End of table !
```

S2.8PNT, Updated String Address

This field returns the updated string address, and it is pointing to the first character that was not matched. This is normally a separator following the mnemonic in the string being scanned.

Characters not allowed in table words (terminal characters) are:

delete (7FH)
space
"
#
'
(
)
+
,
.
/
:
;
<
=
>
?

5.2.12 SVC2.12 - OPEN/CLOSE DEVICE

This request is used to take a device off-line, or to bring on-line a device that was previously off-line.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S2.SNR Subfunction 12	(3) S2.PAR Reserved
(4) S2.12FD Name pointer, or device number		(6) S2.12ADR Optional SVC-handler address	

S0.FC, Function Code

S2F.12CL	000001	Close.
S2F.12OP	10	Open.
S2F.12PR	.....1..	Write protected.
S2F.12NF	....1...	Non-file structured.
S2F.12AD	...1....	SVC-handler address specified, only directory oriented devices.
S2F.12AL	..1.....	Fetch auto start line. Only valid at open file structured.

S0.RS, Return Status

S2S.12AS	21	Device is assigned, can't be closed.
S2S.12DE	22	Device not found.
S2S.12IS	23	New volume already present.
S2S.12ON	24	Directory device not in close state.

S2.12FD, Name Pointer

This field shall contain either a pointer to a symbolic device name, or numeric value less than 255, that is the numeric identity of the device.

S2.12AD, SVC-handler Address

This optional field may contain the address to a user written filehandler.

S2F.12OP, Function Open

If a directory device is opened file-structured by a symbolic name, the volume name will be returned in the file-name field of the file-descriptor.

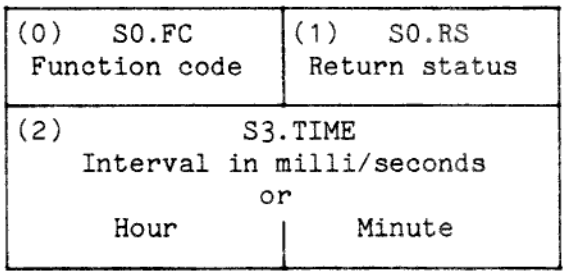
S2F.12AL, Fetch Auto Start Line

Data contained in the auto start line area on the disk is returned to name pointer+8. 80 characters are moved.

5.3 SVC3 - TIMER REQUESTS

This request is used to coordinate with the real time, and supports both interval and time-of-day requests.

PARAMETER BLOCK



S0.FC, Function Code

	..0000xx	Time sepcification:
S3F.MIL	..000001	Milliseconds.
S3F.SEC	..000010	Seconds.
S3F.TOD	..000011	Time of day.
	..100xxx	Commands:
S0F.TST	..100000	Test request.
S0F.CAN	..100001	Reserved.
S3F.CMIL	..100010	- " -
S3F.CSEC	..100011	- " -
S3F.CTOD	..100100	- " -
	.x.....	Wait-proceed bit.

S0.RS, Return Status

S3S.PAR	30	Invalid interval/time-of-day.

S2.3TIME, Interval

This field contains either an unsigned interval value, or an absolute time of day value. This field is also used at cancel function, and should then point at the previus parameter block that sould be cancelled.

5.4 SVC4 - TASK DEVICE

It is sometimes necessary for a symbiont task to re-trigg one of its task-devices, that means add an item to the event queue of the task. It is also possible to indicate that the symbiont handler should cancel the request in progress. This request performs those functions, and is only used by the task who owns the task-device. More information about symbiont and task-devices will be found in chapter 5.6, function Wait For Event.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status
(2) S4.LU Logical unit	(3) Reserved

S0.FC, Function Code

S4F.TRIG	..0000.1	Trigg initiator.
S4F.CAN	..00001.	Set cancel pending.

S0.RS, Return Status

S4S.ASGN	40	Not assigned.
S4S.TYPE	41	Invalid device type.

S4.LU, Logical Unit

This field holds the logical unit that should be accessed.

5.5 SVC5 - LOADER HANDLING

This request is used to load an overlay or a task. A task is normally loaded through SVC6, where to find more detailed information about task loading. An overlay is loaded into the requesting tasks segment at relative address specified in the parameter block. An overlay can also be started at the same time, that means chain the program execution to a new program segment.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) Reserved	(3) Reserved
(4) S5.TID Name pointer or task number		(6) S5.LAD Load address for overlay	
(8) S5.SAD Start address		(10) S5.FD File descriptor	
(12) S5.SIZE Additional size			

S0.FC, Function Code

S5F.LOAD	..00...1	Load.
S5F.STRT	..00..1.	Start overlay.
S5F.ABS	..00.1..	Absolute start address at overlay start.
S5F.OVL	..001...	Overlay handling, else task handling.
S0F.NW	.x.....	Wait-proceed bit.
S0F.PRO	x.....	Unconditional proceed bit.

S0.RS, Return Status

S5S.TID	50	Illegal task name/number.
S5S.CODE	54	Illegal code/item at load.
S5S.SIZE	55	Overlay don't fit.

S5F.LOAD, Load Overlay

The overlay is loaded from the device specified by the file descriptor, into the requesting tasks segment at relative address specified by S5.LAD. The calling program is placed in Load wait until the overlay is loaded. If the overlay is successfully loaded, the root program may call it as a subroutine. Overlays can call other overlay directly; the calling code is overlaid with the new overlay, but the task is aborted if the overlay load failed. Note that the overlay must fit within the root segments size.

S5F.STRT, Start Overlay

This call starts the named overlay. If absolute start is requested, the S5F.ABS-bit is set, the overlay is started at the address specified by the S5.SAD-field. If relative start is requested, the S6F.ABS-bit is cleared, the overlay is started at its establish transfer address plus the value in the S5.SAD-field.

5.6 SVC6 - TASK REQUEST

This request is used to manipulate with other tasks or own task.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S6.PRIO Priority	(3) S6.OPT Option
(4) S6.TID Name pointer or task number		(6) S6.PAR Parameter	
(8) S6.SAD Address		(10) S6.FD File descriptor	
(12) S6.SIZE Additional size			

S0.FC, Function Code

S6F.LOAD	..000..1	Load task.
S6F.STRT	..000.1.	Start task.
S6F.ABS	..0001..	Absolute start address.
S6F.QTST	..001000	Test event queue.
S6F.QWAI	..0010.1	Wait for queue event.
S6F.QTRM	..00101.	Terminate event.
S6F.QDIS	..001100	Disable event queue.
S6F.QENI	..001101	Enable event queue.
S6F.SUSP	..001110	Suspend myself.
S0F.TST	..100000	Test task.
S0F.CAN	..100001	Cancel task.
S6F.PAUS	..100010	Pause task.
S6F.CONT	..100011	Continue task.
S6F.PRIO	..1001.1	New task priority.
S6F.OPT	..10011.	New task option.
S6F.TSKW	..101000	Wait for task termination.
S6F.ADDQ	..101001	Add to event queue.
S6F.STSW	..101010	Wait for task status change.
S6F.TYPE	..101011	New task type.
S0F.NW	.x.....	Wait-proceed bit.
S0F.PRO	x.....	Unconditional proceed bit.

S0.RS, Return Status

S6S.TID	60	Illegal task name/number.
S6S.PRES	61	Task already present.
S6S.PRI	62	Illegal priority.
S6S.OPT	63	Illegal option.
S6S.EQUE	64	Event queue disabled.
S6S.STAT	65	Invalid task status.
S6S.QPAR	66	Invalid termination parameter.
S6S.QITM	67	More items present in event queue.
S6S.TYPE	68	Invalid task type.

S6.PRIO, Priority

This field is used at functions S6F.STRT and S6F.PRIO, where the value zero means that the priority defined at task-establish time should be used.

S6.OPT, Options

The S6.OPT-field is used at function S6F.OPT, and shall contain the options of the task.

S6O.DASG	0000...1	Default assign allowded.
S6O.NSTK	0000..1.	No stack check.
S6O.EMSG	0000.1..	Error message print-out by system.
S6O.RCOV	00001...	System error recovery.

This field is also used at function S6F.TYPE:

S6T.RES	00000..1	Set the task memory resident.
S6T.NAB	00000.1.	Set the task non-abortable from other tasks.

S6.TID, Name Pointer

This field shall contain either a pointer to a symbolic task name, or a numeric value less than 255, that is the numeric identity of a task. A zero value means that the request is self-directed.

S6.PAR, Parameter

This field is used at functions S6F.STRT, S6F.QWAI, S6F.QTRM, S6F.QTST and S6F.ADDQ.

S6F.LOAD, Function Load Task

The required fields are: S6.PRIO, S6.OPT, S6.TID, S6.FD and S6.SIZE.

The specified task is loaded from the device specified by S6.FD. If a task is already present in the system with the given S6.TID or the S6.TID is invalid, the call is rejected. A memory area, expanded with S6.SIZE and large inough, is allocated and the task is given the name found by the S6.TID field. If there is not memory enough, the call is rejected.

S6F.STRT, Function Start Task
The required fields are: S6.PRIO, S6.TID, S6.PAR and S6.SAD.

This call starts the named task. If absolute start is requested, the S6F.ABS-bit is set, the task is started at the address specified by the S6.SAD-field. If relative start is requested, the S6F.ABS-bit is cleared, the task is started at its establish transfer address plus the value in the S6.SAD-field. If the S6.PRIO field is not zero, it is taken as the priority for the task. If the S6.PAR-field is not zero, it is taken as the address to a vector, where the first two bytes of the vector is the number of bytes that should be transferred to the task. All these bytes are transferred to the specified task's stack. On odd numbers of bytes, a last byte of zero is added. The conditions when a task is started are:

- Registers A,B,C,D,E,H,L are transferred from the starting task. If started from the MTM-System, the registers E,D,C,B holds a 32-bit bit pattern corresponding to the switches specified.
- Register Y points at the first byte after the program.
- Register X points at the lowest possible value-1 for the SP.
- On top of stack an 16-bit unsigned integer that holds the number of parameter bytes present in the stack. If no parameters are transferred, the value is zero.

The size of the additional memory added to a task is calculated by subtracting register Y from X.

S6F.TST, Funtion Test Task
The required fileds are: S6.PRIO, S6.OPT, S6.TID and S6.PAR.

This function is useful to test the presence of a task, and where the TCB-address is returned in S6.PAR.

S6F.CAN, Function Cancel Task
The required fields are: S6.PRIO and S6.TID.

This call permits a task to terminate itself, or another task, in an orderly fashion. The Return Code, S6.PRIO, may be treated as desired by another task waiting for its termination. Normally the return code 0 represents normal termination. If the task has I/O in progress at the time the call is made, the I/O must be completed before the task goes to end of job. Then all its files and devices are closed. If the task is a non-resident task, it is removed from memory by deleting all control information pertaining to the task. The task will never return from the call if it is self-directed.

S6F.PAUS, Function Pause Task
The required field is: S6.TID.

This function causes the specified task to enter Pause state. The task does not continue to execute until it is released by

an S6F.CONT call. A task may suspend itself. In that case, another task must be available to release it subsequently.

S6F.CONT, Function Continue Task
The required field is: S6.TID.

This function takes a paused task out of its Pause state. The task continues to execute as before it was paused, provided it is not in any other Wait state.

S6F.PRIO, Function Change Priority
The required fields are: S6.PRIO and S6.TID.

This function changes the priority of the specified task to that specified in the S6.PRIO field of the parameter block. The call is rejected if the priority is outside the valid range of 1-255.

S6F.OPT, Function Change Options
The required fields are: S6.OPT and S6.TID.

This call will change the options on a task according to the bit pattern in S6.OPT. It is possible to combine S6F.PRIO with S6F.OPT.

S6F.TSKW, Function Wait for Task Termination
The required fields are: S6.PRIO, S6.OPT and S6.TID.

This call is used when a task wants to wait for another task and its termination. At return from the call, S6.PRIO holds the new status of task, S6.PRIO contains the return code from the task.

S6F.ADDQ, Function Add to Event Queue
The required fields are: S6.TID and S6.PAR.

The parameter in the S6.PAR field of the parameter block is added to the specified task´s Event Queue, if the queue is enabled. Otherwise, the call is rejected with appropriate error status.

S6F.TSTW, Function Wait for Task Status Change
The required fields are: S6.PRIO, S6.OPT and S6.TID.

This call is used by a task, when the task wants information about any status change on another task. At return from the call, the fields content are the same as on S6F.TSKW. The status changes are:

- Transition to/from Dormant state.
- Transition to/from Pause state.

S6F.TYPE, Function Change Type
The required fields are: S6.OPT and S6.TID.

This call will change the type of a task according to the bit pattern in S6.OPT.

EVENT QUEUE HANDLING

The following functions describes how to handle the event queue. In general, every own queued no-wait request will be added to the event queue, if enabled. The required fields are almost always: S6.PRIO, S6.TID and S6.PAR. The content of the fields, if valid are:

S6.PRIO The termination status on S6F.QTRM, else the SVC-type.
S6.TID Task number, zero if self-directed.
S6.PAR Address of the invoked parameter block.

S6F.QENI, Function Enable Event Queue
The event queue of a task must be opened prior to using the event queue. Any addition to a non-enabled event queue will faile, and those requests will have a return status off SOS.OFFL.

S6F.QWAI, Function Wait for Event
The required fields are: S6.PRIO, S6.TID and S6.PAR.

This request is used when no more actions can be taken by the task. The task will enter Trap wait state. As soon as any item is added to the event queue, or if the event queue isn't empty, the task returns with return status zero, and all fields valid. If the item belongs to an external task, S6.TID is non-zero, the item is saved in a slough queue within the task. The item will remain in the slough queue until it is terminated by the task. If the item is self-directed, S6.TID is zero, it is not saved in the slough queue. The field S6.PAR holds the address of the SVC-parameter block that was the reason for the event. In revision 3.00 and higher, this field contains the node-address at an external event, and the item is not saved in the slough queue.

If the item added to the event queue refers to a task-device, then the symbiont initiator for that task-device will be entered, and the call will not return to the task. The symbiont handler works in the same way as a real device driver with the same conditions. More detailed information about device drivers will be found in OS.8 System Logic Manual.

This function also implicits function S6F.QENI.

S6F.QTST, Function Test Event Queue
The required fields are: S6F.PRIO, S6F.TID and S6.PAR.

This request is used to test if there are anything in the event queue. If empty, the request returns with return status zero, else it returns with return status S6S.QITM and the content of the fields are valid. This call will not remove the event from the event queue.

S6F.QTRM, Function Terminate Event

The required fields are: S6.PRIO and S6.PAR.

This function is used to remove an external request from the slough queue, and to terminate it. S6.PRIO shall contain the final return status for that request, and S6.PAR shall contain the parameter received at S6F.QWAI. It is also possible to combine S6F.QTRM with S6F.QWAI.

S6F.QDIS, Function Disable Event Queue

This function is used to close the event queue, and every present or queued request will be terminated with return status S0F.OFFL.

S6F.SUSP, Function Suspend

The required field is: S6.PRIO

This request is used to relinquish control of the processor and be added to the Ready-Queue. The priority to be used is specified in the field S6.PRIO. If the value is zero, the current priority will be used.

SVC 6 PARAMETER BLOCK FIELD

	S6.PRIO	S6.OPT	S6.TID	S6.PAR	S6.SAD	S6.FD	S6.SIZE
S6F.LOAD	U	U	U			U	U
S6F.STRT	U	U	U	U	U		
S6F.QTST	S=SVCNR		S=TNR	S=PBLK			
S6F.QWAI	S=SVCNR		S=TNR	S=PBLK			
S6F.QTRM	U=SO.RS			U=PBLK			
S6F.SUSP	U						
SOF.TST	S	S	U	S=TCB	S		
SOF.CAN	U=RCOD		U				
S6F.PAUS			U				
S6F.CONT			U				
S6F.PRIO	U		U				
S6F.OPT		U	U				
S6F.TSKW	S=STAT	S=RCOD	U				
S6F.ADDQ			U	U=PBLK			
S6F.STSW	S=STAT	S=RCOD	U				
S6F.TYPE		U	U				

U = should be initiated by user before SVC instruction.
S = returned by system after SVC instruction.
PBLK = address of invoked parameter block.
RCOD = return code at task termination.
SO.RS = final return status for the request.
STAT = new task status.
SVCNR = Type of SVC.
TCB = addresss to Task Control Block.
TNR = task number, zero if self directed.

5.7 SVC7 - FILE REQUEST

This request is used to create files, assign files, devices and tasks to a logical unit, and to modify them.

The meaning and the use of each field is explained in the descripiton of each function requiring that field.

When dealing with non random-access devices the Record Length and Size fields doesn´t need to be specified.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S7.LU Logical unit	(3) S7.MOD Modifier
(4) S7.FD Name pointer, or device number		(6) S7.CLAS Class	(7) S7.TAM Access mode
(8) S7.RECL Record length		(10) S7.UBUF Reserved	
(12) (LSW)	S7.SIZE Size	(MSW)	

S0.FC, Fucntion Code

S7F.ALLO	..0....1	Allocate.
	..0...1.	Reserved.
S7F.ASGN	..0..1..	Assign.
S7F.DELC	..0.1...	Delete at close.
S7F.CLOS	..01....	Close.
S0F.TST	..100000	Test request.
S0F.CAN	..100001	Cancel all previus requests.
	..100010	Reserved.
S7F.CHKP	..100011	Checkpoint (from 3.00).
	..100100	Reserved.
	..100101	- " -
S7F.RNAM	..100110	Rename.
S7F.FAT	..100111	Fetch attributes.
	.x.....	Wait-proceed bit.
	x.....	Unconditional proceed bit.

S0.RS, Return Status

S7S.ASGN	70	-	Assignment error.
S7S.AM	71	-	Illegal access mode.
S7S.SIZE	72	-	Size error, size field invalid or specifies non existing space.
S7S.TYPE	73	-	Type error, LU is not a direct-access device.
S7S.FD	74	-	File descriptor error, file descriptor of invalid format.
S7S.NAME	75	-	Name error, matching name not found.
S7S.KEY	76	-	Invalid key.
S7S.FEX	77	-	File already exist.
	1x	-	I/O error, SVC1 code given.

S7.FD, Name Pointer

This address byte points at the file name. (See file descriptor formats.) Values lower than 256 means that the reference is made to a system device with the same number.

S7.MOD, Modifier

The modifier specifies the type of a file at allocation and assignment time. The data formats are described in the section titled FILEFORMAT. The file type is divided into two 4-bit groups (nibbles):

The most significant nibble specifies the type of data in the file:

FCM.ASC	1x	ASCII data readable without any special handling.
FCM.LIST	2x	List file, ASCII data together with positioning information.
FCM.OBJ	3x	Object code, readable by the ESTAB.
FCM.BIN	4x	Binary data, which is unspecified.
FCM.TSK	5x	Task file, either relocatable or absolute.
FCM.ISM	6x	ISAM index file.
FCM.DIR	Fx	Directories.

The least significant nibble, if specified, defines a set of languages and directory types:

FCM.ASM	x1	ASSEMBLER source code.
FCM.BAS	x2	BASIC source code, or data produced by BASIC.
FCM.COB	x3	COBOL source code, or data produced by COBOL.
FCM.FTN	x4	FORTRAN source code, or data produced by FORTRAN.
FCM.PAS	x5	PASCAL source code, or data produced by PASCAL.

Some special modifiers:

FCM.UNDF	00	Undefined data, verifies to any other type.
FCM.EFD	FD	Element File Directory.
FCM.MFD	FF	Master File Directory.

S7.CLAS, Class

This field specifies the resource class to be accessed, and they are:

S7C.ALL	000	Scan all resources.

S7.TAM, Access Mode

The low nibble of this byte specifies the Access Priviliges. If the access mode for a direct access file is 'SW', it will be changed to 'EW'! Table 7-1 shows compatibilities between access priviliges.

S7A.SRO	000	Sharable Read Only.
S7A.ERO	001	Exclusive Read Only.
S7A.SWO	010	Sharable Write Only.
S7A.EWO	011	Exclusive Write Only, will position to end-of-file.
S7A.SRW	100	Sharable Read Write.
S7A.SREW	101	Sharable Read, Exclusive Write.
S7A.ERSW	110	Exclusive Read, Sharable write.
S7A.ERW	111	Exclusive Read-Write.

The high nibble is the command modifier:

S7A.SBUF	00.....	System buffering required, device independent I/O.
S7A.UBUF	01.....	Reserved.
S7A.PHYA	10.....	Physical access, access on disk sector level.
S7A.BYTE	11.....	Byte addressing I/O, file treated as a file of bytes.

For detailed information about file I/O, refer to chapter 6.

TABLE 7-1. ACCESS PRIVILIGE COMPATIBILITY

	ERSW	ERO	SRO	SRW	SWO	EWO	SREW	ERW
ERSW	No	No	No	No	Yes	No	No	No
ERO	No	No	No	No	Yes	Yes	No	No
SRO	No	No	Yes	Yes	Yes	Yes	Yes	No
SRW	No	No	Yes	Yes	Yes	No	No	No
SWO	Yes	Yes	Yes	Yes	Yes	No	No	No
EWO	No	Yes	Yes	No	No	No	No	No
SREW	No	No	Yes	No	No	No	No	No
ERW	No	No	No	No	No	No	No	No

S7.RECL, Record Length

This address field on Allocate File, contains the logical record length for an Index File. If the field is zero VARIABLE Record Length is asumed.

S7.SIZE, Size

This field is defined for the Allocate Call depending on the type of file beeing allocated. When the high order bit in the field is set, the remaining bits express the size of a single data block, continuous file. For an Indexed file the Size field is divided into two 16-bit fields; the first field (LSW) contains the segment size in sectors, the second field (MSW) contains the number of segments to be preallocated. If the size field is set to zero in the index case, default sizes are taken from the volume information. (See disk initialization). Size is not used for Non-Direct-Access devices.

S7F.CHK, Function Checkpoint (from 3.00)

The checkpoint function flushes the system Buffer Management buffers and updates the File Information Block. LU is the only required parameter.

The user may wish to employ Checkpointing after sensitive data is added to a buffered file. Logical blocking of data in memory in system buffers leaves the file vulneable. The integrity of the data can be preserved on the direct-access device by Checkpointing. In case of system failure, all data on Indexed files up to the latest Close or Checkpoint operation is recoverable; data appended after the most recent Checkpoint is lost. Checkpoint differs from a Close/ Assign sequence in that no repositioning is performed. File name, access privileges, and keys need not be specified.

Applicable error codes:

S7S.ASGN	70	LU Error.
	1x	I/O Error, as returned by SVC1.

S7F.RNAM, Function Rename

This function changes the name of an assigned file. The file must currently be assigned ERW. The required parameters are LU and File-decriptor. The LU must be assigned to a direct-access file (unless the caller is an Executive Task which may rename non direct-access devices). The Volume field of the file descriptor is ignored. The specified File descriptor replaces the previous File descriptor in the directory if rename function is successful. If the modifier field is 0, then the modifier will not be changed, else it will be assigned the value in the parameter block.

Applicable error codes are:

S7S.ASGN	70	LU Error.
S7S.AM	71	Access Mode Error.
S7S.TYPE	73	Type Error, LU is non direct-access.
S7S.FD	74	File Descriptor Error, file descriptor of invalid format.
S7S.NAME	75	Name Error, new name allready exists.
	1x	I/O Error, as returned by SVC1.

S7F.DELC, Function Delete At Close

This function allows user to delete a file on a direct-access device. When the call is done the file is not deleted, only a flag in the FCB is set. The actual delete is performed when the file is closed. This feature allows a very simple Temporary File handling. The only parameter required is LU.

Applicable error codes are:

S7S.ASGN	70	LU Error.
S7S.AM	71	Protect Error, file not ERW assigned.
S7S.TYPE	73	Type Error, LU is non direct access device.
	1x	I/O Error, as returned by SVC1.

S7F.CLOS, Function Close

This function discontinues an assigned logical connection between a task and a file or device. LU is the only required parameter. The specified LU is de-assigned. Logical Units assigned for Write access to files or buffered terminals have any partially filled buffers written to the file by CLOSE call. Direct access devices with the Delete At Close flag set will be deleted. (see delete.)

Applicable error codes are:

S7S.ASGN	70	LU Error.
	1x	I/O Error, as returned by SVC1.

S7F.FAT, Function Fetch Attributes

Certain programs may require, for proper operation, knowledge of the physical attributes of the device or file associated with a given LU. The only required parameter is the Logical Unit. The system returns information in fields Modifier, Name pointer, Record length, User supplied buffer address and Size.

The Modifier byte is set to indicate the file or device type.

The Name pointer must be an address to a buffer where the system returns the name or the mnemonic of the assigned resource. If the pointer value is less than 256, the device number is returned in this field, rather than the name.

The Record length field is set to the physical record length associated with the resource.

The reserved field S7.UBUF is redifined for this call, and receives the attributes.

The current size of a direct-access file is returned in the Size field.

Applicable error codes are:

S7S.ASGN 70 LU error, illegal LU or LU not assigned.

FILEFORMAT

Some of the data formatting within a file are standardized, and are described here.

ASCII files, compressed records:
- Spaces are compressed to 80H + the number of spaces.
- Records are separated by a NULL-byte.
- Records are stored after each other without any padding, to minimize the storage required.

ASCII files, print records (from 3.00):
- Each record starts with positioning information, as specified by SVC1 for formatted ASCII at random access.
- The rest is the same as specified by 'compressed records'.

Binary files, fixed record length:
- Records are stored continuously after each other.
- The data bytes are not specified.
- Both random and sequential access can be done.

Load modules:
- The data formatting is defined by the code-type.

5.8 SVC8 - RESOURCE HANDLING

This request type is used to establish and remove resources in the Operating System on line. This call is normally used by system programmers, because it requires a very good knowledge of the structure and functions of the operating system. If this request is used in an improper way, the Operating System may be damaged.

PARAMETER BLOCK

(0) S0.FC Function code	(1) S0.RS Return status	(2) S8.RNR Resource number	(3) S8.PRIO Resource prio
(4) S8.ID Name pointer or resource number		(6) S8.CLAS Clas	(7) S8.TYPE Type
(8) S8.ADR Entry or 'RDT'-list address		(10) S8.CS S8.SIZE	(11) S8.IL

S0.FC, Function Code

S8F.EST	..000.01	Establish resource.
S8F.RMOV	..000010	Remove resource, only available to the owner.
S8F.TEST	..000011	Test the presens of a resource.
S8F.NRCB	..0001..	RCB already present.

S0.RS, Return Status

S8S.ID	80	-	Illegal name/number.
S8S.CLAS	81	-	Illegal class.
S8S.PRES	82	-	Already present.
S8S.PRNT	83	-	Parent not present.
S8S.DUAL	84	-	Dual not present.
S8S.RCB	85	-	Invalid 'RCB'-type.
S8S.EOM	86	-	End of memory.

S8.RNR, Resource Number

This field is used to specify the numeric identity of a resource. If the number zero is chosen, the system will select the first free number. The numeric identity will be returned in this field.

S8.PRIO, Priority or Number

This field shall contain either the task/device priority, or the parent/dual resource number.

S8.ID, Name Pointer

This field shall contain either a pointer to a symbolic name, or a numeric value less than 255, that is the identity of the resource.

S8.CLAS, Class

This field shall contain the resource class, where the classes are:

S8C.DEV	001	Devices.
S8C.TSK	010	Tasks.
S8C.COM	011	Common.
S8C.VOL	100	Volumes.
S8C.SVC	101	SVC-functions.
S8C.SVC2	110	SVC2-subfunctions.

S8.TYPE, Type

This field shall contain the resource type:

RTT.PURE	00	Shared resource.
RTT.RCB	01	Exclusive resource.
RTT.RRT	10	Dummy, S8.ADR points at a new resource.
RTT.AREA	11	Area, no entry.
RTT.DIR	.... .1..	Directory oriented.
RTT.SVC	.... 1...	Entry at all SVC-calls.
RTT.OFFL	...1	Off line, not accessable.
RTT.PROT	..1.	Protected, write not allowed.
RTT.NFST	.1..	Non-file structured, only for system´s use.
RTT.NRMV	1...	Resident, non-removable.

S8.ADR, Entry/RDT

This is either the entry address of a shared resource, or the address to a Resource Descriptor Table (RDT) for an exclusive resource.

S8.SIZE, Size

Shall contain the additional memory size that should be allocated at establish of a task.

S8.CS, Channel Select Code

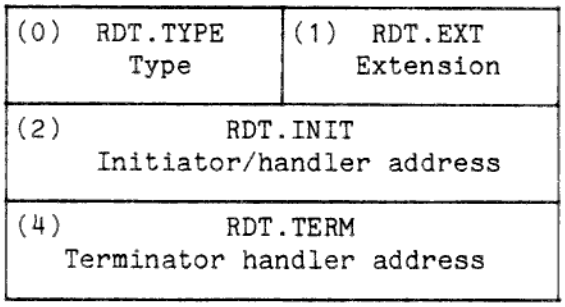
Contains the card identity for the interface.

S8.IL, Interrupt Level

Contains the interrupt level for a physical device.

RESOURCE DESCRIPTOR TABLE (RDT)

A RDT describes shortly an exclusive resource, such as a Device Control Block (DCB), or a Task Control Block (TCB). This table is used by the SVC8-handler to create the control blocks necessary to handle the resource. Each exclusive resource has at least a RDT.



RDT.TYPE, Specifies the type of resource:

RCT.RCB	.... .000	No special type.
RCT.DCB	.... .001	DCB, Device Descriptor Table present.
RCT.TCB	.... .010	TCB, Task Descriptor Table present.
RCT.FCB	.... .011	Only for system´s use.
RCT.VCB	.... .100	VCB, Volume Descriptor Table present.
RCT.AREA	.... .101	Area, no entry.
RCT.PRNT	.... 1...	Coordination parent specified.
RCT.DESC	...1	Only for system´s use.
RCT.NW	..1.	Don´t support no-wait functions.
RCT.PRO	.1..	Don´t support un-conditional proceed.
RCT.NAB	1...	Non-abortable, cant be cancelled.

RDT.EXT, Makes it possible to expand any control block, i.e. expand a DCB, where the expansion is used by the device driver. Refer to the section titled Extended Descriptor Table.

RDT.INIT points at the handler entry for the resource, i.e. the initiator for a device driver.

RDT.TERM points at an optional terminator, i.e. to do code conversions or CRC calculation in a device driver.

TASK DESCRIPTOR TABLE (TDT)

This table is a continuation of the RDT. It is used by the system to create a Task Control Block (TCB), which is used by the system to control a task. The information in the TDT is a short description of the characteristics of a task:

(6) TDT.TYPE Task type	(7) TDT.OPT Task options
(8) TDT.SADR Standard start address	
(10) TDT.TLIM Individual slice limit	
(12) TDT.NNOD Number of nodes	(13) TDT.NFCB Number of FCB
(14) TDT.STK Required stack size	

TDT:TYPE, Describes the type of the task:

TCT.RES	1	Set the task memory resident.
TCT.NAB	.... ..1.	Set the task non-abortable for other tasks.

TDT.OPT, Holds the options on the task:

TCO.DASG	1	Default assign allowed.
TCO.NSTK	.... ..1.	No stack check.
TCO.EMSG	.... .1..	Error message print-out by the system.
TCO.RCOV	.... 1...	System recovery.

TDT.SADR, Is the normal start address of the task.

TDT.TLIM, Gives the task an own time-limt that should be used if the time-slice function is enabled. Zero means that the global time-slice limit should be used.

TDT.NNOD, Specifies the number of nodes that should be allocated to the task. In general, the number of nodes required by a task is calculated from:

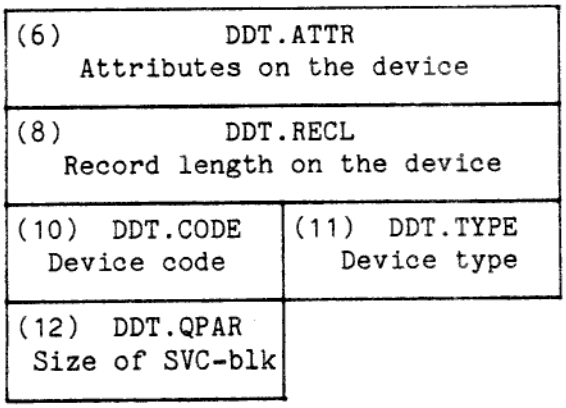
- The number of outstanding queued no-wait requests.
- The number of task devices owned by the task.
- The number of assignments to physical devices, not files.

TDT.NFCB, Specifies the number of File Control Blocks (FCB) that should be allocated to the task. One FCB is used for every assign to a file, and one extra at assign time for an element within a file.

TDT.STK, specifies the required stack for the task.

DEVICE DESCRIPTOR TABLE (DDT)

Both real and task devices are specified by this table. The DDT is a continuation of RDT.



DDT.ATTR, Is a bit pattern that describes the support on the device:

ATR.READ	1	Read.
ATR.WRIT	....1.	Write.
ATR.FASC	.... .1..	Formatted ASCII.
ATR.SPEC	.... 1...	Special formatting.
ATR.RND	....1	Random access.
ATR.IACT	....1.	Interactive device, echo of input.
	.... .1..	Reserved for future use.
	.... 1...	Reserved for future use.
	....1	Reserved for future use.
ATR.FR	.... ..1.	Forward record.
ATR.FF	.... .1..	Forward file.
ATR.WF	.... 1...	Write file-mark.
ATR.BR	...1	Back space record.
ATR.BF	..1.	Back space file.
ATR.RW	.1..	Rewind.
ATR.ATTN	1...	Attention.

DDT.RECL, Specifies the record length. Zero means variable length.

DDT.CODE, Identifies the device among several of almost the same type.

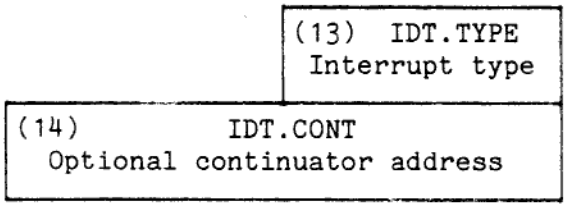
DDT.TYPE, Describes the type of DCB:

DCT.ICB	1	Interrupt Control Block (ICB) present.
DCT.DEDI	.... ..1.	Dedicated interrupt service.
DCT.ENI	.... .1..	Reserved.
	.... 1...	Reserved.
DCT.TASK	...1	Indicates a task device.
DCT.DUAL	..1.	Dual DCB information.

DDT.QPAR, Specifies the number of bytes that sould be copied from the parameter block to the DCB.

INTERRUPT DESCRIPTOR TABLE (IDT)

This table, which is a continuation of the DDT, holds a short description of the interrupt side of a device. This table is only necessary if specified by DCT.ICB, and is used to create an Interrupt Control Block (ICB).



IDT.TYPE, Indicates the type of interrupt:

ICT.CCB	1	Channel Control Block (CCB) present.
ICT.NOIQ	.... ..1.	Makes the ICB resident on the interrupt and time-out chain.

IDT.CONT, Is used by the interrupt system as an address to the device driver continuator. This address is normally initiated or changed by the driver.

CHANNEL DESCRIPTOR TABLE (CDT)

This table, which is a continuation of the IDT, is used to create the Channel Control Block (CCB) for a device. This table is required if specified by ICT.CCB:

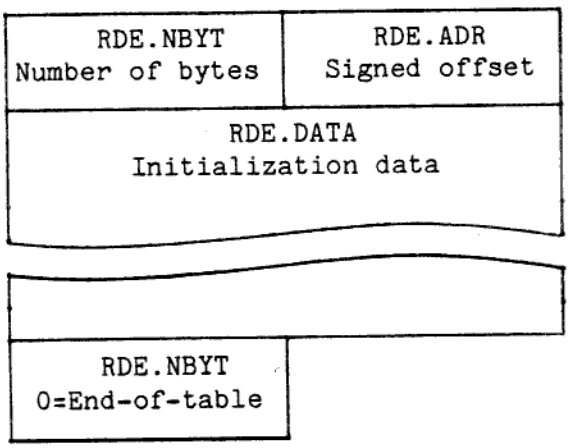
(16)	CDT.TLIM
Time-out limit in chosen interval	
(18)	CDT.THND
Optional time-out handler address	

CDT.TLIM, Is the time-out limit of the device. The value is defined by the device time-out scan frequency, which is a system generation constant, normally resolution is 100 ms.

CDT.THND, Holds the time-out handler address for the device. If not specified, a value of zero, the time-out situation is handled by the system.

EXTENDED DESCRIPTOR TABLE (EDT)

If an extension is specified in the RDT, the EDT must be added after the last desriptor table. This allows the programmer to both expand the control block, and initiate it with some data. If no initialization is required, the shortest EDT possible (one binary zero) must be added.



RDE.NBYT, Specifies the number of bytes that should be copied into the control block. A binary zero is used to terminate the EDT.

RDE.ADR, Is a relative signed offset in the control block where to start to copy the data.

RDE.DATA, Contains the data to be copied to the control block. Must contain EXACTLY RDE.NBYT bytes !

CHAPTER 6

INPUT/OUTPUT PROGRAMMING

6.1 INTRODUCTION

All input/output requests are made via supervisor call SVC 1. Chapter 5 describes SVC 1. This chapter discusses the functional aspects of the devices and direct-access files supported by OS.8. Specific device-dependent information: supported functions, status returned, and formatting performed are included.

All devices and files support binary transfer, proceed I/O, wait for completion, sequential access and conditional proceed, unless otherwise noted in the individual driver or file handler description. The supported attributes listed with each description are in additional to those previously listed.

A device code, a number between 0 and 255, defines all supported devices.

The following paragraphs describes both contiguous and indexed files. Files are available on all direct-access devices supported by any driver.

6.2 FILES

Indexed and Contiguous Files are treaded exactly in the same way when performing input/output. The differances are that the contiguous file cannot be expanded while the indexed can; the EOF pointer always points at end of physical file in contiguous files while it points at end of logical file in indexed files. A file can be accessed in three different ways: Physical (256 byte sectors), Logical (fixed or variable record length), and Byte (as a stream of bytes).

6.2.1 Physical Access

Input/output requests are done without any buffering or data formatting. Fixed length 256 byte sectors are supported.

S1F.READ/S1F.WRIT, Read/Write
: Current file pointer is aligned to a sector boundary before data is transferred. If the device driver can handle partial sectors the buffer size can be between 1 to 65535, else it must be a multiple of 256.

S1F.IBIN, Image Binary
: The only supported data format is image binary.

S1F.RND, Random Access
: Random address is a sector (256 bytes) address.

S1F.FR, Forward Space Record
: Forward 256 bytes (one sector) if not EOF.

S1F.WF, Write File Mark
: EOF = Current random position. Space allocated after EOF will be returned to free space at close (indexed only).

S1F.BR, Backspace Record
: Backspace 256 bytes (one sector) if not BOF.

S1F.RW, Rewind
: Position to beginning of file.

S1F.FEOF, Fetch End-Of-File Position
: End of file pointer is returned in the random address field. On variable record length, it is the number of bytes in the file, and on fixed record length, it is the number of logical records in the file.

NOTE:
: At close, EOF is aligned to the nearest rounded down logical record number.

6.2.2 Logical Access

This mode is used to accomplish device independent input/output. The system data buffering mode is used.

S1F.WRIT, Write - Variable Record Length
Data is written on the disk from the current file pointer.

S1F.READ, Read - Variable Record Length
Data is read from the current file pointer. If the data size is larger than the buffer size, the rest of the record is truncated.

S1F.IBIN, Image Binary
Data is transferred byte by byte without any translation.

S1F.xASC, ASCII Transfer
Spaces are (de)compressed and at write, data on the disk is terminated by a binary zero. If data contains (CR) a new record will be started when found.

S1F.RND, Random Access - Variable Record Length
The random address is a byte address within the file and the user is responsible for correct data formatting when records are written, so the record following the one written is not destroyed.

S1F.WRIT, Write - Fixed Record Length
Data is written from the current file pointer (unless random access is requested). If the requested size written is less than the record size, the rest of the record is padded with binary zeros or spaces depending on whether Binary or ASCII was requested. If the requested size is greater than the record size data is truncated.

S1F.READ, Read - Fixed Record Length
Data is read from the current file pointer. If the record size is greater than the buffer size, the record is truncated.

S1F.RND, Random Access - Fixed Record Length
On fixed record length, record number within the file.

S1F.FR, Forward Space Record
If fixed record length, one record forward. If variable record length, forward 256 bytes.

S1F.WF, Write File Mark
EOF = Current random position. Space allocated after EOF will be returned to free space at close (indexed only).

S1F.BR, Backspace Record
If fixed record length, one record backwords. If variable record length, 256 bytes backwords.

S1F.RW, Rewind
Position to beginning of file.

S1F.FEOF, Fetch End-Of-File Position

End of file pointer is returned in the random address field. On variable record length, it is the number of bytes in the file, and on fixed record length, it is the number of logical records in the file.

6.2.3 Byte Access

This mode is used to accomplish byte input/output on all files regardless of the record length. Binary data transfer is assumed, but with much care ASCII can be used and will then act as variable record length transfer. Any byte in the file can be accessed. Input/output is performed in the same way as logical access towards records with variable record length.

6.3 TERMINAL DRIVER

6.4 SP1/UART OUTPUT DRIVER

6.5 SP1 INPUT DRIVER

6.6 CASSETTE TAPE DRIVER

This driver uses one SP1-input interface and one SP1-output interface.

6.6.1 Supported Devices

FACIT CASSETTE TAPE DRIVE 4203, NON-BUFFERED

6.6.2 Supported Attributes

Read, Write, Image Binary, Unconditional Proceed, Rewind, Backspace Record, Forward Space Record, Write File Mark, Forward Space File Mark, Backspace File Mark, and Variable Record Length.

6.6.3 Functional Description, S0.FC

S1F.READ, Read
: Data is read into the user buffer from the magnetic tape. The transfer ends on buffer full or end of record, whichever comes first. If the record is longer that the user buffer, no error status is returned but a termination status. On a parity error, ten retries are attempted before error status is returned. After a parity error the tape is positioned in the inter-record gap following the record with the error.

S1F.WRIT, Write
: Data is written from the user buffer to the magnetic tape until the buffer is empty. On parity error, an extended record gap is written up to two times, and the write is retried.

S1F.IASC, Image ASCII
: On Read Image ASCII, the most significant bit in each byte is cleared.

Eject, 51Q
: This function will eject the tape cassette.

6.6.4 Return Status, S0.RS

S1S.DWN, Device Down
: Device not ready. Tape unavailable at the start of the request.

S1S.EOF, End Of File
: Filemark detected during request.

S1S.EOM, End Of Media
: Request caused the reflective marker at the end of tape to be sensed.

S1S.TOUT, Time-Out
: The tape drive failed to generate an interrupt within the stipulated time.

S1S.RER, Recoverable Error
Beginning of tape, or write protected.

S1S.UNR, Unrecoverable Error
Hard error.

6.7 MAGNETIC TAPE DRIVER

6.7.1 Supported Devices

PERTEC
AMPEX

6.7.2 Supported Attributes

Read, Write, Image Binary, Unconditional Proceed, Rewind, Backspace Record, Forward Space Record, Write File Mark, Forward Space File Mark, Backspace File Mark, and Variable Record Length.

6.7.3 Functional Description, S0.FC

S1F.READ, Read
Data is read into the user buffer from the magnetic tape. The transfer ends on buffer full or end of record, whichever comes first. If the record is longer that the user buffer, no error status is returned but a termination status. On a parity error, ten retries are attempted before error status is returned. After a parity error the tape is positioned in the inter-record gap following the record with the error.

S1F.WRIT, Write
Data is written from the user buffer to the magnetic tape until the buffer is empty. On parity error, an extended record gap is written up to two times, and the write is retried.

S1F.IASC, Image ASCII
On Read Image ASCII, the most significant bit in each byte is cleared.

6.7.4 Return Status, S0.RS

S1S.DWN, Device Down
Device not ready. Tape unavailable at the start of the request.

S1S.EOF, End Of File
Filemark detected during request.

S1S.EOM, End Of Media
Request caused the reflective marker at the end of tape to be sensed.

S1S.TOUT, Time-Out
The tape drive failed to generate an interrupt within the stipulated time.

S1S.RER, Recoverable Error
Beginning of tape, or write protected.

S1S.UNR, Unrecoverable Error
Hard error. The termination status contains the sense status from the interface.

6.7.5 Termination Status, S1.TS

.......1	The record length exceeded the user buffer length, or file mark detected.
......1.	Hard error.
.....1..	Corrected error.
....1...	End-of-tape.
...1....	Beginning-of-tape.
..1.....	Write protected.
.1......	Rewinding.
1.......	Drive off line.

6.8 DISC DRIVERS

CHAPTER 7

GUIDE TO USING SVC 2 FACILITIES

7.1 INTRODUCTION

Command processors are a part of almost all utility programs and of many application programs. Every time a program is written to accept commands, either by conversation with an operator or by job-control statements, a command processor must interpret those commands. Most command processors are so similar that one could be put together from a "canned" routine package.

This chapter gives the user a guide to the use of the SVC 2 calls provided in OS.8 for command processing functions.

7.2 COMMAND DECODING

The Command Statement is read via SVC 1 from any logical unit or it is passed from one task to another. The issue at hand is how the Command Statement is handled when in the buffer.

A command Statement usally starts with a mnemonic. The first mnemonic is generally called the verb; it tells what to do.

OS.8 provides a call, Scan Mnemonic Table (SVC2.8), which looks for a match on any mnemonic table. These mnemonics must be all alphanumeric characters and cannot contain a delimiter. Thus, $FRED is a valid mnemonic; but, GLA%H is not. The mnemonic table is organized as a byte string. One Null character (binary zero) separates each mnemonic in the table from the following mnemonic. Two successive Null characters terminate the table.

The mnemonic syntax permits abbreviations represented in the mnemonic table by a preceding byte specifying the abbreviation. To represent the mnemonic APPEND, where AP is the minimum acceptable abbreviation, the code within the mnemonic table is:

```
DB    2           required abbreviation
DB    'APPEND'    the complete command
DB    0           end of mnemonic
```

In order to match a mnemonic table entry, the command string must match all required characters. If more alphanumerics are present in the command string, they must match the nonrequired characters. If all required and nonrequired characters are exhausted and more alphanumerics in the command string exist, a match is not recognized. The scan terminates on the first nonalphanumeric.

For the previously given example, a legitimate match is found on:

```
AP
APP
APPE
APPEN
APPEND
```

```
A match is not found on:
  A         (too short)
  APPENG    (no match on nonrequired characters)
  APPENDIX  (too long)
```

The mnemonic table scan routine terminates when it finds a non-alphanumeric character in the command string. If a valid match was found at this point, an index number indicating the position within the mnemonic table of the matched mnemonic is returned. If a match was not found, a bad status is returned. The system does not check the separator (the nonalphanumeric character that terminated the scan routine). The user must check this character. A buffer pointer is returned by the system pointing to the separator where the scan terminated. Entries in the mnemonic table are counted from ZERO; that is, if a match was found on the third mnemonic in the table, the index is returned with a value of 2.

An example of how to use the Mnemonic Table Scan routine follows. Assume that the mnemonic table consists of four entries: SORT, MERGE, PRINT, and STOP.

```
MNMTAB    DB     2,'SORT',0
          DB     2,'MERGE',0
          DB     2,'PRINT',0
          DB     2,'STOP',0,0
```

Notice the two bytes of ZERO ending the mnemonic table. If the command buffer is named CMDBUF, a SVC2.8 parameter block to handle this table is coded:

```
SCAN      DB     0,0,8,0
          DA     CMDBUF,MNMTAB,0
```

Assume that the command line has just been read into a buffer named CMDBUF. The verb is found:

```
FNDVB     SVC    2,SCAN            scan the table
          JNZ    CMDERR            branch if no match
          LD     HL,SCAN+S2.8INX   pick up the index returned
          LI     H,0
          ADR    HL,HL             compute the jump table offset
          LA     DE,JTAB
          ADR    HL,DE             step to proper entry
          L      E,(HL)
          INCD   HL                get handler address
          L      D,(HL)
          EXDR
          JDR    HL                go to command executor
*
JTAB      DA     SORT,MERGE,PRINT,STOP
```

This routine did not check the separator; if the characters SORT* were in the command buffer, the result would be the same as if SORT (Null) was the command buffer content. Assume that only a Null is

allowed to follow the verb. The following code checks if the returned pointer points to the separator after a valid match:

```
LD    HL,SCAN+S2.8PNT  get the terminator pointer
L     A,(HL)           pick up the terminator
OR    A                check for Null
JNZ   CMDERR           branch if not found
```

If different verbs require separators, the separator check can be done after the jump table branch.

7.3 OPERAND DECODING

Most command languages require operands for thier verbs. The verb tells the processor what to do; the operand tell what to do it to, or how to do it. Operands in OS.8 command processor syntax are:

Decimal numbers up to 4,294,967,295
Hexadecimal numbers up to FFFFFFFF
Octal numbers up to 37777777777
File Descriptors
Mnemonics
Arbitrary ASCII strings

These operands are sufficient for most needs. If other forms of operands are required, the user must decode them by own programming.

7.4 NUMERIC CONVERSIONS

The Pack Numeric Data (SVC2.4) call allows decimal, hexadecimal and octal numbers to be decoded. Leading spaces are always ignored. The first nontranslatable character found terminates the operation. The termination pointer is returned by the SVC-function and set to point to that character so the separator can be easily checked.

Assume a rewind command's syntax was:

REWIND lu

where: lu is a decimal number less that 256.

Assume that the verb finder found a match on REWIND and wants to process lu:

```
REWIND  LD    HL,SCAN+S2.8PNT      get termination pointer
        L     A,(HL)               pick up separator
        CI    ' '                  check if valid
        JNZ   CMDERR               branch if invalid separator
        STD   HL,PACKDEC+S2.4ADR   set up ASCII buffer
        SVC   2,PACKDEC            get decimal number
        JNZ   CMDERR               error if none or too many digits
        LD    HL,PACKDEC+S2.4PNT   get termination pointer
        L     A,(HL)               pick up separator
        OR    A                    check for Null
        JNZ   CMDERR               branch if not
        SVC   1,RWDPBLK            rewind the lu
```

```
PACKDEC   DB    S2F.4DEC+S2F.4IND,0,4,10H
          DA    0,0,RWDPBLK+S1.LU
          -
RWDPBLK   DB    S1F.RW,0,0,0
          DA    0,0,0
```

7.5 FILE DESCRIPTORS

This section applies to programs which perform file or device handling.

When the Pack File Descriptor (SVC2.3) is called, it expedites decoding the entire syntax, including volume name, file name, element name, type modifier, and separators; it handles the optional cases and flags them.

To illustrate the use of this call, the syntax of the rewind command may be changed to:

REWIND fd

A valid processing routine is:

```
REWIND    LD    HL,SCAN+S2.8PNT     get termination pointer
          STD   HL,PACKFD+S2.3ADR   set up ASCII buffer
          SVC   2,PACKFD            pack fd
          JNZ   CMDERR              branch on syntax error
          SVC   7,ASGN0             open the device or file on LU 0
          SVC   1,RWDPBLK           rewind the device
          SVC   7,CLOSE0            close LU 0
          -
PACKFD    DB    S2F.3FN,0,3,0
          DA    0,NAME,0
          -
ASGN0     DB    S7F.ASGN,0,0,0
          DA    NAME
          DB    S7C.ALL,S7A.ERW
          DA    0,0,0,0
          -
NAME      DMB   28,' '              File Descriptor field
```

In this example, SVC 1 or SVC 7 calls were not checked for errors, because the illustrations do not concern the use of SVC 1 or 7.

The Termination Status field S2.3TS contains a bit pattern that describes which fields in the File Descriptor that has been processed. If the program requires certain fields, this staus can be checked:

```
          L     A,PACKFD+S2.3TS   get the termination status
          NI    S2T.NFN           verify volume name packed
          JNZ   CMDERR            branch if missing
```

OS.8

SYSTEM LOGIC MANUAL (SLM)

CONTENT

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This System Logic Manual (SLM) is a guide to the internal structure of the Operating System OS.8. It is intended for use by people involved in System maintenance and completion.

This manual deals exclusively, and specifically, with OS.8 revision R2.50. Hence, specific methods of implementation of various functions should not be constructed as the method of implementation to be used in all future releases os OS.8.

Use of this manual requires that the reader be familiar with the features, functions and conventions of OS.8 from the user´s point of view as documented in:

- DataBoard-4680 System Manual.
- DataBoard-4680 Software Catalog.
- DataBoard-4680 Assembler Manual.
- DataBoard-4680 OS.8 Operator´s Manual.
- DataBoard-4680 OS.8 Program Reference Manual.

OS.8 is an operating system that provides program management in a multi-tasking environment. System control via terminal operator, interrupt handling, I/O servicing and inter-task communication/control are build-in functions of OS.8. Disc file management features are also provided when the system is equipped with a disc, and as such, OS.8 is oriented towards a disc operating environment. A file directory and allocation bit map are maintained on each disc volume to allow for disc portability.

Chapter-2 of this manual describes the general structure of OS.8.

Chapter-3 discusses the conventions followed by the system in terms of interfacing between modules.

Chapter-4 is designed to provide a short technical overview of the system and the major module groupings.

Chapter-5 contains a detailed technical description of Device Drivers.

Chapter-6 explains the use of the Data Formatter.

Chapter-7 describes Symbiont handlers and Task Devices.

Chapter-8 discusses briefly the aspects on system generation.

Chapter-9 contains the format of system control blocks.

CHAPTER 2

SYSTEM STRUCTURE

2 SYSTEM STRUCTURE

This chapter describes the system level structure of OS.8 from a technical viewpoint.

2.1 PROCESSOR STATES

At any given time, the Processor may be in either Stop mode or Run mode. The transition from Stop mode to Run mode requires the occurrence of an interrupt, if enabled by the current PSW.

Once the Processor has been put into the Run mode, the current PSW controls the operation of the system. By changing the contents of the current PSW, a running program can:

- Enable and disable various interrupts.
- Switch between different system routines.

2.2 PROGRAM STATUS WORD

The Program Status Word (PSW) is an 8-bit register stored in the CTRL-card at port 7, and in SPT if I/O-RAM is not present. The low nibble of the PSW contains the current system level, where 0 is the highest level and 15 is the lowest level. The high nibble contains some control flags such as User-/System-stack flag.

2.3 INTERRUPT SYSTEM

The interrupt system of the Processor provides rapid responses to external and internal events that require service by special software routines. In the interrupt response procedure, the Processor preserves its current state and transfers control to the required interrupt handler.

Interrupts occur at various times during processing, either by an external event at a higher level than the current one, or when lowering the current PSW to a level below a previous requested interrupt level.

2.4 SYSTEM LEVELS

The system executes in one of 16 system levels, where 0 is the highest one. Level 14 is not in use. The transition from a low level to a high one is normally done through an external event, while the System Interrupt Handler controls the reversed transition. The function of the system depends on which level it is executing in:

TABLE 2-1

0	Hardware Drivers.
7	
8	Software Drivers.
9	Queue Handling.
10	Real-Time service.
11	System Queue Service.
12	Ready Queue Service.
13	Tasks, with 256 task priorities.
14	Reserved.
15	Idle Loop, Stop Mode.

2.4.1 Hardware Drivers, levels 0-7

These levels are used for the service of interrupt driven devices, where the drivers are trigged either by an external interrupt, or by a driver time-out.

2.4.2 Software Drivers, level 8

This is a reserved level that is used for non-interrupt driven devices, where the driver is trigged only by the time-out function.

2.4.3 Queue Handling, level 9

This level is entered when queues are scanned and modified by the system.

2.4.4 Real-Time Service, level 10

The update of all real-time depenent functions is done on this level.

2.4.5 System Queue Service, level 11

This level is used to coordinate events controlled by the Resource Control Blocks.

2.4.6 Ready Queue Service, level 12

The Ready Queue Service handles all task dispatching and scheduling. It is also known as the Task Master.

2.4.7 Tasks, level 13

This is the system level where all tasks execute code. This level is divided into 256 priorities, controlled by the Ready Queue Handler.

2.4.8 Idle Loop, level 15

When no more code is to be executed, the system enters this level and the Processor enters Stop mode with all interrupts enabled.

CHAPTER 3

SYSTEM CONVENTIONS

3.1 PROGRAM MODES

OS.8 programs, tasks and routines run in one of four welldefined modes. These modes are differentiated by a combination of PSW bits and status bits of an active task. Any mode not defined is not permissible. At any given instant time, the Processor is executing code in one of these modes.

They are, in increasing order of priority and privilige:

1. User Mode (UM).
2. System Mode User (SMU).
3. System Mode System (SMS).
4. Interrupt Mode (IM).

3.1.1 User Mode - UM

The UM mode is the mode in which all user tasks run. Internal and external interrupts are enabled. This mode may only be exited via interrupt or execution of SVC.

This mode is defined by:

- System Level is Task Level.
- User stackpointer and user register.
- Executing user code.
- Current Task-ID and Priority is valid.

3.1.2 System Mode User - SMU

The SMU mode is the mode in which system code is executed on behalf of a task, and is scheduled and dispatched as though it where a routine of the task. This mode is entered at a SVC. For this reason, all higher system levels are enabled.

This mode is defined by:

- System Level is Task Level.
- User stackpointer and system registers.
- Executing system code.
- Current Task-ID and Priority is valid.

3.1.3 System Mode System - SMS

The SMS mode is the mode in which the system changes critical information such as queues. This code is nonreentrant and Ready Queue Service interrupts are disabled. As no new task may be dispatched while the system is in this mode, routines that run in this mode must necessarily be short and quick to execute. This mode is exited via an external interrupt.

This mode is defined by:
- System Level higher than Task Level.
- User stackpointer and system registers.
- Executing system code.
- Current Task-ID and Priority is valid.

3.1.4 Interrupt Mode - IM

The IM mode is used only for interrupt service routines wihtin Device Drivers, Real-Time Update, System Queue Service, Ready Queue Service and Idle Loop. All higher system levels are enabled.

This mode is defined by:
- System Level other than Task Level.
- System stackpointer and system registers.
- Executing system code.

3.2 REGISTER USE

For definition, the Primary Register Set includes both resgisters AF, BC, DE, HL, Y, X and CS, IL (current Card Select and System Level). The Secondary Register Set includes AF´, BC´, DE´ and HL´.

The operating system never makes use of the secondary register set without storing it temporary on the stack. Both register sets are stored at the task on its stack, at task contex.

3.3 SUBROUTINE CONVENTIONS

Parameters are passed in registers or in memory such as SPT,TCB etc. Register modification within a subroutine is normally done according to the function of the subroutine. All other registers are normally preserved. The normal exit from a subroutine should be the address following the call. Exits to unlabled addresses are not permitted.

3.4 SVC FUNCTION CONVENTIONS

All SVC instructions causes the system to enter SMU state. On entry to the executor, the address of the parameter block is passed in register Y. It is the responsibility of the executor to perform validity checking of any parameters passed in the parameter block.

3.5 INTERRUPT CONVENTIONS

Interrupts cause control to be passed to the individual interrupt handler in IM state. On entry, the address of the control block is passed in register X.

CHAPTER 4

EXECUTIVE DESCRIPTION

4.1 INTRODUCTION

This chapter desceibes the general structure of OS.8 from a technical viewpoint. As illustrated in Figure 4-1, OS.8 is composed of several major module groupings. This chapter discusses each of these module groupings and how they interact.

The Executive contains logic for processing Supervisor Calls (SVCs) and other internal interrupts. I/O support is provided by the Data Formatter together with Device drivers and major portions of the Executive. Device Drivers are discussed separatly in chapter 5.

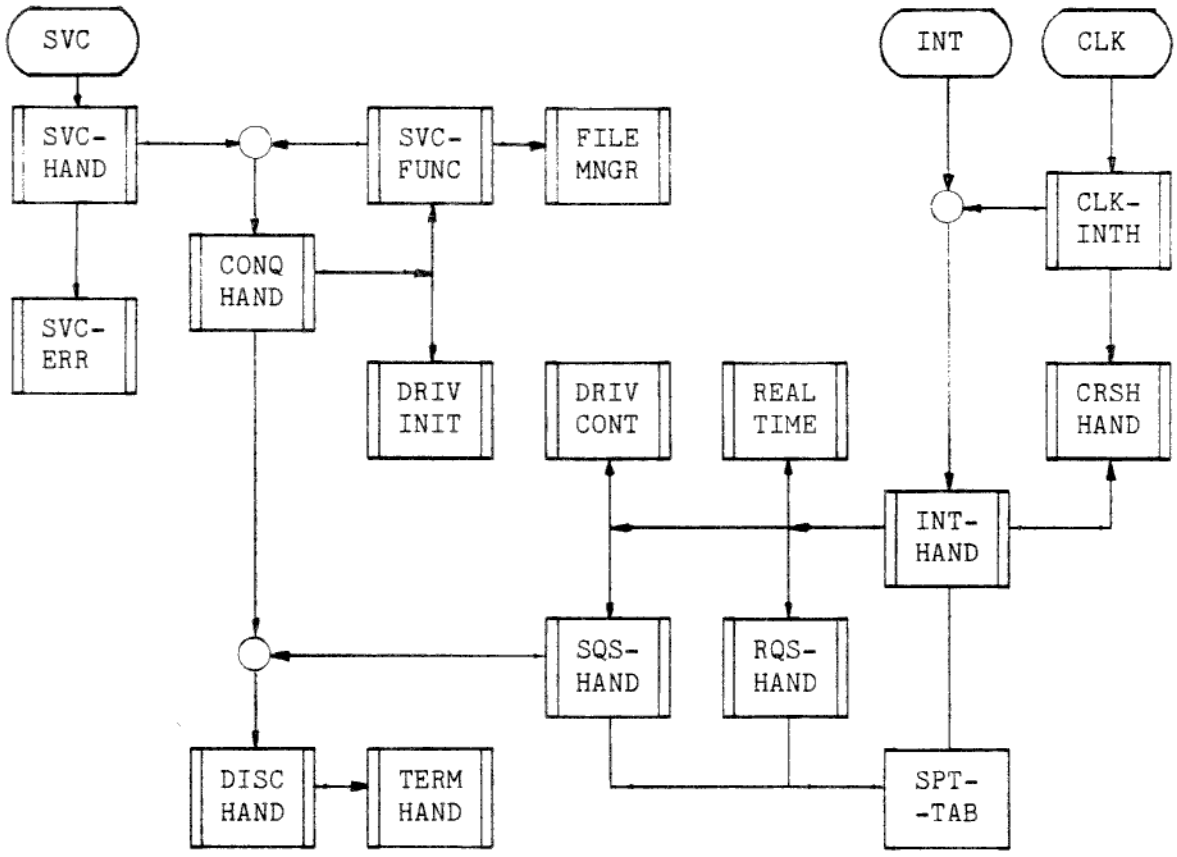


Figure 4-1. OS.8 Module groupings

4.2 SYSTEM INITIALIZATION

System initialization is performed whenever the system is started. On entry, the status of the PSW is unknown, so the first operation performed is to put the Processor into an uninterruptable, privilige state. The System Pointer Table (SPT) is cleared and then initiated by the data from Initial Value Table (IVT).

The dynamic data structures are then created from an address table located at Program Location Counter 3 (PLC3). Each item within this table points at a SVC8 parameter block without S0.FC and S0.RS. A call to the SVC8-handler is then done for each item in the table. The rest of the memory is then scanned to build the dynamic memory pool.

Finally, task number 1 is started through the SVC6-handler, and the system tries to enter the Idle Loop.

4.3 SVC HANDLER

All SVC instructions cause entry to the SVC Handler. This module performs common processing such as saving the primary register set, verifing the rest size of the stack, looks for the specified SVC number and, if found, calls the Connection Handler. Some SVC´s, such as SVC 2, have second level handlers that performs similar preprocessing.

The SVC error handler is entered when an error occurs in the SVC Handler. The error handler is also entered when the task option TCO.EMSG is present, and an error return status is found. The handler loggs a message, according to the error, on the System Console.

4.4 RESOURCES

A resource is a process or an area that can be used by a task. There are two types of resources:

- Shared, that can be used by several tasks at any time, such as reentrant processes, common areas etc.
- Exclusive, that can be used only by one task at any time, and where other tasks normally has to wait in a queue. For instans device drivers, memory allocater etc.

The system contains three major groups of resources, and each group has two reference roots, one for symbolic handling and one for numeric handling. The major groups are:

- Volumes, directory oriented mass storage devices.
- Devices, real and imaginary.
- Tasks, always treaded as a resource.

Most of the requests to a resource has to wait in a queue before they are processed. The queuing is done on priority basis and on first-come, first-served within the same priority.

Coordination of system resources is controlled through the Resource Control Block (RCB). The Connection and Disconnection Handlers contains routines to manage the RCB. The RCB is a tree structure. Each path in the tree corresponds to a group of system resources that must be coordinated as one resource. For example, the direct memory access processor, the magnetic tape controller and the magnetic tape device correspond to all the resources that must be coordinated to control the access to the magnetic tape.

Coordination is implemented by providing routines to connect to, queue to, disconnect from and release entries in the RCB-tree. The generation of RCB-trees are provided by the SVC8-handler. A task is

not connected to any required RCB until it can be connected to all required RCB´s, thus preventing deadlock conditions.

4.4.1 Connection Handler

The Connection Handler either enters a shared resource, or queues the request to an exclusive resource. When connected, the connection handler initiates the Control Block and calls the resource handler specified in the Control Block. Upon completion return from the exclusive handler, the connection handler calls the Disconnection Handler. If not completed, the connection handler either puts the task in completion wait, or returns to the SVC Handler, depending on the function code in the parameter block.

4.4.2 Disconnection Handler

The disconnection handler is called to release a task from a RCB and its tree. When released, the next request for any path in the tree is propagated, and that task is removed from connection wait state.

4.4.3 Termination Handlers

When a task is disconnected from a RCB, an optional terminator handler is called. This terminator is called in SMS/SMU, and performs some post-processing for that resource. This function is normally used by device drivers to do some hard work, that sould not be done within the interrupt handler.

4.5 EXECUTORS

All Executors acts as a subroutine of the calling task and works in SMU state. When entered, register Y points at the SVC-block and, if an exclusive resource, register X points at the Control Block. When the requested function is performed, the Executor returns with true carry to indicate completion, and register A shall then contain the return status of the request.

4.5.1 SVC Functions

Several functions within the operating system are performed by SVC calls, thus making these functions available to tasks. Each type of SVC has a handler, which normally are fully reentrant. Any new SVC handler may be added to the system, both at cold start and in run time.

The following SVC handlers are exclusive functions:

- SVC 8 Resource Manipulation.
- SVC 2.1 Memory Allocater/Deallocater.
- SVC 2.10 Bit Map Manager, within a volume.
- SVC 2.11 Directory Manager, within a volume.

4.5.2 Device Drivers

Each type of peripheral or task device has a control program, driver. These are normally fully reentrant, with the exception of the interrupt handling phase on dedicated drivers. New devices with drivers may be added to the system in the same way as SVC functions.

The initiation phase of a driver runs as a subroutine of the task in in SMU state. The interrupt handling phase normally runs with all higher levels enabled in IM state. The termination phase of a driver runs in a reentrant state although no task may be executing more than one termination phase subroutine at a time. Each device is controlled by a Device Control Block (DCB).

4.6 FILE MANAGER

This is a package that includes all the logic needed to support the OS.8 file management system. The File Manager is called from either the SVC1 Executor or the SVC7 Executor. This package then decodes each function specified by the parameter block, and invokes the necessary Executors. Each Executor that completes successfully returns to determine if any other requests are still outstanding. When all functions have been processed, or if any Executor encounter an error, the control is returned with the appropriate status. These Executors make use of the following routines contained within the File Manager:

A directory management package for maintaining information on all currently allocated files.

A bit map management package which provides a method for allocating and deleting on direct-access volumes.

The File Manager also contains SVC 1 intercept routines which intercept all I/O calls to a file.

4.7 NON-MASKABLE INTERRUPT HANDLER

This is a user defined handler, normally used to handle power down/up sequences. If not defined, a dummy should be included at system generation time.

4.8 CLOCK INTERRUPT HANDLER

This is a dedicated interrupt service handler, and is enetered at a frequency defined by the hardware, normally 100 HZ (10 ms). This routine decrements the head of the interval queue, which is sorted by by the time value. If the item elapses, the Real-Time Handler is trigged and an overflow counter will be started. All the following interrupts will now be counted in the overflow counter until the real-time service is done. Upon carry from the overflow counter, system overload, the Crash Handler will be called.

4.9 SYSTEM INTERRUPT HANDLER

Interrupts are normally handled in a standardized way. The program counter is stored on the current stack by the hardware. The primary register set is stored on the current stack. The system stack is then selected, if not already present, and the previus stack pointer is stored on this stack. The new system level is then selected.

The Interrupt Service Tables, corresponding to this new level are scanned to find the device from which the interrupt was generated. When it is found, a call is done to the specified interrupt service handler.

When the interrupt is served, return to this handler is done. If the request is completed, the RCB will be added to the System Queue and the System Queue Handler is trigged.

The rest of the interrupt linkage is scanned to look for another interrupt from a device. If so, that continuator will be called.

At the end, the propagated Software Interrupt Level is compared with the previous level and if higher, a recursive jump is done in the Interrupt Handler. Then the interrupted process is restored.

If no device was found, the Illegal Interrupt Counter is decremented, and if it becomes zero the Crash Handler is called.

4.10 CRASH HANDLER

Throughout OS.8 are checks for normally impossible states of the system. When such a condition is found the system brings itslef to a halt before futher destroying the condtions that led up to the impossible situation. This is done by calling the Crash Handler.

The Crash Handler saves all the CPU-registers in an crash diagnostic area. Then a programmed reset of the I/O system is done and a PSW is loaded that puts the system in an uninterruptable state. If the Crash Dump option is included at system generation time, a dump of the diagnostic area is done.

4.11 REAL-TIME HANDLER

When the Real-Time Handler is entered, the head of the interval queue is removed. This item is then examined to find out the reason for being trigged.

Depending on the reason, one of the following updating actions will be done:

- interval and time of day requests.
- time of day clock.
- day and year calendar including leap year handling.
- driver time-out counting.

The Real-Time Handler is trigged by:

- the Clock Interrupt Handler when the head of the interval queue is elapsed.

4.12 SYSTEM QUEUE HANDLER

The system contains a System Queue, which is a linked list of RCB. Whenever an item is added to this queue, an internal interrupt is generated to trigg the System Queue Handler. The system uses this queue to schedule events coordinated by the RCB. The handler removes each item in the queue, and calls the Disconnection Handler.

The System Queue Handler is trigged by:
- the System Interrupt Handler when a request to an interrupt driven driver is complete.

4.13 READY QUEUE HANDLER

All tasks which are currently in ready state are in a queue called the Ready Queue, which is a linked list of TCB. A task is dispatched for execution when it is the head of the ready queue. If time slicing is disabled, the task currently executing remains in this state until it voluntarily relinquishes control or until a higher priority task becomes ready. If time slicing is enabled, the task relinquishes control when its time expires, if an equal priority task is ready. Therefore, if no equal priority task is ready, the task continues to execute for another time slice.

The Ready Queue Handler is trigged by:
- the Real-Time Handler when a task becomes ready to execute after a real-time request, or when a task has exceeded its time limit.
- the System Queue Handler when a task is scheduled to execute the initiation or termination phase of an exclusive resource.
- a task which lowers its priority below another ready task.

4.14 EXECUTIVE MESSAGE

Since the Executive routines cannot issue SVC calls, all messages output by the Executive are processed by the Terminal Manager. This is accomplished by adding an event to the tasks event queue.

4.15 SYSTEM POINTER TABLE (SPT)

The SPT contains necessary information for proper operation, and is used by the system with direct address instructions. Most of the data in the SPT are pointers and roots. The System Stack, Interrupt Vectors and Interrupt Service Tables are also allocated in the SPT.

CHAPTER 5

DRIVER DESCRIPTION

5 DRIVER DESCRIPTION

Each type of peripheral or task device has a control program, driver. This driver handles all input and output for the device. It checks if transfer error occur and reports those to the requestor.

A driver has at least one entry, an Initiator and perhaps a Terminator. Physical devices may also have a Continuator and/or a Time-out/Cancel Handler. The driver is NOT allowed to use the secondary register set, without storing it first!

Drivers that transfers data on byte by byte basis, normally uses the Data Formatter to load and store the bytes.

5.1 DRIVER INITIATOR

The Initiator is called from the Connection Handler, and runs as a reentrant subroutine (interrupts are enabled) of the task issuing the I/O request. In general, the Initiator uses the information, which was stored in the DCB by the Connection Handler, to prepare the device dependent information required to perform the requested function. This is often done through the Data Formatter.

After all processing has been done, the Initiator starts the physical I/O process by causing an interrupt on the device requested. The Initiator then returns to the Connection Handler which returns control to the calling task on an I/O proceed call, or puts the calling task into I/O wait on an I/O and wait call.

Call conditions:
- Executs in SMU mode.
- The SVC-block is copied into the DCB if specified.
- The initiator address of the Data Formatter is stored in DCB.FMTE.
- The ICB is linked into time-out and interrupt chain.
- The time-out counter CCB.TCNT is initialized from CCB.TLIM if the flag DCS.INT is cleared.
- Channel seleceted on physical device.
- Register X -> DCB.
- Register Y -> SVC-block.
- Register A := the function code without S1F.NW and S1F.PRO.

Return conditions:
- Register X -> DCB.
- Register Y -> SVC-block.
- Carry flag reset means not complete.
- The flag DCS.INT in DCB.STAT must be set to enable interrupt polling and time-out checking if not complete.
- Carry flag set means complete
- Register A := return status on completion.

5.2 DRIVER CONTINUATOR

When an interrupt is detected from the device, the operating system causes control to pass to the Interrupt Service Phase, Continuator. The Continuator executes with all higher interrupt levels enabled, if not called disabled. This phase controls the actual I/O to the device by I/O instructions. On completion of the I/O, the Continuator disables the interrupts from the device and returns to the System Interrupt Handler, which adds the DCB to the system queue. The System Interrupt Handler always re-initializes the time-out counter.

Call conditions:
- Executes in IM.
- The time-out counter CCB.TCNT is initialized from CCB.TLIM.
- Channel selected on physical device.
- Register X -> DCB.
- On dedicated disabled interrupt, the driver is NOT allowded to enable the CPU, or use Register Y and Register pair BC!

Return conditions:
- Register X -> DCB.
- Carry flag reset, not complete.
- Carry flag set, complete.
- Register A := return status on completion.
- The flag DCS.INT cleared if no more interrupts are expected.

5.3 DRIVER TIME-OUT AND CANCEL

A hang-up error occurs when a device fails to generate an interrupt on an operation that was initiated. The time limit for this interrupt is computed by the Driver and stored in the DCB.TLIM, or is initiated at system generation time.

The error is detected by the Device Time-Out Manager, which decrements the time counter in the DCB. When the counter becomes zero and when no time-out is allowded (controlled by the flag DCS.TIME), the Time-Out Handler address, if specified, is scheduled. If a time-out is allowded, the continuator is called in the normal way.

The Time-Out Handler is also called when a request is canceled, and is responsible for the cancel checking!

If futher more I/O must be initiated, the Time-Out Handler causes an interrupt on the device, often by re-entering the Initiator.

Call Conditions:
- Executes in IM.
- Channel selected on physical device.
- Register X -> DCB.
- The flag DCS.INT in DCB.STAT is cleared.
- The flag DCS.TOUT in DCB.STAT is set.
- The flag RCS.CAN in RCB.STAT is set at cancel.

Return conditions:
- Register X -> DCB.
- Carry flag reset, not complete.
- The flag DCS.INT in DCB.STAT must be set to enable the checking.
- Carry flag set, complete.
- Register A := return status on completion.

5.4 DRIVER TERMINATOR

The Terminator is called from the Disconnection Handler, as a result of a System Queue interrupt. The Terminator perform post-processing on the I/O request being terminated, such as code converting or CRC calculation. If futher more I/O must be initiated, the Terminator causes an interrupt on the device, often by re-entering the Initiator.

Call conditions:
- Executes in SMU if the calling task is in I/O wait state, or in IM if no task is waiting.
- Channel selected on physical device.
- Register X -> DCB.
- Register Y -> SVC-block.

Return conditions:
- Register X -> DCB.
- Register Y -> SVC-block.
- Carry flag reset, not complete.
- Carry flag set, complete.
- Register A := return status on completion.
- The flag ICT.NOIQ must be set if the ICB shall remain on the time-out and interrupt chain.

5.5 DRIVER EXAMPLE

This is a simple example of an output driver that uses the Data Formatter. Note the critical instruction sequence when enabling the interface.

```
*
*
* DRIVER INITIATOR
* ----------------
*
* ON CALL:        X -> DCB
*                 Y -> SVC-BLOCK
*                 A := FUNCTION CODE MASKED
*
* ON RETURN:      X -> DCB
*                 Y -> SVC-BLOCK
*                CY := 0, NOT COMPLETE
*                CY := 1, COMPLETE
*                 A := RETURN STATUS
*
DRV.INIT  EQU   *
          CALL  DATA.FMT            THE DATA FORMATTER DOES THE CHECKING
          JTCS  WRONGFC             CAN´T HANDLE UN-KNOWN FUNCTIONS !
          DECR  E                   EXAMINE FUNCTION REQUESTED.
          JTZS  WRONGFC             I DON´T SUPPORT READ.
          DECR  E
          JFZS  DONE                STANDARD FUNCTIONS, ACCEPT THEM.
          CALL  CHEK.PNT            WRITE, CHECKPOINT HERE...
*
*
* DRIVER CONTINUATOR
* ------------------
*
* ON CALL:        X -> DCB
*
* ON RETURN:      X -> DCB
*                CY := 0, NOT COMPLETE
*                CY := 1, COMPLETE
*                 A := RETURN STATUS
*
DRV.CONT  EQU   *
          OR    A                   CLEAR CARRY BEFORE...
          CALL  DATA.FMT            ...LOADING THE NEXT BYTE...
          JTCS  COMPLETE
          OUT   DATA                ...THEN GIVE IT TO THE DEVICE.
          RET
```

```
*
*
* REQUEST COMPLETE
*
COMPLETE  EQU   *
          CALL  DATA.FMT            POSTPROCESS THROUGH DATA FORMATTER
*
DONE      EQU   *
          XR    A                   RETURN STATUS 0...
          OUT   C4                  ...DISABLE INTERFACE INTERRUPT...
          RBT   DCS.INT,DCB.STAT(X) ...DISABLE INTERRUPT POLLING...
          STC                       ...MARK COMPLETE...
          RET                       ...BACK TO SYSTEM.
*
*
* CALL THE DATA FORMATTER
*
DATA.FMT  EQU   *
          L     L,DCB.FMTE(X)       PICK UP THE ADDRESS TO THE CHECK-
          L     H,DCB.FMTE+1(X)     -POINTED DATA FORMATTER...
          JDR   HL                  ...AND ENTER.
*
*
* CHECKPOINT AND WAIT FOR INTERRUPT
*
CHEK.PNT  EQU   *
          POP   HL
          ST    L,ICB.CON(X)        SET UP INTERRUPT CONTINUATOR...
          ST    H,ICB.CON+1(X)
          MVI   80H,CCB.TM(X)       ...AND STATUS TEST MASK.
          LI    A,80H               LOAD INTERFACE CONTROL...
          DIS                       ...DISABLE CPU BEFORE MARKING...
          SBT   DCS.INT,DCB.STAT(X) ...INTERRUPT POLLING ALLOWED...
          OUT   C4                  ...AND ENABLE THE INTERFACE...
          ENI                       ...THEN ENABLE THE CPU.
          XR    A                   MARK NOT COMPLETE.
          RET                       BACK TO SYSTEM.
*
*
* WRONG FUNCTION CODE
*
WRONGFC   EQU   *
          LI    A,SOS.IFC           SET UP RETURN STATUS.
          STC                       MARK REQUEST COMPLETE.
          RET                       BACK TO SYSTEM.
```

CHAPTER 6

DATA FORMATTER

6.1 INTRODUCTION

The Data Formatter is a support package, that may be used by device drivers, to load or store bytes in memory. The formatter administrates the buffer pointer and counter stored in the DCB, and handles all data formats except S1F.SPEC. The register pair BC is never used, to allow dedicated drivers to use the formatter.

The formatter is always called from the driver through the address found in DCB.FMTE, in which the formatter checkpoints itself.

6.2 FORMATTER INITIATOR

The formatter initiator address is always placed into DCB.FMTE before the driver initiator is called. The formatter initiator, if called from the device driver, checks the function code of the request. If I/O transfer is requested, the formatter prepares to load or store characters for the driver.

The preparation is to compute the address to the end of the buffer+1. The buffer size is negated and stored as a counter in DCB.BCNT. This is done to speed up the buffer addressing. Then the formatter checkpoints itself to the proper handler, depending on the data format, and returns to the driver.

Call conditions:
- Register X -> DCB.
- DCB.QFC contains the function code of the request.
- DCB.QBAD is the address to the first byte in the buffer.
- DCB.QBSZ is the buffer size.

Return conditions:
- Register X -> DCB.
- Register E contains the converted function code.
- Carry flag set, unknown function code.
- Sign flag set, I/O transfer.
- Zero flag set, wait for completion.

Code in E:
```
 0 - wait for completion.
 1 - read.
 2 - write.
 3 - write with read check.
 4 - forward record.
 5 - forward file.
 6 - write file mark.
 7 - back record.
 8 - back file.
 9 - rewind.
10 - attention.
11-30 - depends on driver.
```

6.3 FORMATTER CONTINUATOR

The formatter continuator is called from the driver continuator, and will either load or store a new character. The formatter updates the buffer pointer and counter, and checks for buffer full/empty. It is impossible to load/store characters outside the buffer. If the formatter is called with carry flag set, the formatter terminator is entered.

Call conditions:
- Register X -> DCB.
- Carry flag reset, load/store next character.
- Register A := next character to store.
- Carry flag set, terminate transfer.
- Register A := termination status, NOT return status.

Return conditions:
- Register X -> DCB.
- Carry flag reset, not complete.
- Register A := next character loaded/stored.
- Carry flag set, complete.

6.4 FORMATTER TERMINATOR

The terminator is entered when told from the driver. The number of bytes transferred is calculated and stored in DCB.QBCN, the buffer pointer is restored to its initial value, and the termination status is stored in DCB.QTS. Then the formatter checkpoints itself to the initiator, to allow for re-initialization, and returns to the driver.

Return conditions:
- Register X -> DCB.
- Carry flag set, complete.
- Register A := termination status.

CHAPTER 7

SYMBIONT DESCRIPTION

7.1 INTRODUCTION

A task device driver is called from the SVC 6 Handler when the symbiont task issues a SVC 6 Wait For Event call, and when an event has occured related to the task device. Events occurs when a request to the task-device is done, either from a task issuing a request, or from the symbiont executing a SVC 4 directed to its own task-device.

The handler is always entered at its initiator address, and is called - and must return, with the same conditions as a real device driver.

However being an ordinary task, the handler has more freedom than a real device driver has, since it can execute SVC instructions.

If the handler is unable to perform the requested function, an item may be added to the event queue of the symbiont. This item triggs the symbiont, which acts upon this trigg and re-triggs the task device handler, when it is able to perform the requested function. The handler may then terminate the request.

CHAPTER 8

SYSTEM GENERATION

8.1 INTRODUCTION

This chapter provides the information for the system planner and personnel to plan and configurate and generate an OS.8 system. It describes the statements used to configurate a system, and includes guidelines for selecting system generation (SYSGEN) parameters which produce the desired system.

The system may be configured to support a variety of hardware and application environments, from a large system to a single user, batch oriented, program development system. This capability for tailoring the OS.8 system to the desired user environment ensures the efficient use of resources (i.e., memory, peripherals, and time) for accomplishing the desired functions.

The system generation is accomplished by running the OS.8 Task Establisher to produce a single absolute load module. The configuration is done through the command stream to the ESTAB.

Some modules in the OS.8 are located at different PLC´s. The order between the PLC´s defines the memory use in the system:

- All modules, except these below, PLC 0.
- SPT, PLC 1.
- Cold start, PLC 2.
- Initiation table, PLC 3.
- SVC 8, PLC 4.
- Initiation data, PLC 5.

8.2 GENERATION PARAMETERS

The command stream contains a number of commands to control the PLC order, to select the desired SVC functions, volumes, devices and tasks, and to specify some system generation constants. Refer to the command stream that is delivered together with the Object Libraries containing the Operating System.

8.2.1 PLC Control

The following commands controls the Program Location Counters (PLC) orders and thier locations:

PLCNR-selects a PLC for a subsequent PLCBASE command.

PLCBASE-will origin the PLC specified by a PLCNR command.

PLCORDER-specifies in which order the PLC-segments are to be put into memory.

PLCSTART/PLCEND-includes the global symbolic name of each PLC.

8.2.2 Module selection

To include the SVC functions, volumes, device and tasks, the user just selects them with the SELECT command.

8.2.3 Generation constants

The contants are used to define fixed functions in the system. They are specified by the EQU command.

SGN.YEAR, SGN.MNTH, SGN.DAY
: These constants specifies the date that shall be the current date after the system is started. Normally set to the day of the generation. If SGN.DAY is set to zero, the update of date and time-of-day is inhibited.

SGN.APPL-is a user defined value in the range 0-255, to indentify his own application. This value is stored in SPT.

SGN.TSKX-the maximum number of tasks which may be in the system at any one time.

SGN.MBOT-the lowest available RAM memory address that may be used by the system.

SGN.MTOP-the highest available RAM memory address that may be used by the system.

SGN.FREQ-the real-time clock frequency in HZ, normally 100 HZ.

SGN.RESL-the real-time clock interval in milli-seconds (ms). It is normally expressed in the form 1000/SGN.FREQ.

SGN.HOUR-hour where to switch day, expressed in the form HH:MM.

SGN.TLIM-the global task time-slice limit in ms.

SGN.DLIM-the device time-out scan interval in ms, normally 100 ms.

SGN.IIC-the maximum number of illegal interrupts allowed before crash.

SGN.FMGB-the number of file manager buffers to be allocated. This parameter should be set to 0, if the File Manager is excluded.

CHAPTER 9

DATA STRUCTURES

9.1 INTRODUCTION

This chapter presents the formats of System Control Blocks and table entry. Most fields are self explanatory; those which are not are explained following the figure for that Control Block.

Every public resource has to be defined at a numeric level, and optional at a symbolic level. This is done by a Resource Reference Table (RRT) for the numeric level, and by a Resource Mnemonic Table (RMT) for the symbolic level. These tables are linked into the group linkage in which they should be found.

An exclusive resource must also have a Resource Control Block (RCB), which controls the access to the resource, and establish the queue into the resource.

9.2 GENERAL NAMING CONVENTIONS

Each field within a data structure is identified by its name and a descriptive title. Offsets are given in the form (DD), where DD is the offset in decimal.

9.2.1 Data Structures.

All data structures (defined by TAB$STR) in OS.8 are named with three character symbolic names, e.g. DCB, SPT. All fields within these structures are defined by a name of the form SSS.FFFF, where <SSS> is the structure name, and <FFFF> is the field name.

9.2.2 Bit Declarations

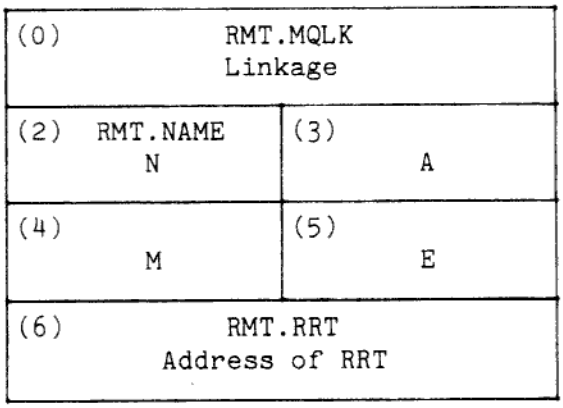
Certain fields in a data structure contain flag bits to denote information. These flag bits are manipulated with either bit instructions (e.g., SBT, RBT, TBT) or logical immediate instructions (e.g., NI, OI, XI). All bits within a field are defined by a name of the form SSF.BBBB where <SS> are two characters which refers to the structure name, <F> is a character which refer to the field, and <BBBB> identifies the function of the flag bit.

For example, in the DCB (Device Control Block) there is a field DCB.STAT which contains the status bits. Bit 0=0 means that the driver is not waiting for an interrupt, neither a time-out. Bit 0=1 means that the driver is waiting for an interrupt or a time-out. The bit mask definition of this flag is:

```
DCS.INT   EQU   1                 Interrupt enabled flag.
```

9.3 RESOURCE MNEMONIC TABLE (RMT)

This table defines the resource in a symbolic way, and contains the name of the of the resource, and the address to the corresponding RRT.



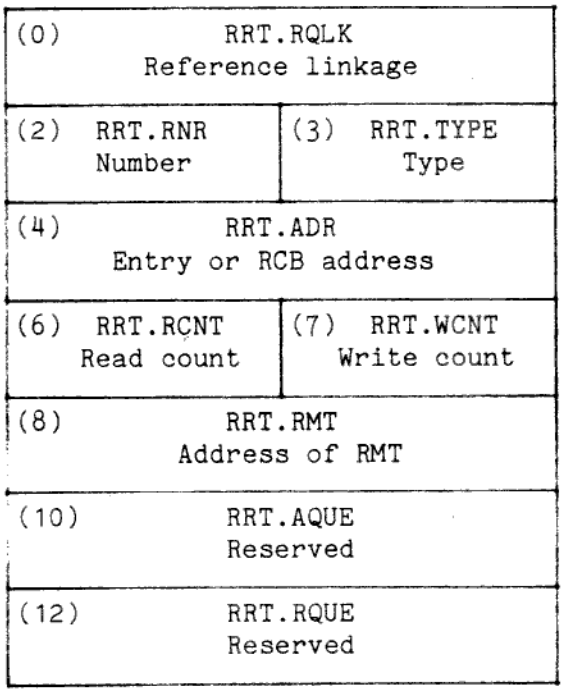
RMT.MQLK links all RMT together into the group where they should be found. The linkage is sorted by the first character in RMT.NAME in ascending order.

RMT.NAME holds the symbolic name of the resource. The characters must be printable ASCII characters.

RMT.RRT points at the corresponding RRT.

9.4 RESOURCE REFERENCE TABLE (RRT)

This table defines the resource in a numeric way, and contains the number and type of resource. It also holds the assign linkage and the entry to handler on shared resources, or the address to the RCB on exclusive resources.



RRT.RQLK links all RRT together into the group where they should be found. The linkage is sorted be the RRT.RNR in ascending order.

RRT.RNR is the internal number of the resource within the group.

RRT.TYPE specifies the resource type.

RRT.PURE	00	Re-entrant.
RTT.RCB	01	Exclusive.
RTT.RRT	10	Dummy.
RTT.AREA	11	Area.
RTT.DIR	.... .1..	Directory structured.
RTT.SVC	.... 1...	Entry at all SVC-calls.
RTT.OFFL	...1	Off line.
RTT.PROT	..1.	Protected.
RTT.NFST	.1..	Non-file structured.
RRT.NRMW	1...	Non-removable.

RRT.ADR is either the entry on a shared resource, or address of RCB.

RRT.RCNT/RRT.WCNT are used as read and write assign counters by the SVC-7 handler. A negative value means exclusive assigned.

RRT.RMT points at corresponding RMT.

9.5 NODES

The most central part in the operating system is the node. The node represents the task at connection to an exclusive resource. The node is linked into the requested resource queue.

(0) NOD.LNK Linkage	
(2) NOD.CPRI Task priority	(3) NOD.CTCB Task number
(4) NOD.SVC Address of parameter block	
(6) NOD.RNR Resource number	(7) NOD.EVNT Event code
(8) NOD.RCB Address of RCB	
(10) NOD.AQLK Assign linkage	
(12) NOD.OWN Reserved	

NOD.LNK links all nodes together from a RCB. The linkage is sorted by the NOD.CPRI in ascending order.

NOD.CPRI is the task priority at the time when the request was done.

NOD.CTCB is the task number.

NOD.SVC contains the address of the parameter block that was specified when the SVC was done.

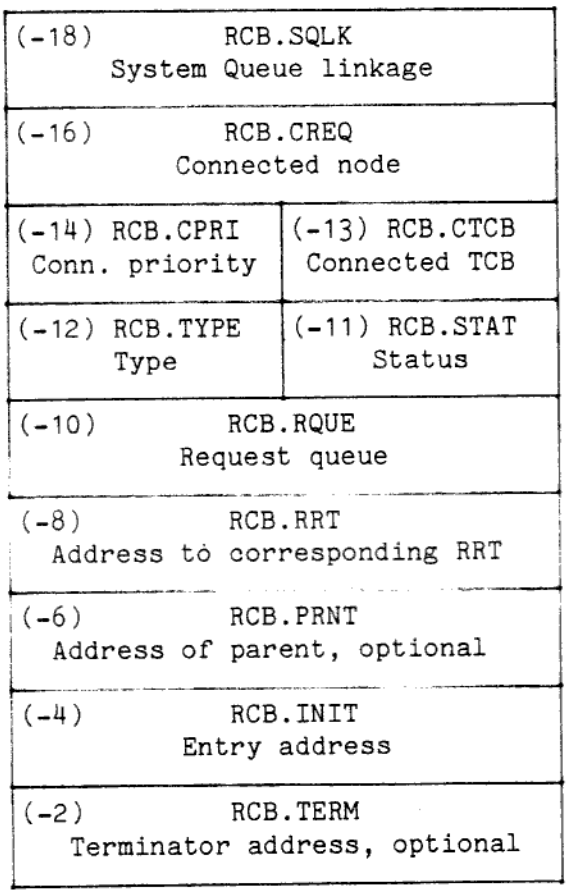
NOD.RNR holds the number of the requested resource.

NOD.EVNT is the SVC-number.

NOD.RCB points at the requested RCB.

9.6 RESOURCE CONTROL BLOCK (RCB)

This table holds all information about an exclusive resource, such as the status on the resource, the request queue into the resource and entry to the resouce. The RCB is normally the head of all control blocks in the operating system.



RCB.SQLK is used to queue the RCB to System Queue at termination phase.

RCB.CREQ is the address to the node which is in progress.

RCB.CPRI will always hold the highest connected priority, or the highest queued request´s priority.

RCB.CTCB holds the connected task-ID.

RCB.TYPE defineds the type of the RCB:

RCT.RCB	.... .000	RCB indentifier.
RCT.DCB	.... .001	DCB - " -.
RCT.TCB	.... .010	TCB - " -.
RCT.FCB	.... .011	FCB - " -.
RCT.VCB	.... .100	VCB - " -.
RCT.AREA	.... .101	Exclusive area.

RCT.PRNT	.... 1...	Coordination parent present.
RCT.DESC	...1	Descendent present.
RCT.NW	..1.	No-wait isn´t supported.
RCT.PRO	.1..	Proceed - " -.
RCT.NAB	1...	Non-abortable, cant be cancelled.

RCB.STAT specifies the status on the resource. The common bits are described below, and the rest is defined at each control block.

RCS.BUSY	1	Busy, the resource is active.
RCS.OFFL	.... ..1.	Off line, no access is allowded.
RCS.CAN	.... .1..	Cancel, the request in progress is canceled.
RCS.NW	.... 1...	No-wait request in progress.
RCS.ONSQ	...1	Present on System Queue.

RCB.RQUE will queue every request node when the resource is busy. The nodes will be sorted in ascending priority order.

RCB.RRT contains the address to the corresponding Resource Reference Table (RRT).

RCB.PRNT contains the address to the parent, to whom the request must be coordinated to. For instans, a disc controller that supports more than one disc, but only one at each time.

RCB.INIT contains the entry address of the resource, or the address to an exclusive area.

RCB.TERM is the address to an optional termination handler.

9.7 DEVICE CONTROL BLOCK (DCB)

The DCB is used by the Connection, Disconnection and System Interrupt Handlers to idendtify characteristics of each device in the system, and to serve as a work space for drivers during an I/O request.

(0) DCB.ATTR Attributes		(2) DCB.RECL Record length	
(4) DCB.CODE Device code	(5) DCB.SNR Device number	(6) DCB.TYPE Device type	(7) DCB.STAT Device status
(8) DCB.FMTE Checkpoint for data formatter		(10) DCB.QPAR SVC-blk size	(11) DCB.QADR Reserved
(12) DCB.QSVC Address of SVC-blk in progress		(14) DCB.QFC Function code	(15) DCB.QRS Return status
(16) DCB.QLU Logical unit	(17) DCB.QTS Term. status	(18) DCB.QBAD Buffer address	
(20) DCB.QBSZ Buffer size		(22) DCB.QBCN Byte count	
(24) DCB.QRND Random address			
(28) DCB.RAND Current random address on device			
(32-) DCB.OPT Optional, depends on driver			

DCB.ATTR specifies the request types that the device support:

```
ATR.READ  .... .... .... ...1  Read.
ATR.WRIT  .... .... .... ..1.  Write.
ATR.FASC  .... .... .... .1..  ASCII formatting.
ATR.SPEC  .... .... .... 1...  Special formatting.
ATR.RND   .... .... ...1 ....  Random access.
ATR.IACT  .... .... ..1. ....  Interactive device.
          .... .... .1.. ....  Reserved.
          .... .... 1... ....  Reserved.

          .... ...1 .... ....  Reserved.
ATR.FR    .... ..1. .... ....  Forward record.
ATR.FF    .... .1.. .... ....  Forward file.
ATR.WF    .... 1... .... ....  Write filemark.
ATR.BR    ...1 .... .... ....  Back record.
ATR.BF    ..1. .... .... ....  Back file.
ATR.RW    .1.. .... .... ....  Rewind.
ATR.ATTN  1... .... .... ....  Attention.
```

DCB.RECL defines the maximum length of a record for the Device. Established at SVC8 time. Used by the Driver to truncate requests larger than maximum. Record length zero means variable record length.

DCB.CODE specifies the device type. Established at SVC8 time.

DCB.SNR will contain the system device number at request initiation.

DCB.TYPE describes the type:

DCT.ICB	1	ICB present.
DCT.DEDI	.... ..1.	Dedicated interrupt service, not scan.
DCT.ENI	.... .1..	Reserved.
	.... 1...	Reserved.
DCT.TASK	...1	Task device.
DCT.DUAL	..1.	Dual DCB present.

DCB.STAT holds the current status:

DCS.INT	1	Interrupt enabled, scan and time-out.
DCS.TIME	.... ..1.	On time-out, call continuator.
DCS.TOUT	.... .1..	Time-out generated.

DCB.FMTE is a checkpoint field used by the Data Formatter.

DCB.QPAR specifies the number of bytes that sould be copied from the SVC parameter block to the DCB. If 0, no copy is done.

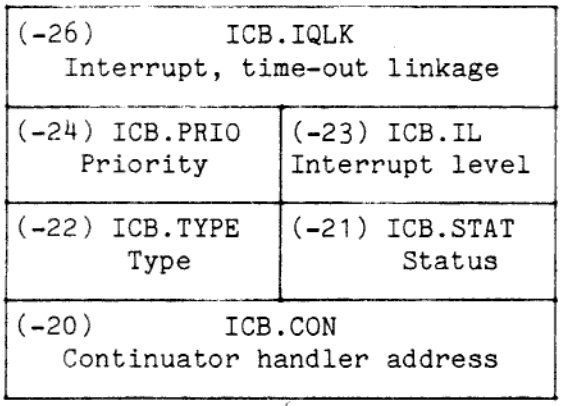
DCB.QADR is reserved for future use.

DCB.Qxxx corresponds to the SVC 1 parameter block.

DCB.RAND is used as work field to hold the current random address.

9.8 INTERRUPT CONTROL BLOCK (ICB)

This table is used by the interrupt system and by the real-time handler.



ICB.IQLK is used to link all active devices together, within the same level.

ICB.PRIO is used as sort item when linking the ICB.

ICB.IL is the system level for the device.

ICB.TYPE specifies the ICB:

ICT.CCB	1	Channel Control Block (CCB) present.
ICT.NOIQ	.... ..1.	Makes the ICB resident on the chain.

ICB.STAT holds the current status:

ICS.ONIQ	1	Present on the chain.
ICS.SINT	.... ..1.	System software interrupt generated.
ICS.TOUT	.... .1..	System time-out generated.

ICB.CON specifies the continuator handler to be called at interrupt.

9.9 CHANNEL CONTROL BLOCK (CCB)

The channel control block is used by the interrupt system to scan the interfaces.

(-38) CCB.CS Channel address	(-37) CCB.TM Status test mask
(-36) CCB.FLGS User flags	(-35) CCB.XOR Status xor-mask
(-34) CCB.TLIM Device time-out limit	
(-32) CCB.TCNT Device time-out counter	
(-30) CCB.THND Device time-out handler	
(-28) CCB.SUBC Users second level cotinuator	

CCB.CS is the card select code of the interface.

CCB.TM is the interrupt test mask, that is used with a Logical And on the interface status. If the result is non-zero, the interrupt generated device is found, and the continuator is then called.

CCB.FLGS is for driver use.

CCB.XOR is a bit pattern that is used with a Logical Exclusive-Or on the interface status before the CCB.TM is used. This bit pattern specifies the bits to be inverted, and is intiated to -1 at SVC8 establish.

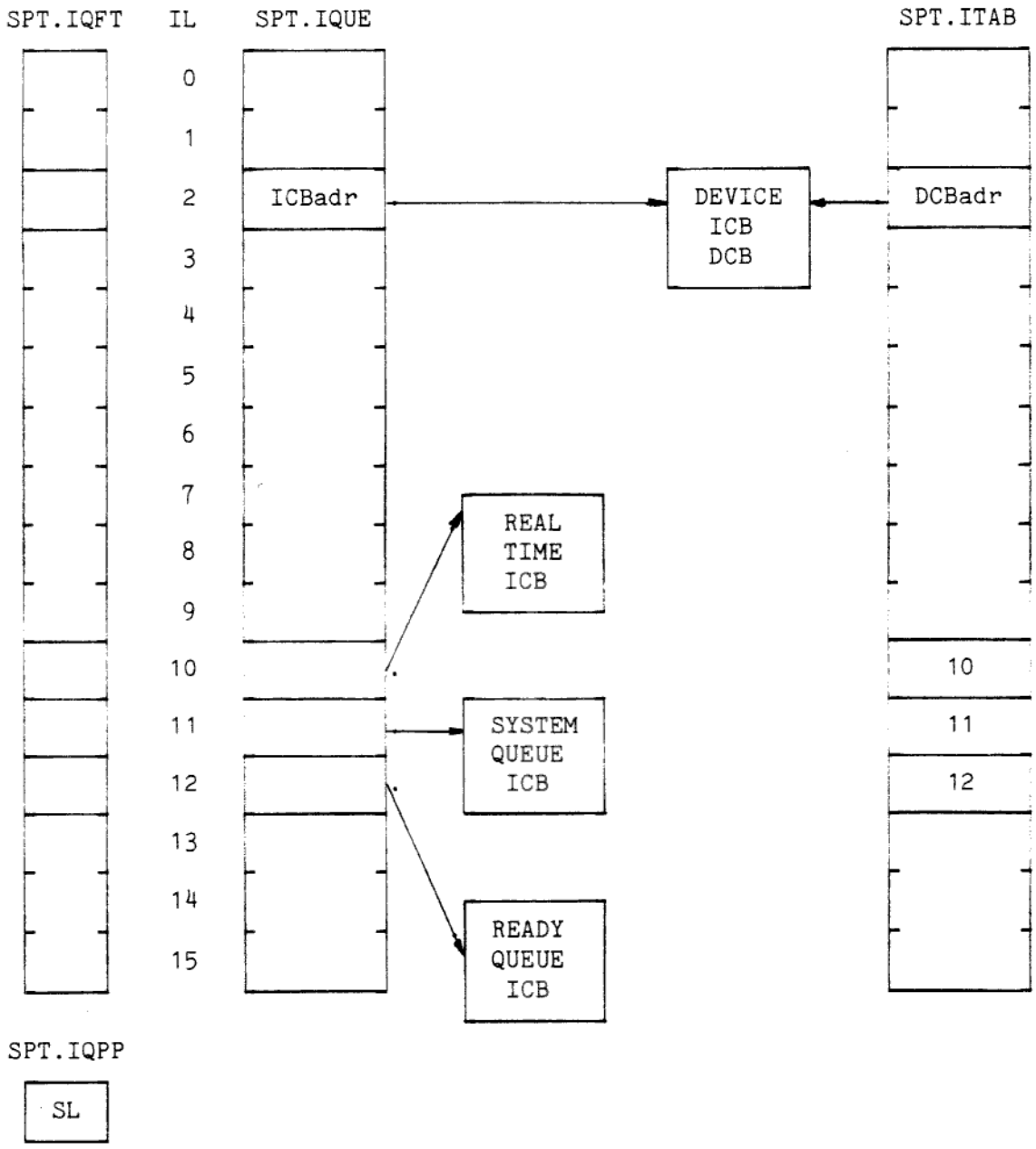
CCB.TLIM holds the time-out limit for the device. It is normally a value modulo 100 ms.

CCB.TCNT is the time-out counting field. It is reset by the system before calls to driver handlers.

CCB.THND contains the address to a time-out handler for the device. If not specified, 0 address, the system handles the time-out in a standardized way.

CCB.SUBC is a field free for driver usage. It is often used as a checkpointing field.

9.10 INTERRUPT SERVICE TABLES



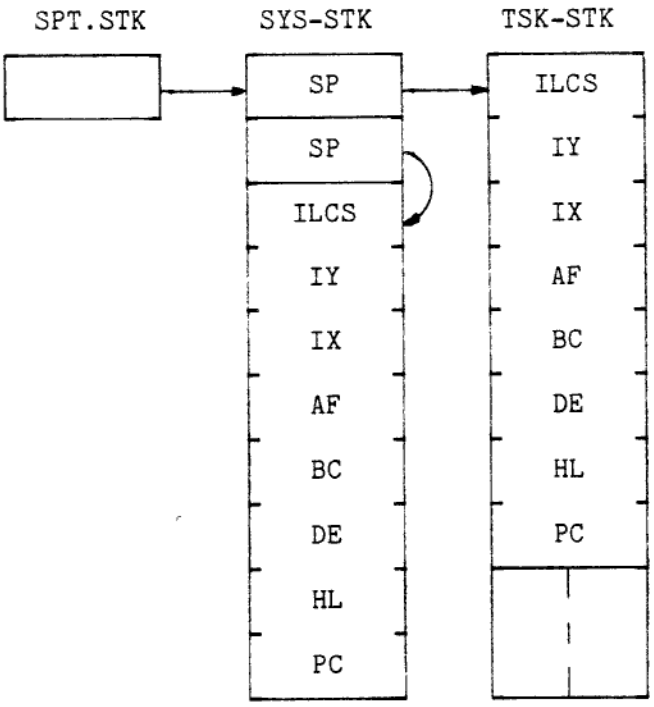
SPT.IQFT-is a byte vector used to mark a soft trigged system level.

SPT.IQPP-holds the highest propagated soft trigged system level.

SPT.ITAB-contains either the DCB-address of a dedicated device, or the system level at device scan.

SPT.IQUE-is a root-table that links together all active devices on the same level. The table is used at both time-out and interrupt handling.

9.11 STACK STRUCTURES



SPT.STK, Contains the address to the system stack that should be used at an interrupt.

SYS-STK, The structure of the system stack when the system is executing in Interrupt Mode.

TSK-STK, The structure of a task stack when a task is interrupted.

APPENDIX A

GLOSSARY AND SHORTENINGS

CCB	Channel control block.
CDT	Channel description table.
DCB	Device control block.
DDT	Device description table.
DMT	Device mnemonic table.
DRT	Device reference table.
EDT	Extened descriptor table.
FCB	File control block.
FMT	File mnemonic table.
FRT	File reference table.
ICB	Interrupt control block.
IDT	Interrupt description table.
RCB	Resource control block.
RDT	Resource descriptor table.
RMT	Resource mnemonic table.
RRT	Resource reference table.
SPT	System pointer table.
STB	System table.
SVC	Supervisor call.
SYP	System pointer table structure.
TCB	Task control block.
TDT	Task description table.
TMT	Task mnemonic table.
TRT	Task reference table.
VCB	Volume control block.
VDT	Volume description table.
VMT	Volume mnemonic table.
VRT	Volume reference table.

APPENDIX B

CRASH LAYOUT

```
II=ii CC=cc IL=il CS=cs TP=tp TN=tn TC=ttcb SS=sspp     (SPT)
PC=ppcc SP=sspp  F=flag  A=aa BC=bbcc DE=ddee HL=hhll  (Primary)
 Y=yyyy  X=xxxx  F=flag  A=aa BC=bbcc DE=ddee HL=hhll  (Secondary)
SS
addr aabbccdd .... .... .... .... .... .... vvxxyyzz   (System stack)
aadr aabbccdd .... .... .... .... .... .... vvxxyyzz
SP
addr aabbccdd .... .... .... .... .... .... vvxxyyzz   (Current stack)
addr aabbccdd .... .... .... .... .... .... vvxxyyzz
```

System pointer table (SPT):

II = illegal interrupt counter.
CC = crash code.
IL = interrupt level, system mode if less than 80.
CS = card selection code.
TP = task priority.
TN = task number.
TC = task control block.
SS = system stack-pointer.

Primary register set:

PC = program location counter.
SP = program stack-pointer.
 F = condition code.
 A = A-register.
BC = BC-register pair.
DE = DE-register pair.
HL = HL-register pair.

Secondary register set:

 Y = Y-index register.
 X = X-index register
 F = condition code.
 A = A-register.
BC = BC-register pair.
DE = DE-register pair.
HL = HL-regsiter pair.

APPENDIX B (Continued)

CRASH CODES

NR	SYMBOLIC	DESCRIPTION
1	CC.COLD	SVC-8 failed in cold start.
2	CC.USER	User crash entry.
3	CC.ZERO	Entry trough address zero.
4	CC.NMI	Too many non-maskable interrupts.
5	CC.VINT	Too many un-exspected vector interrupts.
6	CC.INTER	Too many illegal interrupts in 4680 I/O-structure.
7	CC.TIME	Real-time service never done, loops on levels =< 10.
8	CC.SETSL	Attempt to set up a non-existing level.
9	CC.TRGSL	Attempt to trigg a non-existing level.
A	CC.ICBIL	Attempt to trigg an ´ICB´ with a non-existing level.
B	CC.SINT	Attempt to execute on a non-existing level.
C	CC.ADDIL	Attempt to add a ´DCB´ on a non-existing level.
D	CC.RMVIL	Attempt to remove a ´DCB´ from a non-existing level.
E	CC.RMVIQ	Failed to remove a ´DCB´ from an interrupt chain.
F	CC.ADDRQ	Attempt to add a ´TCB´ twice to Ready-Q.
10	CC.RMVRQ	Failed to remove a ´TCB´ from Ready-Q.
11	CC.RMVBQ	Failed to remove a ´TCB´ from Buffer-Q.
12	CC.ADDSQ	Attempt to add a ´RCB´ twice to System-Q.
13	CC.RMVSQ	Failed to remove a ´RCB´ from System-Q.
14	CC.TASK	Attempt to find a non-generated task-number.
15	CC.STOP	Attempt to stop a task in System Mode.
16	CC.DESC	Attempt to connect at decendent in a tree.
17	CC.PROP	Attempt to propagate priority on a non-existing task.
18	CC.TTDEV	Attempt to trigg a non-existing task-device.
19	CC.DISP	Attempt to dispatch a non-existing task.
1A	CC.DISC	Attempt to disconnect a non-existing task.
1B	CC.RMVTQ	Failed to remove a node from ´SPT.MLNK´.
1C	CC.NODE	Attempt to release a non-existing node.
1D	CC.QUEUE	A queue has become a circular list.
1E	CC.RMTQ	Failed to remove a ´RMT´ from symbolic chain.
1F	CC.RRTQ	Failed to remove a ´RRT´ from numeric chain.
20	CC.TCAN	Failed to cancel a task.
21	CC.AMEM	Attempt to allocate memory twice.
22	CC.RMEM	Attempt to release memory twice.

APPENDIX B (Continued)

CRASH CONDITIONS

NR	CONDITION
1	A := Return status, Y -> SVC-blk, HL -> next in cold start table.
2	Specified by user.
3	SP -> last item pushed on stack.
4	SP -> previous A.
5	SP -> previous A.
6	A := current system level, X -> previous DCB in chain.
7	SP -> BC, AF, HL, PC.
8	C := non-existing system level.
9	L := non-existing system level.
A	A := non-existing system level in an ICB.
B	A := non-existing system level.
C	A := non-existing system level in an ICB, X -> DCB.
D	A := non-existing system level in an ICB, X -> DCB.
E	X -> DCB, DE -> ICB.IQLK, HL -> root of chain.
F	X -> TCB.
10	X -> TCB, DE -> TCB.RQLK, HL -> SPT.RQUE.
11	X -> TCB, DE -> TCB.RQLK, HL -> SPT.BUFQ.
12	X -> RCB, DE -> RCB.SQLK.
13	X -> RCB.
14	B := task number.
15	SP -> AF1, BC1, DE1, HL1, CSIL, Y, X, AF, BC, DE, HL, PC.
16	X -> RCB, Y -> SVC-blk, B := SVC-type, C := resource number.
17	B := task number, (SP) := RCB, DE -> xxx.CPRI.
18	B := task number, DE -> node, (SP) := DCB.
19	B := task number, (SP) := RCB.
1A	B := task number, SP -> Y, X, PC.
1B	DE -> node, HL -> SPT.MLNK.
1C	DE := invalid node address.
1D	B := 0 !
1E	C := resource number, DE -> RMT.
1F	C := resource number, DE -> RRT.
20	X -> TCB.

APPENDIX C

ERROR CODES

COMMON ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
000	00	0	SOS.OK	No error.
001	01	1	SOS.EON	End of nodes.
002	02	2	SOS.IFC	Invalid function code.
003	03	3	SOS.PRO	Can't connect at unconditional proceed.
004	04	4	SOS.OFFL	Off line.
005	05	5	SOS.PRES	Not present in this system.
006	06	6	SOS.NYET	Not yet implemented function.
007	07	7	SOS.CAN	Request is cancelled.
010	08	8	SOS.SVC	Invalid SVC function.

SVC-1 I/O ERROR CODES

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
012	0A	10	S1S.LU	Illegal LU, LU not assigned.
013	0B	11	S1S.AM	Invalid access modes.
014	0C	12	S1S.TOUT	Time-out.
015	0D	13	S1S.DWN	Device down.
016	0E	14	S1S.EOF	End-of-file.
017	0F	15	S1S.EOM	End-of-media.
020	10	16	S1S.RER	Recoverable error.
021	11	17	S1S.UNR	Unrecoverable error.
022	12	18	S1S.RND	Invalid random address.
023	13	19	S1S.NRND	Non-exsistent random address.

SVC-2 SUBFUNCTION ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
024	14	20	S2S.ISB	Illegal sub-function number.

SVC-3 TIMER ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
036	1E	30	S3S.PAR	Invalid timer parameter.

SVC-4 TASK DEVICE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
050	28	40	S4S.ASGN	Not assigned.
051	29	41	S4S.TYPE	Invalid device type.

SVC-5 LOADER ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
062	32	50	S5S.TID	Illegal task-id.
063	33	51	S5S.PRES	Task present.
064	34	52	S5S.PRIO	Illegal priority.
065	35	53	S5S.OPT	Illegal option.
066	36	54	S5S.CODE	Illegal code/item at load.
067	37	55	S5S.SIZE	Overlay don't fit.

SVC-6 TASK ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
074	3C	60	S6S.TID	Illegal task-id.
075	3D	61	S6S.PRES	Task present.
076	3E	62	S6S.PRIO	Illegal priority.
077	3F	63	S6S.OPT	Illegal option.
100	40	64	S6S.EQUE	Event queue disabled.
101	41	65	S6S.STAT	Invalid task status.

SVC-7 FILE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
106	46	70	S7S.ASGN	Assignment error, double assign.
107	47	71	S7S.AM	Illegal access modes.
110	48	72	S7S.SIZE	Size error.
111	49	73	S7S.TYPE	Type error.
112	4A	74	S7S.FD	Illegal file descriptor.
113	4B	75	S7S.NAME	Name error.
114	4C	76	S7S.KEY	Invalid key.
115	4D	77	S7S.FEX	File exist error.

SVC-8 RESOURCE ERRORS

OCT	HEX	DEC	SYMBOLIC	ERROR TEXT
120	50	80	S8S.ID	Illegal resource-id.
121	51	81	S8S.CLAS	Invalid resource class.
122	52	82	S8S.PRES	Resource already present.
123	53	83	S8S.PRNT	Parent not present.
124	54	84	S8S.DUAL	Dual DCB not present.
125	55	85	S8S.RCB	Invalid RCB-type.
126	56	86	S8S.EOM	End-of-memory.

APPENDIX D

DIFFERENCES FROM PREVIOUS REVISIONS

Between R1.00 and R2.00

SVC2.8 It is now possible to expand the mnemonic table.

SVC3 Interval in milli-seconds, not clock tics.

SVC5 Overlay handling added.

SVC6 Event queue handling, field contents are changed. New functions: S6F.TSKW, S6F.ADDQ and S6F.STSW.

SVC8 Task Descriptor Table (TDT): Stack size is moved and replaced with additional task size. Task slice limit is now expressed in milli-seconds.

CHK.S1F The returned index is now in register E.

Between R2.00 and R2.50

SVC2.3 Modifier decoding added.

SVC2.7 Slice handling added.

SVC2.12 Possibile to specify user file-handler.

SVC6 Suspend function added.

SVC7 Modifiers changed.

SVC8 RTT.SVC added, entry at all SVC calls.

CCB.XOR Drivers, CCB.HST is renamed. This field is used by the Interrupt Handler in the following way:

```
-
INP   STAT        Read interface status.
X     CCB.XOR(X)  Reverse the interesting bits.
N     CCB.TM(X)   Isolate the bits to be tested.
JFZ   DRIVER      If match, call the driver.
-
```