RSX-180/280

System Services

Kernel Version 6.37

January, 2023

Table of Contents

Introduction

RSX180 and RSX280 are multi-tasking, multi-user, multi-terminal Operating Systems (OSes) for the Zilog Z180 and Z280 CPUs, modeled after DEC's RSX-11M OS for the PDP-11 series of mini-computers.

The RSX180 and RSX280 OSes provide a number of system services[1] that allow tasks to interact with peripheral devices and with other tasks, and to control their execution. The system services enable tasks to:

- Obtain task and system information
- Perform I/O functions
- Measure time intervals
- Spawn other tasks
- Communicate and synchronize with other tasks
- Suspend and resume task execution
- Exit

This document describes the services available by the RSX180/280 kernel[2] version 6.37 from an assembly-language programming perspective. High-level languages will typically encapsulate the access to the system services in a convenient set of library routines.

1 Also known as *directives* in the RSX-11M world. Further in this document we may equally use *directive* when referring to a *system call* or *system service*.

2 Also known as *Executive* in the RSX-11M world.

1. Using System Services

1.1. System call processing

System call invocation differs between RSX180 and RSX280:

- **RSX180** System Services are invoked via a Z80 RST (Restart) instruction, with the function code following the call. The exact RST instruction is defined during system generation, and defaults to RST 20h. A jump vector, set by the kernel when the task is first invoked, transfers the execution to an auxiliary routine in the common region (Figure 1). The routine saves the task's CPU registers, maps the kernel into memory and then uses a dispatch table to call the appropriate code that processes the system service. Returning from the system call is done in reverse: the execution is transferred to a routine in common memory that maps the user task back into memory, restores the task's CPU registers and then returns to the instruction following the system call.
- **RSX280** System Services are invoked via a Z280 SC (System Call) instruction with the function code as argument. The SC instruction generates a trap that causes the CPU to switch to system state, mapping the kernel into memory (Figure 2). The kernel code uses a dispatch table to call the appropriate routine that processes the system service. Returning from the system call is done via a RETIL (Return from Interrupt) instruction after restoring the task's CPU registers; the instruction causes the CPU to switch to user state, mapping the user task back into memory and returning to the instruction following the system call.

The SYSFN.INC system include defines symbolic names (mnemonics) for the different system services, as well as a 'SC' macro that expands to the particular system call instruction sequence for the target system. The macro allows system services to be invoked using a system-independent syntax. The argument to the macro is the symbolic name of the service, e.g.:

```
SC      .ALUN
```

In both RSX180 and RSX280, arguments to the system call are passed via CPU registers[3] and are consistent (compatible) between the two systems, allowing a non-privileged task written for one system to be recompiled and executed on the other with minimal or no changes.

The system services normally preserve all CPU registers, except the accumulator and flags which are used to return success/error status back to the application.

3 Unlike RSX-11M, where arguments are pushed on the stack or passed via a data structure.

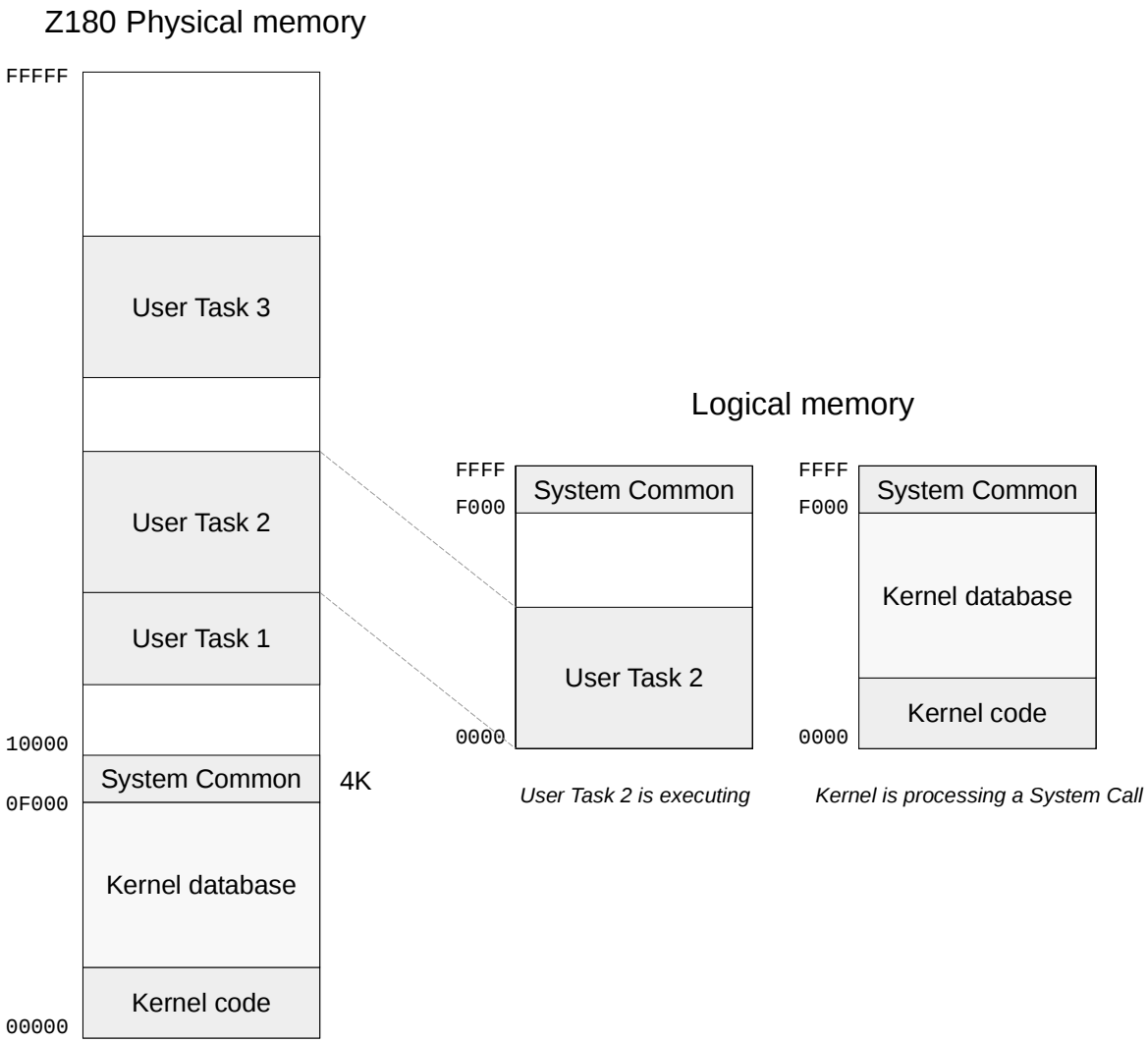

Figure 1: RSX180 Memory Map

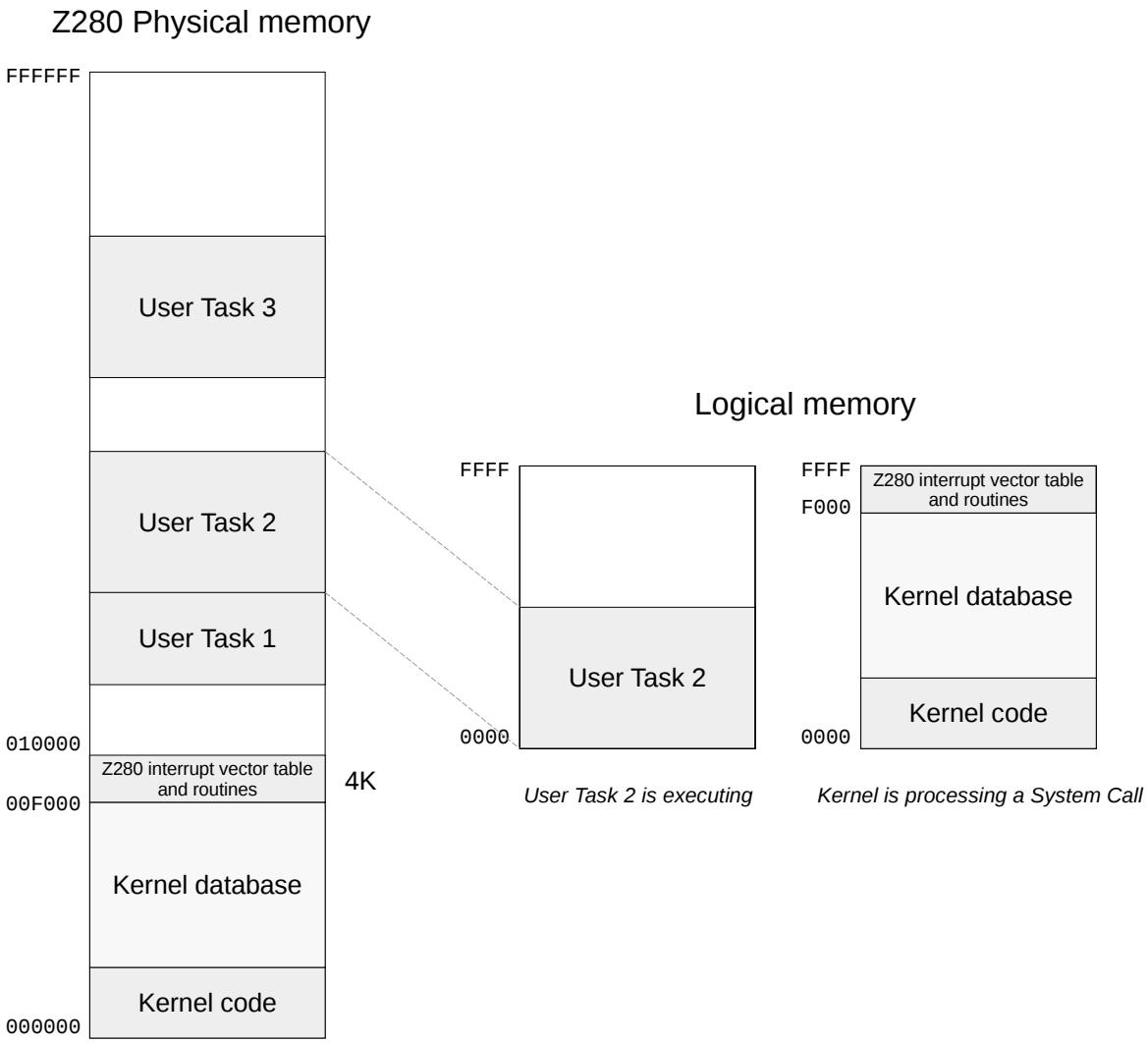

Figure 2: RSX280 Memory Map

1.2. Error conditions

The carry flag of the CPU is used to indicate whether the system call operation succeeded or failed: a carry flag clear indicates success, while a carry flag set indicates that an error occurred. A successful system call will normally return zero (E.OK) in the CPU accumulator[4], while a rejected call will return in the accumulator a code describing the reason for the failure.

4 With a few exceptions, e.g. Read Single Event Flag (.RDEF), Set Event Flag (.SETF) and Clear Event Flag (.CLEF).

Error codes under RSX180 and RSX280 are an 8-bit quantity, and are always negative (sign bit is set.) The system include file ERRORS.INC defines symbolic names for the errors that may be generated by the system calls. This document refers to those names in the system call descriptions chapter.

1.3. Task types

RSX180/280 tasks can be either *privileged* or *non-privileged*.

- **Privileged tasks** can access kernel data structures directly and call kernel routines, and therefore can act as a kernel extension. Privileged tasks can also issue logical block I/O to a mounted volume and have unrestricted access to the file system. MCR and many other system utilities are privileged.
- **Non-privileged tasks** cannot access directly the kernel database and cannot make use of certain system calls. Most user-written tasks fall into this category.

1.4. Task states

The system recognizes the existence of a task only after it has been successfully installed and has an entry in the System Task Directory (STD). Once a task has been installed, it is either *dormant* or *active*:

- **Dormant:** immediately following the processing of an INStall command by the Monitor Console Routine (MCR), a task is known to the system, but is dormant. A dormant task has an entry in the STD, but no request has been made to activate it.
- **Active:** a task is active from the time it is requested until the time it exits. Requesting a task means issuing the .RQST, .RUN, or .RPOI system call, or an MCR RUN command. An active task is eligible for scheduling, whereas a dormant task is not.

  The active state has three sub-states:

  - **Ready-to-run:** a ready-to-run the task competes with other tasks for CPU time on the basis of priority. The highest priority ready-to-run task obtains CPU time and thus becomes the current task.
  - **Suspended:** a suspended task is unable to compete for CPU time for synchronization reasons or because a needed resource is not available. Task priority effectively remains unchanged, allowing the task to compete for memory space.
  - **Stopped:** a stopped task is unable to compete for CPU time because of pending I/O completion, event flag(s) not set, or because the task stopped itself. When stopped, a task's priority effectively drops to zero and the task can be checkpointed by any other task, regardless of that task's priority. If an AST occurs for the stopped task, its normal task priority is restored only for the duration of the AST routine execution; once the AST is completed, task priority returns to zero.

1.5. Event flags

Event flags are a means to coordinate and synchronize task operation. Event flags can be set in response to specific events, such as when an I/O transfer culminates or a time interval elapses.

Under RSX180 and RSX280, 32 event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique Event Flag Number (EFN) unique to the task and are set or cleared as a result of that task's operation.

Tasks can set, clear and test individual event flags; task execution can also be suspended or stopped until an event flag is set.

1.6. Logical Unit Numbers

A Logical Unit Number (LUN) is a number associated with a physical device unit. The LUN provides an efficient mapping to the physical device unit, eliminating the necessity to search device tables during I/O operations.

The association between the LUN and the physical device unit is specific for each task, and can be determined at task build time (static assignment), changed by the operator via the MCR REAssign command, or by the task itself dynamically via an Assign LUN (.ALUN) system call.

The current version of RSX180/280 limits the umber of LUNs to 16.

1.7. System Traps

System traps are transfers of control (also called software interrupts) that provide tasks with a means of monitoring and reacting to events. The kernel initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine.

There are two kinds of system traps:

- **Asynchronous System Traps** (ASTs) detect events that occur asynchronously to the task's execution. That is, the task has no direct central over the precise time that the event (and therefore the trap) may occur. The completion of an I/O transfer may cause an AST to occur, for example.
- **Synchronous System Traps** (SSTs) detect events directly associated with the execution of program instructions. They are synchronous because they always recur at the same point in the program when trap-causing instructions occur. For example, an illegal instruction causes an SST.

A task that uses the system trap facility issues system directives that establish entry points for user-written service routines. SST entry points are specified in a single table, while AST entry points are set individually

for each kind of AST. When a trap condition occurs, the task automatically enters the appropriate routine, if its entry point has been specified.

1.7.1. Asynchronous System Traps

ASTs can be used to inform the task that a certain event has occurred, such as the completion of an I/O operation. As soon as the task has serviced the event, it can return to the interrupted code.

The kernel pushes on the task's stack the contents of the AF and PC CPU registers before transferring the execution to the AST routine, emulating the following instruction sequence:

```
push  af
call  AST
pop   af
```

The AST service routine must save and restore all other CPU registers used. If the registers are not restored after an AST has occurred, the task's subsequent execution may be unpredictable.

Some directives can specify both an event flag and an AST; with these directives, ASTs can be used as an alternative to event flags or the two can be used together. Therefore, you can specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can determine the action required.

Although not able to distinguish execution of an SST routine from task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine, but not by another AST routine.

The following notes describe general characteristics and uses of ASTs:

- If an AST occurs while the related task is executing, the task is interrupted so that the AST service routine can be executed.
- If an AST occurs while another AST is being processed, the kernel queues the latest AST (First-In First-Out, FIFO). When the current AST service is complete, the task then processes the next AST in the queue, unless AST recognition was disabled by the AST service routine.
- If an AST occurs while the related task is suspended or stopped, the task remains suspended or stopped after the AST routine has been executed, unless it is explicitly resumed or unstopped either by the AST service routine itself, or by another task (the MCR RESUME or UNSTOP command, for example).
- If an AST occurs while the related task is waiting (or stopped) for an event flag to be set (a Wait For or Stop For directive), the task continues to wait after execution of the AST service routine unless the event flag is set upon AST exit.

- If an AST occurs for a checkpointed task, the kernel queues the AST (FIFO), brings the task into memory, and then activates the AST.
- A task can disable ASTs via the AST Control directive (.ASTCT) during critical sections of code (a critical section might be one that accesses data bases also accessed by AST service routines, for example.) and re-enable them afterwards. If ASTs occur while AST recognition is disabled, they are queued (FIFO) and then processed when AST recognition is enabled again.

1.7.2. AST Service Routines

When an AST occurs, the kernel pushes the CPU AF and PC registers onto the task's stack and saves the task's status and Wait For mask into the Task Context structure. The new Wait Mask is cleared so that the AST service routine has access to all the available system services. Depending on the reason for the AST, the stack may also contain additional parameters. Note that no other CPU registers are saved; if the routine makes use of them, it must save and restore them itself.

After processing an AST, the task must remove the trap-dependent parameters from its stack; that is, everything from the top of the stack down to, but not including, the task's PC. It must then issue an AST Service Exit directive with the stack set as indicated in the description of that directive (see .ASTX). When the AST service routine exits, it returns control to one of two places: another AST, or the original task.

RSX180 and RSX280 support the following AST types:

Symbolic name	Description
AST.IO	Completion of an I/O request
AST.UC	Unsolicited input from terminal
AST.RD	Data received
AST.CK	Clock event (mark time)
AST.ST	Connected task emitted status
AST.AB	Abort request
AST.CL	Command arrival for CLI

The AST types are described below:

- AST.IO: when a task queues an I/O request and specifies an AST service entry point, an AST will occur upon completion of the I/O request. The task's stack will contain the following values:

Stack offset	Length (bytes)	Contents
SP+4	2	Saved AF
SP+2	2	Saved PC (return address)
SP+0	2	Address of I/O status block

- AST.RD: if a task needs to be notified when it receives a message, it uses the AST Define (.ASTDF) system call to associates an AST routine to a Receive Data event. An AST will occur when the message is sent to the task. The stack will contain the following values:

Stack offset	Length (bytes)	Contents
SP+2	2	Saved AF
SP+0	2	Saved PC (return address)

- AST.CK: when a task issues a Mark Time directive and specifies an appropriate AST service entry point, an AST will occur when the indicated time interval has elapsed. The task's stack will contain the following values:

Stack offset	Length (bytes)	Contents
SP+4	2	Saved AF
SP+2	2	Saved PC (return address)
SP+0	2	Event flag number, or zero if none was specified

- AST.ST: if a task needs to be notified when an offspring task, connected by a Spawn, Connect, or Send, Request And Connect directive, exits or emits status, it uses the AST Define (.ASTDF) system call to associate an AST routine to an Emit Status event. An AST will occur when the offspring task exits or emit status to the parent or connected tasks. The parent task's stack contains the following values:

Stack offset	Length (bytes)	Contents
SP+4	2	Saved AF
SP+2	2	Saved PC (return address)
SP+0	2	Exit Status Block (ESB) address

- AST.AB: if a task becomes aborted via directive or MCR when the Specify Requested Exit AST (SREAs) is in effect, the abort AST is entered with the task's stack containing the following values:

Stack offset	Length (bytes)	Contents
SP+4	2	Saved AF
SP+2	2	Saved PC (return address)
SP+0	2	Trap-dependent parameter

- AST.UC: if a task issues an I/O IO.ATA function to the terminal driver, unsolicited terminal input will cause entry into the AST service routine. Upon entry into the routine, the task's stack contains the following values:

Stack offset	Length (bytes)	Contents
SP+4	2	Saved AF
SP+2	2	Saved PC (return address)
SP+1	1	Terminal unit number
SP+0	1	Unsolicited character

- AST.CL: if a command arrives for a CLI, the Command Arrival AST is entered. The stack contains:

Stack offset	Length (bytes)	Contents
SP+2	2	Saved AF
SP+0	2	Saved PC (return address)

1.7.3. Synchronous System Traps

SSTs are supported only by RSX280[5]. They allow a task to react to Z280 traps, such as:

- Privileged instructions
- Extended instructions
- Invalid memory access
- Division exception
- Breakpoints

The user can set up an SST vector table, containing one entry per SST type. Each entry is the address of an SST routine that services a particular type of SST (a routine that services privileged instructions, for example). When an SST occurs, the kernel transfers control to the routine for that type of SST. If a

5 Future RSX180 versions may support SSTs for the single, Illegal Instruction Trap of the Z180 CPU.

corresponding routine is not specified in the table, the task is aborted. The SST routine enables the user to process the failure and then return to the interrupted code.

SST routines must always be reentrant if there is a possibility that an SST can occur within the SST routine itself. Although the kernel initiates SSTs, the execution of the related service routines is indistinguishable from the task's normal execution. An AST or another SST can therefore interrupt an SST routine.

1.7.4. SST service routines

The kernel initiates SST service routines by pushing the task's PC and trap-specific parameters onto the task's stack. After removing the trap-specific parameters, the service routine returns control to the task by issuing a Return (RET) instruction. Note that no other CPU registers are saved; if the SST routine makes use of them, it must save and restore them itself.

To the system, SST routine execution is indistinguishable from normal task execution, so that all directive services are available to an SST routine. An SST routine can remove the interrupted PC from the stack and transfer control anywhere in the task; the routine does not have to return control to the point of interruption. Note that any operations performed by the routine (such as the modification of registers or the setting or clearing of event flags) remain in effect when the routine eventually returns control to the task.

A trap vector table within the task contains all the service routine entry points. You can specify the SST vector table by means of the Specify SST Vector Table For Task directive. The trap vector table has the following format:

Offset	Symbolic name	Trap
0	SST.EI	Extended instruction (EPU-reg)
2	SST.E2	Extended instruction (EPU-mem)
4	SST.PR	Privileged instruction
6	SST.SC	Unrecognized system call
8	SST.AC	Access violation
10	SST.DV	Division exception
12	SST.SS	Single-step
14	SST.BP	Breakpoint-on-halt

A zero address in the table means that no entry point is specified for the particular trap. If an SST occurs and an associated entry point is not specified in the table, the kernel aborts the task.

Depending on the reason for the SST, the task's stack may also contain additional information, as follows:

- SST.EI (Extended instruction, EPU-register):

Stack offset	Length (bytes)	Contents
SP+2	2	Saved PC (address of the next instruction)
SP+0	2	Address of the extended instruction template

- SST.E2 (Extended instruction, EPU-memory):

Stack offset	Length (bytes)	Contents
SP+4	2	Saved PC (address of the next instruction)
SP+2	2	Address of the memory operand
SP+0	2	Address of the extended instruction template

- SST.PR (privileged instruction):

Stack offset	Length (bytes)	Contents
SP+0	2	Saved PC (address of the instruction causing the trap)

- SST.SC (unrecognized system call):

Stack offset	Length (bytes)	Contents
SP+2	2	Saved PC (address of the next instruction)
SP+0	2	System call instruction operand

- SST.AC (access violation):

Stack offset	Length (bytes)	Contents
SP+0	2	Saved PC (address of the instruction causing the trap)

- SST.DV (divide exception):

Stack offset	Length (bytes)	Contents
SP+0	2	Saved PC (address of divide instruction)

- SST.SS (single-step):

Stack offset	Length (bytes)	Contents
SP+0	2	Saved PC (address of the next instruction)

- SST.BP (breakpoint-on-halt):

Stack offset	Length (bytes)	Contents
SP+0	2	Saved PC (address of the halt instruction)

Note that the contents of the Z280 Master Status Register (MSR) is not pushed on the stack. All items except the PC must be removed from the stack before the SST service routine exits.

Consult Zilog's Z280 Technical Manual for more information about the Z280 exceptions and traps.

2. System call descriptions

The SYSFN.INC system include file defines symbolic names for the RSX180/280 system calls. Those names are used in the descriptions below.

2.1. By function

Task execution:

System call	Description	RSX-11M equivalent
.ABORT	Abort Task	ABRT$
.CSRQ	Cancel Scheduled Request	CSRQ$
.EXIT	Exit Task	EXIT$, EXST$
.EXTSK	Extend Task	EXTK$
.RQST	Request Task	RQST$, SDRC$, SPWN$
.RESUM	Resume Task	RSUM$, USTP$
.RUN	Schedule Task Run	RUN$
.STOP	Stop Task	SPND$, STOP$

Task status control:

System call	Description	RSX-11M equivalent
.ALTP	Alter Task Priority	ALTP$

Informational:

System call	Description	RSX-11M equivalent
.GCII	Get CLI information	GCII$
.GDAT	Get current Date and Time	GTIM$
.GDIR	Get Current Directory	GDIR$
.GIN	Get General Information	GIN$
.GTLUN	Get LUN information	GLUN$
.GTPAR	Get Partition information	GPRT$
.GTSK	Get Task information	GTSK$

Event-associated:

System call	Description	RSX-11M equivalent
.CLEF	Clear Event Flag	CLEF$
.CMKT	Cancel Mark-Time request	CMKT$
.EXIF	Exit Task if Event Flag not set	EXIF$
.MRKT	Mark-Time request	MRKT$
.RDEF	Read Event Flag	RDEF$
.SETF	Set Event Flag	SETF$
.STLO	Stop for Logical OR of Event Flags	STLO$
.STSE	Stop for Single Event Flag	STSE$
.WTLO	Wait for Logical OR of Event Flags	WTLO$
.WTSE	Wait for Single Event Flag	WTSE$

Trap-associated:

System call	Description	RSX-11M equivalent
.ASTCT	AST Control	DSAR$, ENAR$, ISAR$
.ASTDF	Define AST routine	SCAA$, SRDA$, SREX$
.ASTX	Exit AST routine	ASTX$
.SVTBL	Specify SST table	SVTK$

I/O and inter-task communications:

System call	Description	RSX-11M equivalent
.ALUN	Assign LUN	ALUN$
.GLUN	Get LUN information	GLUN$
.GTCMD	Get MCR Command Line	GMCR$
.QIO	Queue I/O operation	QIO$, QIOW$
.RCVD	Receive Data	RCVD$, VRCD$
.SEND	Send Data	SDAT$, VSDA$

Parent/offspring tasking:

System call	Description	RSX-11M equivalent
.CONN	Connect to Task	CNCT$
.EMTST	Emit Status	EMST$
.EXIT	Exit Task	EXIT$, EXST$
.RPOI	Request Task and Pass Offspring Information	RPOI$, SDRP$
.RQST	Request Task	RQST$, SDRC$

Kernel database access control:

System call	Description	RSX-11M equivalent
.SUPER	Enter/Leave System mode	-

2.2. In alphabetical order

.ABORT

Abort Task.

Input	
HL	Pointer to 6-character task name, or zero to abort the current task.
Returns	
AF	Error/success code.

Terminates (aborts) the execution of the indicated task. The function is intended for use as an emergency or fault exit, and displays a termination notification at one of the following terminals:

- The terminal from which the task was requested.
- The originating terminal of the task that requested the aborted task.
- The operator’s console (CO:) if the task was started internally form another task by a .RUN system call, or by an MCR RUN command that specified one or more time parameters.

A task must be privileged to abort a task running at another terminal.

When a task is aborted, the kernel frees all the task's resources. In particular, the kernel:

- Detaches all attached devices.
- Closes all open files (files open for write access are locked).
- Runs down the task's I/O.
- Flushes the AST queue and despecifies all specified ASTs.
- Flushes the receive queue.
- Flushes the clock queue for outstanding Mark Time requests for the task.
- Flushes all outstanding CLI command buffers for the task.
- Returns a severe error status (EX.SEV) to the parent task if the task is connected.
- Breaks the connection with any offspring tasks.
- Frees the task's memory if the aborted task was not fixed.

If the aborted task had a Requested Abort AST specified, the task will receive that AST instead of being aborted. No indication that this has occurred is returned to the task that issued the abort request.

Errors:

E.TNF	task not found (not installed)
E.TNAC	task not active
E.TABO	task is already being aborted
E.PRIV	privilege violation
E.INV	invalid value (address of task name outside task limits)
E.BADOP	invalid operation (attempt to abort a system-protected task)

.ALTPR

Alter Task Priority.

Input	
HL	Pointer to 6-character task name, or zero to change the priority of the issuing task.
E	New priority in the range 1..255, or zero to reset the priority to its default value.
D	Which priority to set: zero = running, non-zero = installed (default) priority.
Returns	
AF	Error/success code.

Changes the running or the installed priority of the specified active task to either a new priority, or the task's default (installed) priority. The specified task must be installed and active.

Notes:

- The kernel resets the task's priority to its installed priority when the task exits.
- A non-privileged task can change only its own running priority, and only to a value lower then or equal to its installed priority.
- A privileged task can change the running and installed priority of any task, including its own, to any value.

The kernel reorders any outstanding I/O requests for the task in the I/O queue and reallocates the task's partition. The partition reallocation may cause the task to be checkpointed.

Errors:

E.TNF	task not found (not installed)
E.PRIV	privilege violation
E.INV	invalid value (address of task name outside task limits)

.ALUN

Assign LUN.

Input	
DE	Two-character device name (first char in E, second char in D).
C	Device's unit number.
B	Logical Unit Number (LUN).
Returns	
AF	Error/success code.

Assigns a physical device unit to a Logical Unit Number (LUN) of the issuing task. It does not indicate that the task has attached itself to the device.

The actual physical device assigned to the logical unit is dependent on the logical assignment table (see the ASN command in the MCR Reference Manual). The system first searches the logical assignment table for a device name match. If a match is found, the system assigns the physical device unit associated with the matching entry to the logical unit. Otherwise, the physical device table is searched and, if a match is found, the actual physical device unit is assigned to the logical unit number.

When a task reassigns a LUN from one device to another, the kernel cancels all I/O requests for the issuing task in the previous device queue.

Errors:

E.CHOPN	channel is open (a file is open on the specified LUN)
E.NODEV	no such device
E.BADOP	invalid operation (either the specified LUN value is outside the 1..16 range, or an attempt was made to reassign the LUN assigned to a device attached to the task)

.ASTCT

AST Control.

Input	
C	Zero disables, non-zero enables AST recognition.
Returns	
AF	Success code.

Enables or disables recognition of Asynchronous System Traps (AST).

Notes:

- Disabling ASTs does not despecify them. The ASTs are still queued FIFO as they occur and are dispatched in that order when the task re-enables AST recognition.
- AST recognition is automatically disabled when an AST service is executing, and re-enabled when the AST routine exits via the .ASTX system call.
- AST recognition is initially enabled when the task's execution is started.

Errors:

None; the system call always succeeds with an E.OK return code.

.ASTDF

Define AST service routine.

Input	
DE	Address of the AST routine, or zero to disable AST processing for the specified type.
C	AST type.
Returns	
AF	Error/success code.

Declares (specifies) an AST routine to be called when the corresponding event occurs. If the routine address is zero, the AST for the specified type is effectively disabled (despecified).

Only the following AST types are allowed to be specified by this call:

- AST.RD (data received, see .RECV)
- AST.ST (offspring task emitted status, see .CONN, .EMTST, .EXIT)[6]
- AST.AB (abort request)
- AST.CL (command line for CLI)

For the remaining types, individual AST routines are specified as parameter to the associated system call:

- AST.IO (I/O transfer completion), see .QIO
- AST.CK (mark time), see .MRKT
- AST.UC (unsolicited character from attached terminal), specified as parameter to the IO.ATA QIO function.

See Section 1.7.2 for a description of the stack contents when ASTs routines are called.

Errors:

E.INV	invalid AST type
E.BADOP	a non-privileged task attempted to re-specify of despecify and Abort Request AST (AST.AB) after one had already occurred
E.NOMEM	not enough system pool memory to perform the operation

6 This is different than in RSX-11M, where the *status notification* AST routine address is specified as parameter to the Spawn, Connect, or Request directives.

.ASTX

Exit AST service routine.

Input	
None.	
Returns	
AF	Error code.

Terminates the execution of an AST service routine.

Notes:

- The task must be in AST state, else the directive will be rejected.
- The task must restore any modified registers (except AF, which is saved/restored by the kernel) before returning from an AST (see Section 1.7.2).
- Any AST-specific parameters must be removed from the task's stack before issuing the AST exit directive.
- If another AST is queued and ASTs are not disabled, then the AST will be immediately executed. Otherwise, the task pre-AST state is restored.
- The task can alter its return address by manipulating the information on its stack prior to executing an AST Exit (.ASTX) directive. For example, to return to task state at an address other than the one pushed on the stack, the task can simply replace the PC word on the stack. This may be useful in those cases in which error conditions are discovered in the AST routine; but you should use extreme caution when doing this alteration since AST service routine bugs are difficult to isolate.

Errors:

E.BADOP invalid operation (task not in AST state)

.CLEF

Clear Event Flag.

Input	
E	Event flag number.
Returns	
AF	Previous event flag state on success, else an error code.

Clear the indicated event flag. On success, the function returns in the CPU accumulator the previous flag state: zero if the event flag was unset, non-zero if was set.

Errors:

E.BADFL invalid Event Flag Number (outside 1..32 range.)

.CMKT

Cancel Mark Time request.

Input	
BC	AST routine address, or zero for any
E	Event Flag number, or zero for any
Returns	
AF	Success code.

Cancels a specific Mark Time request (see .MRKT) or all Mark Time requests issued by the current task.

Notes:

- If neither the EFN nor AST parameters are specified (both are zero), all Mark Time Requests issued by the task are canceled.
- If both EFN and AST parameters are specified (nonzero), only Mark Time Requests issued by the task specifying either that event flag or AST address are canceled.
- If only one EFN or AST parameter is specified (nonzero), only Mark Time Requests issued by the task specifying the event flag or AST address are canceled.

Errors:

None; the system call always succeeds with an E.OK return code.

.CONN

Connect to Task.

Input	
HL	Pointer to 6-character task name.
D	Status format (0=short, 1=long)
E	Number of Event Flag to set when the offspring task emits status.
BC	Address of the Exit Status Block (ESB) to receive status information.
Returns	
AF	Error/success code.

Synchronizes the task issuing the directive with the exit or emit status of another task (offspring) that is already active.

When this system call is issued, the system queues an Offspring Control Block (OCB) to the offspring task and clears the specified event flag. When the offspring exits or emits status, the kernel sets the event flag and stores the status information into the Exit Status Block (ESB) of all connected tasks:

- If the short status format is specified by the connected task, only the offspring status word (2 bytes) will be stored at its ESB address.
- If the long status format is specified, an additional 14-byte of status information will be stored at the ESB address (16 bytes in total):

Offset	Size	Value
0	2 bytes	Offspring status word
2	1 byte	Exit cause (TKTN code, always zero for kernel version 6.37)
3	3 bytes	Offspring's terminal name and unit
6	6 byte	Offspring task name
12	1 byte	Offspring task attributes (T.ATTR)
13	2 bytes	Offspring task status (T.ST)
15	1 byte	Reserved

In addition, if the task specified Exit Status ASTs (AST.ST), the exit AST routine is called with the address of the associated Exit Status Block on the stack.

Notes:

- This directive should not be issued to connect to Command Language Interpreter (CLI) tasks; it is illegal to connect to a CLI task.

Errors:

E.BADOP	attempt to connect to a CLI task
E.TNF	offspring task not in system
E.TNAC	offspring task is not active
E.NOMEM	not enough system memory (could not allocate an Offspring Control Block)
E.BADFL	invalid event flag number
E.INV	address of task name string and/or ESB outside issuing task address limits

.CSRQ

Cancel Scheduled Task request.

Input	
HL	Pointer to 6-character task name.
Returns	
AF	Error/success code.

Cancels all time-synchronized initiation requests for a specified task, regardless of the source of each request. These requests result from a .RUN system call, or from any of the time-synchronized variations of the MCR RUN command.

Notes:

- A non-privileged task can cancel time-based initiation requests only for a task with the same terminal device (TI:).

Errors:

E.INV	invalid value (address of task name string outside issuing task limits)
E.TNF	task not installed
E.PRIV	privilege violation

.EMTST

Emit Status.

Input	
HL	Status code.
Returns	
AF	Error/success code.

Returns the specified 16-bit quantity to all connected tasks. It also sets the corresponding event flags and possibly declares any ASTs the connected tasks may have specified. Whenever status is emitted to one or more tasks, those tasks no longer remain connected to the task issuing the Emit Status directive.

The ERRORS.INC system include file defines symbolic names for the standard status values that can be returned to parent tasks:

Symbolic name	Value	Description
EX.WRN	0	Warning – task succeeded, but irregularities are possible
EX.SUC	1	Success – results should be as expected
EX.ERR	2	Error – results are unlikely to be as expected
EX.SEV	4	Severe error – one or more fatal errors detected, or task aborted

The above status values should be returned in the low byte of the 16-byte status word; the high byte may be used to pass additional task-specific information to the connected tasks.

Errors:

None; the system call always succeeds with an E.OK return code.

.EXIF

Exit Task if Event Flag is not set.

Input	
HL	Exit code.
E	Event flag number.
Returns	
AF	Error/success code.

Terminates the execution of the issuing task if, and only if, an indicated event flag is not set. The system call returns control to the issuing task if the specified event flag is set.

See also the notes for the .EXIT directive.

Errors:

E.BADFL invalid Event Flag Number (outside 1..32 range.)

.EXIT

Exit Task and Emit Status.

Input	
HL	Exit code.
Returns	
-	

Terminates the execution of the issuing task, passing a 16-bit status back to all tasks connected (by the Connect, or Request directives).

On Exit, the kernel frees task resources. In particular, the kernel:

- Detaches all attached devices.
- Flushes the AST queue and despecifies all specified ASTs.
- Flushes the receive queue.
- Flushes the clock queue for any outstanding Mark Time requests for the task.
- Closes all open files (files open for write access are locked).
- Runs down the task's I/O.
- Flushes all outstanding CLI command buffers for the task.
- Breaks the connection with any offspring tasks.
- Returns the specified code to any parent task.
- Frees the task's memory if the exiting task was not fixed.

Notes:

- If an offspring task aborts for any reason, a status of EX.SEV is returned to the parent task.

Errors:

None, the function always exits (and thus never returns to) the issuing task.

.EXTSK

Extend Task.

Input	
HL	Positive or negative increment in 16-byte units, or zero to restore installed size.
Returns	
AF	Error/success code.

Instructs the system to modify the size of the issuing task by a positive or negative increment of 16-byte blocks. If the directive specifies an increment value of zero, the kernel makes the issuing task's size equal to its installed size.

The issuing task must be running in a system-controlled partition and cannot have any outstanding I/O when it issues the directive. The task must also be checkpointable to increase its size; if necessary, the kernel checkpoints the task, and then returns the task to memory with its size modified as directed. If the issuing task is checkpointable, a positive increment may require dynamic memory and extra space in a checkpoint file sufficient to contain the task.

Errors:

E.NOMEM	attempt to extend the task beyond the maximum size (60K for RSX180, 64K for RSX280), or beyond the size of the main partition or the user-controlled partition, or not enough space is left in the checkpoint file, or checkpointing support is not included in the system, or checkpointing is disabled.
E.BADOP	attempt to shrink the task below its default (installed) size
E.INV	attempt to set the task size to zero or to a negative value

.GCII

Get Command Interpreter information.

Input	
HL	Address of a 12-byte parameter structure.
Returns	
AF	Error/success code.

The parameter structure contains the following fields:

Offset	Size	Value
0	2 bytes	Address of buffer to receive CLI information, at least 10 bytes long.
2	1 byte	Size of the above buffer.
3	6 bytes	6-character CLI name (shorter names must be padded with blanks), or null (a zero on the first byte is enough) to lookup by device name.
9	3 bytes	2-character terminal device name followed by 1-byte unit number, or null to lookup by CLI name.

The CLICB.INC system include file defines a handy GCII$ macro that can be used to generate the structure:

```
GCII$  buf,len,cli,dev,unit
```

All parameters are optional, but either a CLI name (*cli*) or a terminal device (*dev*) must be specified. If *buf* is not specified, the macro will reserve a 32-byte local buffer.

On success, the function will return a copy of the CLI Control Block (CLICB) kernel structure, as described in the CLICB.INC system include file. If the destination buffer size is not large enough to contain all the CLI data, the information will be silently truncated. A task can check for this condition by inspecting the CL.DPL and CL.CPL fields of the returned structure, which contain the length of the CLI prompt strings and, if necessary, re-issue the call specifying a larger buffer.

Notes:

- The issuing task must be privileged to get information about a CLI for any terminal other than its own terminal, or about a CLI to which its terminal is not set.
- If both a terminal and a CLI name are specified in the request, information will be returned for the CLI matching the specified name.

Errors:

E.INV	invalid value (address of parameter structure
E.CLNF	CLI not in system
E.NODEV	no such device
E.PRIV	privilege violation (see notes above)

.GDAT

Get current date and time.

Input	
HL	Address of an 8-byte structure to store the current system date and time.
Returns	
AF	Error/success code.

The date/time structure contains the following fields:

Offset	Size	Value
0	2 bytes	Year, as four packed BCD digits.
2	1 byte	Month, as two packed BCD digits in the range 1..12
3	1 byte	Day of month, as two packed BCD digits in the range 1..31
4	1 byte	Hour, as two packed BCD digits in the range 0..23
5	1 byte	Minutes, as two packed BCD digits in the range 0..59
6	1 byte	Seconds, as two packed BCD digits in the range 0..59
7	1 byte	Day of week in the range 1..7 (1=Sunday, 2=Monday, etc.)

Errors:

E.INV invalid value (structure address outside issuing task limits)

.GDIR

Get current directory name.

Input	
HL	Address of a 9-byte buffer to store the current directory name.
C	Function code.
Returns	
AF	Error/success code.

The function code specifies which directory name to return:

GD.TSK	Get task's current directory
GD.TI	Get terminal's current directory
GD.LOG	Get terminal's login directory

Errors:

E.INV	invalid function code, or buffer address outside task limits
E.SSNF	user session not found (user not logged in, or task not associated to a user session)

.GIN

Get General Information.

Input	
HL	Address of buffer to receive the requested information.
C	Function code.
Returns	
AF	Error/success code.

The function code indicates which information to return:

I.FEAT Return the features bits as a 16-bit word

I.HOST Return the Host Name as 9-character, blank-padded string

I.SMEM Return system memory information, 3 words:

- total system memory in kbytes
- total memory taken by the kernel and kernel database in kbytes
- current available memory in kbytes

I.TCKS Return the number of Ticks per Second as a 16-bit word.

I.TSCB Return Login information (Session Control Block) for the specified terminal.

I.UPTM Return the system up time in a 6-byte array:

- ticks
- seconds
- minutes
- hours
- 16-byte days

I.USCB Return Login information (Session Control Block) for the current terminal.

I.VERS Return System Version Number and Type, 2 words:

- version number (lo-byte = minor, hi-byte = major)
- system type (1 = RSX180, 2 = RSX280)

Errors:

E.INV invalid function code, or buffer address outside task limits
E.NODEV no such device (I.TSCB)
E.SSNF user session not found (I.TSCB, I.USCB)

.GTCMD

Get Command Line.

Input	
HL	Address of buffer in user space to receive the command line.
E	Size of the buffer in bytes, including 1-byte returned length field.
Returns	
AF	Error/success code.

On success, the function returns the command length in the first byte of the destination buffer, followed by the command line chars. The terminating character (Carriage Return, Escape or Control-Z) is also stored in the buffer and accounted for the in the returned command length.

Notes:

- The task's invocation command line is stored in kernel memory, and therefore a task must issue this system call as early as possible during execution to read the command line and thus to free the corresponding system resources.
- If no command line is available (either none was specified by the CLI, or the task was not started by a CLI, or if the command line was already read), the directive returns nevertheless success (E.OK), but sets the returned command length to zero.

Errors:

E.INV destination buffer address outside task limits

.GTLUN

Get LUN information.

Input	
HL	Address of a 6-byte buffer to receive LUN information.
C	Logical Unit Number (LUN).
Returns	
AF	Error/success code.

This system call returns information about a physical device unit to which a LUN is assigned. If requests to the physical device unit have been redirected to another unit, the information returned will describe the effective assignment.

On success, the system stores in the buffer the following information:

Offset	Size	Value
0	2 bytes	2-character Device Name
2	1 byte	Unit number, as binary value
3	1 byte	Status byte (U.ST)
4	2 bytes	Characteristics word (U.CW)

Errors:

E.LUN	invalid LUN (outside 1..16 range or not assigned)
E.INV	destination buffer address outside task limits

.GTPAR

Get Partition information.

Input	
HL	Pointer to 6-character partition name, or zero for the task's running partition.
DE	Address of a buffer to receive the partition information.
Returns	
AF	Error/success code.

On success, the buffer will contain a copy of the Partition Control Block (PCB) for the specified partition.

The buffer must be large enough to contain all the information. The PCB.INC system include file describes the structure of the PCB and sets the PCBSZ symbol to the PCB size.

Errors:

E.INV	invalid value (address of partition name string or destination buffer outside task limits)
E.PNF	partition not found

.GTSK

Get Task information.

Input	
HL	Pointer to 6-character task name, or zero for the current task.
DE	Address of a buffer to receive task information.
Returns	
AF	Error/success code.

On success, the buffer will contain a subset of the information contained in the Task Control Block (TCB) kernel structure for the specified task.

The buffer must be large enough to contain all the information. The TCB.INC system include file describes the returned structure fields and sets the GTKSZ symbol to the structure size.

Errors:

E.INV	invalid value (task name string or destination buffer outside issuing task limits)
E.TNF	task not in system

.MRKT

Mark Time.

Input	
HL	Magnitude of time interval.
D	Units (1=ticks, 2=seconds, 3=minutes, 4=hours)
E	Optional event flag number.
BC	Optional AST routine address.
Returns	
AF	Error/success code.

Instructs the system to set an event flag and/or call an AST routine after the specified time interval elapses.

If an event flag is specified (non-zero), the flag is cleared when the directive is issued, and set when the time interval elapses.

If an AST routine address is specified (non-zero), the AST takes place upon expiration of the time interval, and after setting the event flag, if one was specified.

Errors:

E.BADFL	invalid event flag number (outside the 1..32 range)
E.INV	invalid time units, interval magnitude overflow
E.NOMEM	not enough kernel memory (pool) to perform the operation

.QIO

Queue I/O operation.

Input	
HL	Pointer to the QIO structure.
Returns	
AF	Error/success code.

Places an I/O request into a queue of priority-ordered requests for the device unit assigned to the specified LUN (see the .ALUN system call). The QIO structure contains the following fields:

Offset	Size	Symbolic name	Value
0	1 byte	Q.FUNC	Function code
1	1 byte	Q.SUBF	Sub-function code
2	1 byte	Q.LUN	Logical Unit Number
3	1 byte	Q.EFN	Event flag number
4	1 byte	Q.WAIT	Wait option
5	2 bytes	Q.IOSB	Address of a 4-byte I/O Status Block
7	2 bytes	Q.AST	AST routine address
9	0..12 bytes	Q.P1..Q.P6	Function-dependent parameter list

If an event flag is specified (non-zero), the kernel clears the flag when the request is queued, and sets the flag when the I/O transfer completes.

If the wait parameter is non-zero, the system issues an implicit Wait For Event Flag call, suspending the execution of the task until the I/O transfer completes. This implies that a valid event flag number must be specified when the wait option is set.

If an AST routine address is specified (non-zero), the AST takes place upon I/O completion, with the address of the I/O Status Block (IOSB) on the stack..

The first word of the IOSB is set to E.PEND and the second word cleared when the request is queued, and set to the final I/O status when the I/O transfer completes.

The QIO.INC system include file contains two macros, QIO$ and QIOW$, that can be used to generate the QIO structure at assembly time:

```
QIO$  func,lun,[efn],[wait],iosb,[ast][,plist]

QIOW$ func,lun,[efn],iosb,[ast][,plist]
```

The QIOW$ macro is the same as QIO$, but sets the *wait* parameter to 1. The various parameters must be valid expressions accepted by assembler data-generating directives such as DB and DW; square brackets ([ ]) denote optional or function-dependent parameters. Note that the place-holding comma must be retained for any missing parameter.

The *func* parameter is specified as a word value, with the function code (Q.FUNC) in the low byte and the sub-function code (Q.SUBF) in the high byte.

The *plist* parameter list, when required, has the form <*p1*,*p2*,*p3*,*p4*,*p5*,*p6*> and must be enclosed in angle brackets (<>). For convenience, any comma in the parameter list may be omitted if no parameters appear to the right of it; if a parameter appears to the right of any place-holding comma, that comma must be retained.

Notes:

- If the directive is rejected, the specified event flag is not guaranteed to be cleared or set.
- Since an I/O operation results in data transfer to/from the task address space, a task with I/O outstanding cannot be checkpointed. The kernel waits until a task has no outstanding I/O before initiating the checkpoint operation.

Errors:

E.NOMEM	not enough kernel memory (pool) to perform the operation
E.INV	invalid value (QIO structure or IOSB outside task limits)
E.BADFL	invalid flag number (non-zero outside 1..32 range)
E.BADOP	invalid LUN, or not assigned; IO.LOV to device other than task's load device
E.EOF	IO.LOV tried to read past end of task image file
E.PERM	logical block I/O to mounted device from non-privileged task
E.OFL	device unit is offline
E.DEVNM	virtual block I/O to dismounted device
E.TNF	ACP task not in system or not active

.RDEF

Read single Event Flag.

Input	
E	Event flag number.
Returns	
AF	Flag status on success (0=clear, non-zero=set), else error code.

Tests the indicated event flag and returns its polarity.

Errors:

E.BADFL invalid event flag number (outside 1..32 range)

.RECV

Receive Data.

Input	
DE	Address of buffer to received data.
BC	Buffer size in bytes, including 2-byte returned length field.
Returns	
AF	Error/success code.

Dequeues a data block for the issuing task that was queued (FIFO) for the task by a Send Data (.SEND) directive. The returned buffer contains the following information:

Offset	Size	Value
0	2 bytes	Data length in bytes (*n*)
2	*n* bytes	Actual data

If the specified buffer is smaller than the dequeued data block, the extra data will be silently truncated.

Errors:

E.QEMP	receive queue empty
E.BADOP	destination buffer size smaller than 2 bytes
E.INV	destination buffer outside task limits

.RESUM

Resume a Stopped Task.

Input	
HL	Pointer to 6-character task name, or zero for the current task.
Returns	
AF	Error/success code.

Resume the execution of a task that issued a .STOP directive, or that was stopped by a MCR STP command. The directive does not resume execution of tasks stopped for event flags.

Stopped tasks can still receive receive ASTs, and thus a stopped task can issue this directive from an AST to unstop itself. The task will then return to the ready-to-run state upon returning from the AST.

Errors:

E.INV	invalid value (task name string outside the issuing task limits)
E.TNF	task not in system
E.TRUM	task not stopped (informational return code, not an error – CPU carry flag not set)

.RPOI

Request Task and Pass Offspring Information.

Input	
HL	Address of task descriptor.
Returns	
AF	Error/success code.

Requests the specified task, and chains to it by passing all of the parent connections from the issuing task to the requested task. Optionally, the directive can pass a command line and/or queue a data block to the requested task.

The task descriptor has the following format:

Offset	Size	Value
0	6 bytes	Task name
6	2 bytes	Address of command line, or zero for none
8	2 bytes	Length of command line
10	1 byte	Task attributes (only the TA.MCR bit is used)
11	1 byte	User ID
12	1 byte	Group ID
13	2 bytes	Terminal device name
15	1 byte	Terminal unit number
16	2 bytes	Address of data block to send, or zero for none
18	2 bytes	Length of data block

Only privileged tasks may set the UIC and TI: of the requested task; the values are ignored if the directive is issued by a non-privileged task.

Errors:

E.INV	invalid value (task name string, command line or data buffer outside issuing task limits)
E.TNF	task not in system
E.TRUN	task is already running
E.NOMEM	not enough pool memory to perform the operation

.RQST

Request installed Task.

Input	
HL	Address of task descriptor.
D	Status format (0=short, 1=long)
E	Number of event flag to set when the offspring task emits status (ignored if BC=0)
BC	Address of the Exit Status Block (ESB) to receive status information (request and connect), or zero (request only)
Returns	
AF	Error/success code.

Requests an installed task and optionally connects to it.

The task descriptor has the following format:

Offset	Size	Value
0	6 bytes	Task name
6	2 bytes	Address of command line, or zero for none
8	2 bytes	Length of command line
10	1 byte	Task attributes
11	1 byte	User ID
12	1 byte	Group ID
13	2 bytes	Terminal device name
15	1 byte	Terminal unit number
16	2 bytes	Address of data block to send, or zero for none
18	2 bytes	Length of data block

If a command line is specified, the command will be passed to the task. If the address of a data block is specified, the block will be queued to task's the receive list.

If the address of of an Exist Status Block (ESB) is specified, the system queues an Offspring Control Block (OCB) to the offspring task and clears the specified event flag. When the offspring exits or emits status, the kernel sets the event flag and stores the status information into the ESB:

- If the short status format is specified, only the offspring status word (2 bytes) will be stored at the ESB address.

- If the long status format is specified, an additional 14-byte of status information will be stored (16 bytes in total):

Offset	Size	Value
0	2 bytes	Offspring status word
2	1 byte	Exit cause (TKTN code, always zero for kernel version 6.37)
3	3 bytes	Offspring's terminal name and unit
6	6 byte	Offspring task name
12	1 byte	Offspring task attributes (T.ATTR)
13	2 bytes	Offspring task status (T.ST)
15	1 byte	Reserved

Notes:

- The requested task must be installed in the system.
- If the partition in which the requested task is to run is already occupied, the kernel places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority and resource availability, when the partition is free. Lower-priority tasks occupying the partition may be checkpointed to free space for the requested task.
- Successful completion of the system call means that the task has been activated, not that the task is actually running.
- The requested task always runs at its installed priority.
- The inherits the protection UIC (user and group ID) and TI: terminal assignment from the requesting task. The protection UIC determines the task's access rights for accessing files. Privileged tasks can specify a different protection UIC.

Errors:

E.INV task descriptor, command line or data buffer outside requesting task limits
E.NOMEM not enough system memory to perform the operation, task size is bigger than the size of the main partition or the user-controlled partition, or not enough space is left in the checkpoint file or checkpointing support is not included in the system or checkpointing is disabled.
E.TNF task not in system
E.TRUN task is already active

.RUN

Request installed task at a specified time.

Input	
HL	Address of extended task descriptor.
Returns	
AF	Error/success code.

Requests a task at a specified future time, and optionally to be requested periodically.

The extended task descriptor has the following format:

Offset	Size	Value
0	6 bytes	Task name
6	2 bytes	Address of command line, or zero for none
8	2 bytes	Length of command line
10	1 byte	Task attributes
11	1 byte	User ID
12	1 byte	Group ID
13	2 bytes	Terminal device name
15	1 byte	Terminal unit number
16	2 bytes	Address of data block to send, or zero for none
18	2 bytes	Length of data block
20	2 bytes	Relative schedule time magnitude
22	1 byte	Relative schedule time units
23	2 bytes	Reschedule interval magnitude
25	1 byte	Reschedule interval units

The schedule time is specified as a value relative to the time the system call is issued. If the reschedule interval is zero, the task will be requested only once at the specified time.

The time units are as follows:

Value	Meaning
0	Ticks
1	Seconds
2	Minutes
3	Hours

Notes:

- The requested task must be installed in the system.
- The system always sets the TI: device for the requested task to the system console terminal CO:
- If the partition in which the requested task is to run is already occupied, the kernel places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority and resource availability, when the partition is free. Lower-priority tasks occupying the partition may be checkpointed to free space for the requested task.
- Successful completion of the system call means that the task has been activated, not that the task is actually running.
- The requested task always runs at its installed priority.
- Only privileged tasks can specify a protection UIC.
- If the task is still active after the reschedule interval elapses, the kernel will not issue a new re-initiation request, but a new reschedule interval will be started nevertheless.

Errors:

E.PRIV	privileged operation
E.TNF	task not in system
E.NOMEM	not enough system memory (pool) to perform the operation
E.INV	invalid value (invalid units or overflow happened during conversion to ticks)

.SDAT

Set system Date and Time.

Input	
HL	Address of an 8-byte buffer containing the date and time.
Returns	
AF	Error/success code.

Sets the system date and time. If the machine has an RTC chip, the chip registers are updated with the new information.

The issuing task must be privileged. The date/time structure contains the following fields:

Offset	Size	Value
0	2 bytes	Year, as four packed BCD digits.
2	1 byte	Month, as two packed BCD digits in the range 1..12
3	1 byte	Day of month, as two packed BCD digits in the range 1..31
4	1 byte	Hour, as two packed BCD digits in the range 0..23
5	1 byte	Minutes, as two packed BCD digits in the range 0..59
6	1 byte	Seconds, as two packed BCD digits in the range 0..59
7	1 byte	Day of week in the range 1..7 (1=Sunday, 2=Monday, etc.)

Errors:

E.INV invalid value (structure address outside issuing task limits)
E.BADOP invalid operation (structure field out of range)
E.PRIV task must be privileged

.SDIR

Set Current Directory.

Input	
HL	Pointer to 9-character directory name.
C	Function code.
Returns	
AF	Error/success code.

The Function code can be:

SD.TSK	Set current task's directory
SD.TI	Set current terminal's directory

Notes:

- The name must be padded with spaces (20h) if shorter than 9 characters.
- The system call merely stores the directory name in the Task Context structure, without any validation and without ensuring that the directory actually exists.

Errors:

E.INV	invalid function code, or directory name string outside task limits
E.SSNF	user session not found

.SEND

Send Data.

Input	
HL	Pointer to a 6-character destination task name.
DE	Address of buffer containing the data to send.
BC	Length of data.
Returns	
AF	Error/success code.

Queues First-In-First-Out (FIFO) a data block for a task to receive.

Notes:

- The system currently does not impose any restrictions on the size of the data block. Thus, a task can potentially deplete the system memory by indiscriminately using this function to send data to a task that is not processing it.

Errors:

E.BADOP	the length of data to send is zero, or the destination task is an ACP
E.INV	data buffer or task name string outside task limits
E.TNF	task not in system
E.NOMEM	not enough pool memory to perform the operation
E.TRUN	task is active and running (info, not an error)

.SETF

Set single Event Flag.

Input	
E	Event flag number.
Returns	
AF	Previous event flag state on success, else an error code.

Sets the indicated event flag, On success, the function returns in the CPU accumulator the previous flag state: zero if the event flag was unset, non-zero if was set.

Notes:

- If called from an AST routine while the issuing task is stopped or suspended for the same event flag, the task execution will be resumed upon return from the AST routine.

Errors:

E.BADFL invalid event flag number (outside 1..32 range)

.STLO

Stop for Logical OR of Event Flags.

Input	
DE	Address of a 4-byte flag mask.
Returns	
AF	Error/success code.

Stops the execution of the issuing task until at least one of the indicated event flags is set. The task is not stopped if any of the indicated flags are already set when the task issues the directive. When stopped, the task's priority effectively drops to zero and the task can be checkpointed by another task, regardless of that task's priority.

A task that is stopped for one or more event flag can only become unstopped by setting the corresponding event flag; it cannot become unstopped by the .RESUM function call or the MCR UNStop command.

Notes:

- Bit 0 of the first byte of the mask corresponds to event flag 1; bit 7 of the 3rd (last) byte corresponds to event flag 32.
- The kernel does not arbitrarily clear event flags when Wait For conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.

Errors:

E.INV invalid value (flag mask array outside task limits)

.STOP

Stop a Task.

Input	
HL	Pointer to 6-character task name, or zero to stop the current task.
Returns	
AF	Error/success code.

Stops the specified task. When stopped, the task's priority effectively drops to zero and the task can be checkpointed by another task, regardless of that task's priority.

Stopped tasks can still receive receive ASTs. If an AST occurs while the task is stopped, the task will enter the ready-to-run state for the duration of the AST routine execution, and then return to the stopped state once the AST routine exits unless explicitly resumed by a .RESUM directive or by another task.

Notes:

- A non-privileged task can only stop itself, or stop tasks started from the same terminal as the requesting task.
- If the task is currently executing and AST routine, the task will be stopped upon exiting from the AST.

Errors:

E.INV	invalid value (task name string outside issuing task limits)
E.TNF	task not in system
E.PRIV	non-privileged task tried to stop a task started from another terminal
E.TSTP	task already stopped

.STSE

Stop for single Event Flag.

Input	
E	Event Flag number.
Returns	
AF	Error/success code.

Stops the issuing task until the specified event flag is set. If the flag is set at the time of the call, the function returns immediately and the task is not stopped. When stopped, the task's priority effectively drops to zero and the task can be checkpointed by another task, regardless of that task's priority.

A task that is stopped for one or more event flag can only become unstopped by setting the corresponding event flag; it cannot become unstopped by the .RESUM function call or the MCR UNStop command.

Errors:

E.BADFL invalid event flag number (outside the 1..32 range)

.SUPER

Enter/exit System Database Access mode.

Input	
C	1 to map the system data area, 0 to restore the original task mapping.
Returns	
AF	Error/success code.

Maps or unmaps the System Database to the issuing task space. Mapping the Database allows a privileged task to access and/or change the system data structures, and to call certain kernel routines.

Unlike RSX-11M, RSX180/280 privileged tasks are not mapped to the kernel space on startup.

Notes:

- The directive is available to privileged tasks only.
- The task stack and code section that issues this system call, the code that accesses the system database and/or calls kernel routines, must be located in the first 16K bytes of the task address.
- Privileged tasks are potentially hazardous to a running system. Therefore, observe great care when accessing and modifying the system database, since corruption of the system data structures can lead to obscure bugs and even to system crashes. Access to the system data should be preferably done after disabling task dispatching via the SYSLVL kernel variable.

Errors:

E.PRIV privileged operation

.SVTBL

Specify SST vector table.

Input	
DE	SST table address, or zero to deassign the vector table.
Returns	
AF	Error/success code.

Specifies the address of a table of SST service routines entry points for use by the issuing task. The SST.IN system include file defines the table layout and symbolic names for the table entries.

~~For RSX180, the table consists of a single entry for the Illegal Instruction Trap of the Z180 CPU.~~

For RSX280, the table consists of 8 words:

Word offset	Symbolic name	Trap
0	SST.EI	Extended instruction (EPU-reg)
1	SST.E2	Extended instruction (EPU-mem)
2	SST.PR	Privileged instruction
3	SST.SC	Unrecognized System Call
4	SST.AC	Access Violation
5	SST.DV	Divide overflow
6	SST.SS	Single-Step
7	SST.BP	Breakpoint-on-halt

A zero entry in the table indicates that the task does not want to process the corresponding SST.

Errors:

E.INV invalid value (SST table outside task limits)

.WTDAT

Wait for Data.

Input	
None.	
Returns	
AF	Error/success code.

Suspends the execution of the issuing task until data is received at the Receive Queue of the task. The task is not suspended if data is available when the task issues the directive.

Errors:

E.BADOP invalid operation (directive called from an AST routine)

.WTLO

Wait for Logical OR of Event Flags.

Input	
DE	Address of a 4-byte flag mask.
Returns	
AF	Error/success code.

Suspends the execution of the issuing task until at least one of the indicated event flags is set. The task is not suspended if any of the indicated flags are already set when the task issues the directive.

Notes:

- Bit 0 of the first byte of the mask corresponds to event flag 1; bit 7 of the 3rd (last) byte corresponds to event flag 32.
- The kernel does not arbitrarily clear event flags when Wait For conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.

Errors:

E.INV invalid value (flag mask array outside task limits)

.WTSE

Wait for single Event Flag.

Input	
E	Event Flag number
Returns	
AF	Error/success code.

Suspends the execution of the issuing task until the indicated event flag is set. If the flag is set at the time of the call, the function returns immediately and the task is not suspended. When suspended, task priority effectively remains unchanged, allowing the task to compete for memory space.

Errors:

E.BADFL invalid flag number (outside 1..32 range)

Glossary

AST	Asynchronous System Trap.
BCD	Binary-Coded Decimal.
CLI	Command Language Interpreter.
CLICB	CLI Control Block.
CPU	Central Processing Unit.
DCB	Device Control Block.
DPB	Directive Parameter Block.
EFN	Event Flag Number.
ESB	Exit Status Block
FIFO	First-In, First-Out queue.
IOSB	I/O Status Block.
LUN	Logical Unit Number.
MCR	Monitor Console Routine.
OCB	Offspring Control Block.
OS	Operating System.
PCB	Partition Control Block.
SCB	Session Control Block.
STD	System Task Directory.
SST	Synchronous System Trap.
TCB	Task Control Block.
UCB	Unit Control Bloc.
UIC	User Identification Code.