EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

PS/CO/NOTE 86-10
24 March 1986

Project: SMACC
Domain: SYSTEM
Category: USMAN
Status: DRAFT

# *SMACC PROGRAMMING USER'S MANUAL.*

N. de Metz-Noblat

Abstract

This manual describes the current state of the programming environment for the SMACC in the PS control system.

Geneva, Switzerland

# TABLE OF CONTENTS

Section | Page

1 INTRODUCTION TO THE SMACC . . . . 1

1.1 SMACC : an M68000 based Auxiliary Crate Controller . . . . 1
1.2 The SMACC basic software . . . . 2
1.3 Notations . . . . 2

2 RMS68K THE SMACC OPERATING SYSTEM . . . . 3

2.1 TASK MANAGEMENT . . . . 3
2.1.1 CRTCB : Create Task Control Block . . . . 6
2.1.2 START : Start Task . . . . 6
2.1.3 TERM : Terminate Self . . . . 6
2.1.4 ABORT : Abort Self . . . . 6
2.1.5 WAIT : Enter wait state . . . . 7
2.1.6 SUSPND: Suspend self . . . . 7
2.1.7 RELINQ: Relinquish . . . . 7
2.1.8 WAKEUP: Wakeup . . . . 7
2.1.9 RESUME: Resume a Target Task . . . . 7
2.1.10 SETPRI: Set Priority . . . . 7
2.1.11 STOP : Stop Target Task . . . . 8
2.1.12 TERMT : Terminate Target Task . . . . 8
2.2 MEMORY MANAGEMENT . . . . 9
2.2.1 GTSEG : Allocate a Segment . . . . 9
2.2.2 DESEG : Deallocate a Segment . . . . 10
2.2.3 DCLSHR: Declare a Segment Shareable . . . . 10
2.2.4 ATTSEG: Attach a Shareable Segment . . . . 10
2.2.5 SHRSEG: Grant Shared Access to Another Task . . . . 10
2.2.6 TRSEG : Transfer a Segment . . . . 10
2.2.7 RCVSA : Receive Segment Attributes . . . . 10
2.3 INTER-TASK COMMUNICATION . . . . 11
2.3.1 GTASQ : Allocate Asynchronous Queue . . . . 12
2.3.2 SETASQ: Set ASQ/ASR Status . . . . 12
2.3.3 DEASQ : Deallocate Asynchronous Queue . . . . 12
2.3.4 QEVNT : Queue Event To Task . . . . 12
2.3.5 WTEVNT: Wait for Event . . . . 12
2.3.6 RDEVNT: Read Event . . . . 12
2.3.7 RTEVNT: Return from Event Service . . . . 13
2.4 TASK SYNCHRONISATION . . . . 14
2.4.1 ATSEM : Attach to Semaphore . . . . 15
2.4.2 CRSEM : Create a Semaphore . . . . 16
2.4.3 DESEM : Detach from Semaphore . . . . 16
2.4.4 DESEMA: Detach all Semaphores . . . . 16
2.4.5 WTSEM : Wait on Semaphore . . . . 16
2.4.6 SGSEM : Signal Semaphore . . . . 16
2.5 TIME AND DELAY MANAGEMENT . . . . 17
2.5.1 STDTIM: Set System Date and Time . . . . 17
2.5.2 GTDTIM: Get System Date and Time . . . . 17
2.5.3 DELAY : Delay Self . . . . 17
2.5.4 DELAYW: Delay and Wait . . . . 18

Section Page

2.5.5 RQSTPA: Request Periodic Activation . . . . . . . . . . . . . . 18
2.6 SPECIAL FUNCTION CONTROL . . . . . . . . . . . . . . . . . . . . 19
2.6.1 MONITOR TASK . . . . . . . . . . . . . . . . . . . . . . . . 19
2.6.2 SERVER TASK CONTROL . . . . . . . . . . . . . . . . . . . . 20
2.6.2.1 SERVER: Establish Server . . . . . . . . . . . . . . . . 20
2.6.2.2 AKRQST: Acknowledge Service Request . . . . . . . . . . . 20
2.6.2.3 DERQST: Set User/Server Request Status . . . . . . . . . 20
2.6.2.4 DSERVE: Deallocate Server functions . . . . . . . . . . . 20
2.6.3 EXCEPTION MONITOR TASK CONTROL . . . . . . . . . . . . . . 21
2.6.3.1 EXMON : Attach Exception Monitor . . . . . . . . . . . . 21
2.6.3.2 DEXMON: Detach Exception Monitor . . . . . . . . . . . . 21
2.6.3.3 EXMMSK: Set Exception Monitor Mask . . . . . . . . . . . 21
2.6.3.4 RSTATE: Receive Task State . . . . . . . . . . . . . . . 22
2.6.3.5 PSTATE: Put Task State . . . . . . . . . . . . . . . . . 22
2.6.3.6 REXMON: Run Task under Exception Monitor . . . . . . . . 22
2.6.4 LOCAL HANDLING OF TRAPS AND EXCEPTIONS . . . . . . . . . . 23
2.6.4.1 EXPVCT: Announce Exception Vectors . . . . . . . . . . . 23
2.6.4.2 TRPVCT: Announce Trap Vectors . . . . . . . . . . . . . . 23
2.7 INTERRUPT SERVICE MANAGEMENT . . . . . . . . . . . . . . . . . 24
2.7.1 CISR : Connect Interrupt Service Routine . . . . . . . . . 24
2.7.2 RTE : return from interrupt . . . . . . . . . . . . . . . 24
2.7.3 SINT : Simulate Interrupt . . . . . . . . . . . . . . . . 25
2.8 TASK OR SYSTEM INFORMATION DIRECTIVES . . . . . . . . . . . . 26
2.8.1 TSKATTR : Task Attributes . . . . . . . . . . . . . . . . 26
2.8.2 MOVELL: Move from Logical Address . . . . . . . . . . . . 26
2.8.3 MOVEPL: Move from Physical Address . . . . . . . . . . . . 26
2.8.4 TSKINFO: Return Copy of Task Control Block . . . . . . . . 26
2.8.5 SNAPTRAC: Snapshot of System Trace . . . . . . . . . . . . 26
2.9 CDIR : Configure Directive . . . . . . . . . . . . . . . . . . 26
2.10 Input/Output . . . . . . . . . . . . . . . . . . . . . . . . . 27

3 PS MEMORY LAYOUT ON THE SMACC . . . . . . . . . . . . . . . . . 28

3.1 SMACC hardware requirements . . . . . . . . . . . . . . . . . . 28

3.2 General memory partitioning from the RMS68K point of view. 28
3.2.1 Partition 0: supervisor data . . . . . . . . . . . . . . . . 29
3.2.2 Partition 1 : RAM unprotected area . . . . . . . . . . . . . 29
3.2.3 Partition 2: system sofware code 30
3.3 Cluster notion . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.4 The memory environment of a task . . . . . . . . . . . . . . . 31

4 START-UP OF THE SMACC . . . . . . . . . . . . . . . . . . . . . 32

4.1 Hardware start-up . . . . . . . . . . . . . . . . . . . . . . . 32
4.2 RMS68K start-up . . . . . . . . . . . . . . . . . . . . . . . . 33
4.2.1 RMS68K Pre-initializer . . . . . . . . . . . . . . . . . . . 33
4.2.2 RMS68K System Initializer . . . . . . . . . . . . . . . . . 33
4.2.3 Basic software initialization . . . . . . . . . . . . . . . 34
4.2.4 Application software initilization . . . . . . . . . . . . . 35
4.2.5 Exchange of magic memory addresses between FEC and SMACC . . . . 35

5 STANDARD FACILITIES ON THE SMACC . . . . . . . . . . . . . . . 37

Section | Page

5.1 OPERATING SYSTEM, LANGUAGES, AND LIBRARIES . . . . . . . . . . . . . 37
5.1.1 Operating System . . . . . . . . . . . . . . . . . . . . . . . . . 37
5.1.2 NODAL-68K . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
5.1.3 Nodal functions to access RMS68K . . . . . . . . . . . . . . . . . 38
5.1.4 Real arithmetic routines . . . . . . . . . . . . . . . . . . . . . 39
5.1.5 General library routines . . . . . . . . . . . . . . . . . . . . . 39
5.2 COMMUNICATIONS SOFTWARE . . . . . . . . . . . . . . . . . . . . . 40
5.2.1 Level 2 primitives available on the FEC . . . . . . . . . . . . . 40
5.2.2 Datagram service (not yet implemented) . . . . . . . . . . . . . . 42
5.2.3 Remote Procedure Call (being implemented) . . . . . . . . . . . . 43
5.2.4 IMEX/EXEC (not yet implemented) . . . . . . . . . . . . . . . . . 44

6 SPECIAL FACILITIES ON THE SMACC . . . . . . . . . . . . . . . . . . . 45

6.1 Camac LAM and front-pannel interrupts handling . . . . . . . . . . 45
6.2 Error handling . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
6.2.1 Monitor task . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
6.2.2 Logging errors on terminal . . . . . . . . . . . . . . . . . . . . 46
6.2.3 Logging errors to the FEC . . . . . . . . . . . . . . . . . . . . 46
6.3 Error number convention . . . . . . . . . . . . . . . . . . . . . . 46
6.4 Power-fail interrupt . . . . . . . . . . . . . . . . . . . . . . . . 47

7 PROGRAM PRODUCTION FOR THE SMACC . . . . . . . . . . . . . . . . . . 48

7.1 Software Architecture. . . . . . . . . . . . . . . . . . . . . . . 49
7.2 Existing program development tools. . . . . . . . . . . . . . . . . 49
7.3 Program production procedures. . . . . . . . . . . . . . . . . . . 50
7.3.1 Assembly language source files. . . . . . . . . . . . . . . . . . 50
7.3.1.1 code sections. . . . . . . . . . . . . . . . . . . . . . . . . . 51
7.3.1.2 Static variables reservation sections. . . . . . . . . . . . . . 51
7.3.2 P+ source files. . . . . . . . . . . . . . . . . . . . . . . . . . 51
7.3.3 Planc source files. . . . . . . . . . . . . . . . . . . . . . . . 52
7.3.4 First link-edit. . . . . . . . . . . . . . . . . . . . . . . . . . 52
7.3.5 How to create a loadable image of the SMACC. . . . . . . . . . . 52
7.3.6 Remarks on the program production procedures. . . . . . . . . . . 54
7.3.7 Annexe: Supported Program production shema. . . . . . . . . . . . 55

8 DEBUGGING A PROGRAM ON THE SMACC . . . . . . . . . . . . . . . . . . 56

8.1 FEC Remote debugger . . . . . . . . . . . . . . . . . . . . . . . . 57
8.2 Using NODAL tools . . . . . . . . . . . . . . . . . . . . . . . . . 58
8.3 MoniCa debugger . . . . . . . . . . . . . . . . . . . . . . . . . . 58
8.4 Other debug informations . . . . . . . . . . . . . . . . . . . . . 58
8.4.1 System does not start . . . . . . . . . . . . . . . . . . . . . . 59
8.4.2 System crashes . . . . . . . . . . . . . . . . . . . . . . . . . . 59
8.4.2.1 System crash area . . . . . . . . . . . . . . . . . . . . . . . 59
8.4.2.2 Nothing at crashsav . . . . . . . . . . . . . . . . . . . . . . 59
8.4.2.3 System trace table . . . . . . . . . . . . . . . . . . . . . . . 60

Section Page

APPENDIX

A P-PLUS INTERFACE TO RMS68K . . . . . . . . . . . . . . . . . . . 61

B Nodal interface to RMS68K . . . . . . . . . . . . . . . . . . . 79

C RMS68K ERROR CODES . . . . . . . . . . . . . . . . . . . . . . 89

D ACCES TO CAMAC FROM THE SMACC . . . . . . . . . . . . . . . . . 93

E Abolute variables for the SMACC . . . . . . . . . . . . . . . . 97

F Interface to NODAL . . . . . . . . . . . . . . . . . . . . . . 100

G Example of writing a NODAL compatible routine . . . . . . . . . 101

H Example of testing an ISR from NODAL . . . . . . . . . . . . . 105

I STANDARD EVENTS STRUCTURE . . . . . . . . . . . . . . . . . . . 109

J RMS68K usefull internal tables . . . . . . . . . . . . . . . . 113

K RMS68K SYSGEN PARAMETERS . . . . . . . . . . . . . . . . . . . 119

L REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . . 123

M GLOSSARY . . . . . . . . . . . . . . . . . . . . . . . . . . . 127

Index 133

# 1 INTRODUCTION TO THE SMACC

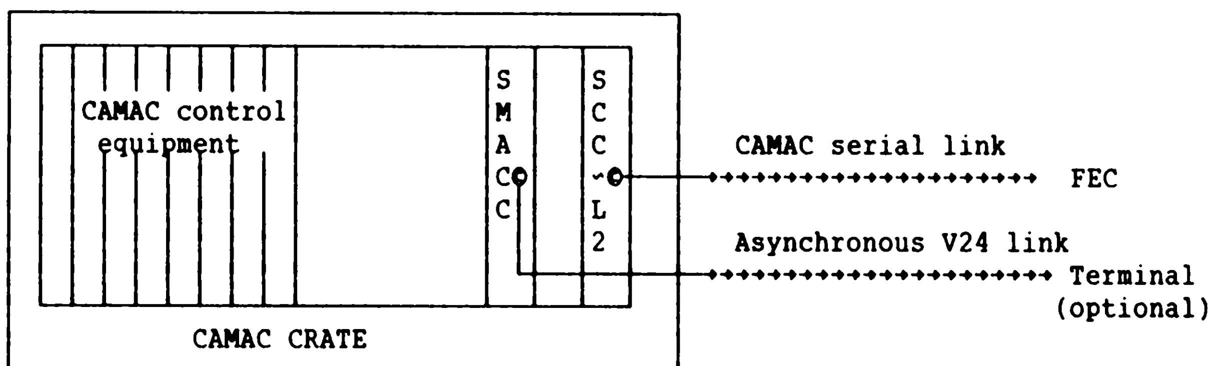
## 1.1 SMACC : an M68000 based Auxiliary Crate Controller

The SMACC is an Auxiliary Crate Controller (ACC) based on a MOTOROLA 68000 (8MHz internal clock) microprocessor destinated to be placed into a CAMAC crate in the PS control system.

Different operation modes are intended to be used :

1) Standard Mode of operation

The standard mode of operation of a SMACC is to provide the work of an Auxiliary Crate Controller.
A crate controller type SCC-L2 controls the whole crate in conjunction with a Front-End Computer (FEC) of the PS control system.
The work of the SMACC is to handle locally all or part of the equipment located in this crate and thus reduce the work to be done in the FEC.
An optional terminal allows for the use of local NODAL programs and thus control the operations done on this SMACC.




2) Operating with a MacIntosh personal microcomputer.

A NODAL interpreter - version MC68000 - is available on the SMACC and on the MacIntosh personal microcomputer.

The MacIntosh can be connected to the SMACC in two ways:

- It is able to drive the serial CAMAC link instead of the FEC and it offers in this case, the same CAMAC access as from a FEC for a NODAL program.

- It is also able to emulate the local terminal and in this case it offers (to the interactive NODAL program executing in the SMACC) the access to its disks and printer.

The MacIntosh is also able to emulate a Dicodime : a routine library to acces basic functions and graphic functions has been developped.

It is possible to transfer files between the MacIntosh and the PRDEV (PRogram DEVelopment computer) trough the PACX network, connecting the MacIntosh as terminal of the PRDEV.

3) Stand-Alone.

For some applications, the SMACC can also be used as a general purpose stand-alone microprocessor on which NODAL and the whole PS programming environment can be used.

## 1.2 The SMACC basic software

This manual describes the SMACC programming environment in the PS control system.

It describes the following:

- RMS68K the operating system and its primitives.
- The PS memory layout for the SMACC.
- How does run start-up and what happends at this time.
- Which languages, libraries and communication softwares are offered in standard on the SMACC.
- Which are the special facilities for interrupts and errors handling.
- Which facilities exist for program testing and debugging.
- The program production.

## 1.3 Notations

To make this manual easier to read, some standard expressions and abbreviations have been used:

**$** : This character indicates that the following value is an hexadecimal value.

Kb : Kilo Bytes = 1024 bytes. All memory length are expressed in bytes. The MC68000 microprocessor is able to handle bytes, 16 bits words and 32 bits double words.

RMS68K : Real-time Multitasking Software for Mc68000 family. This is the name of the monitor used on the SMACC.

## 2 RMS68K THE SMACC OPERATING SYSTEM

The Motorola MC68000 family Real-time Multitasking Software (RMS68K) is the operating system chosen for the SMACC. This operating system provides a full set of real-time facilities. RMS68K provides the following functions:

- receive all hardware and software interrupts and dispatch them to the proprer task for processing.
- act as a dispatcher of tasks competing for use of the processing unit.
- provide inter-task communication and synchronisation.
- provide dynamic memory allocation.
- provide a system initialization facility.
- provide a protection of the user environment.
- provide diagnostic feedback during error conditions.

Applications (whether for process control or for console devices) run as collections of tasks, procedures and data segments under the management of the RMS68K operating system. The addressing space of the MC68000 being large, and there being no virtual memory in our context, addresses in code space (tasks and procedures) and data space is unique and unambiguous. Thus there are no artificial constraints on which tasks may call which procedures and memory segments are used purely as a tool for modularity and protection. However, to gain memory space, EPROM can be used and thus all loadable modules should always be either purely dynamic data or purely code.

### 2.1 TASK MANAGEMENT

Under the control of RMS68K, programs which make up an application are executed as tasks.
Tasks are grouped together within a "session", i.e. by default any action can only affect a task with the same session number.
Two basic types of tasks are distinguished : user and system tasks. Both execute in the user mode of the MC68000 <i> . System tasks have acces to all resources and may perform all operations regardless of session number.
In the SMACC context, all tasks are system tasks running in session 1.

An RMS68K task is composed of a Task Control Block (TCB), up to 4 program or data memory segments <ii> and one asynchronous service queue (ASQ).

---

<i> The processor operates in one of two states of privilege: the user state or the supervisor state. The privilege state determines which operations are legal, is used by memory management to control acces to a part of the memory, and is used to choose between the supervisor stack pointer and the user stack pointer in instruction references.

<ii> The number of segments per task is limitated in standard as delivered by MOTOROLA to 4 segments. It should be possible to grow up this number, but this imply some local modifications that are not handled by MOTOROLA.

A task can be in ten different states:

| | |
|---|---|
| - non existent : | task has not been created. |
| - dormant : | task has been created by CRTCB directive,but is not yet started, or task has been stopped by the STOP directive. |
| - ready : | task is ready to be executed, but another task with higher priority is yet in the "running" state. |
| - running : | task is being executed. |
| - waiting : | task has issued a WAIT directive and wait for a WAKEUP from another task (or from a periodic activation request). |
| - suspended : | task has issued a SUSPND directive and is waiting for a RESUME from another task (or from a periodic activation request).<br>If a RESUME directive is issued for a task which is not in the "suspend" state, the directive has no effect. If a WAKEUP is issued for a task which is not in the "wait" state, the WAKEUP directive is pending until the task actually does go in the "wait" state. |
| - waiting for an event : | task is waiting for an event to come into its ASQ. |
| - waiting for a command : | task is waiting for a debugger command. |
| - waiting for a delay : | task is waiting for a delay (or for an event or a WAKEUP from another task if DELAYW). |
| - waiting on a semaphore: | task is waiting on a semaphore. |

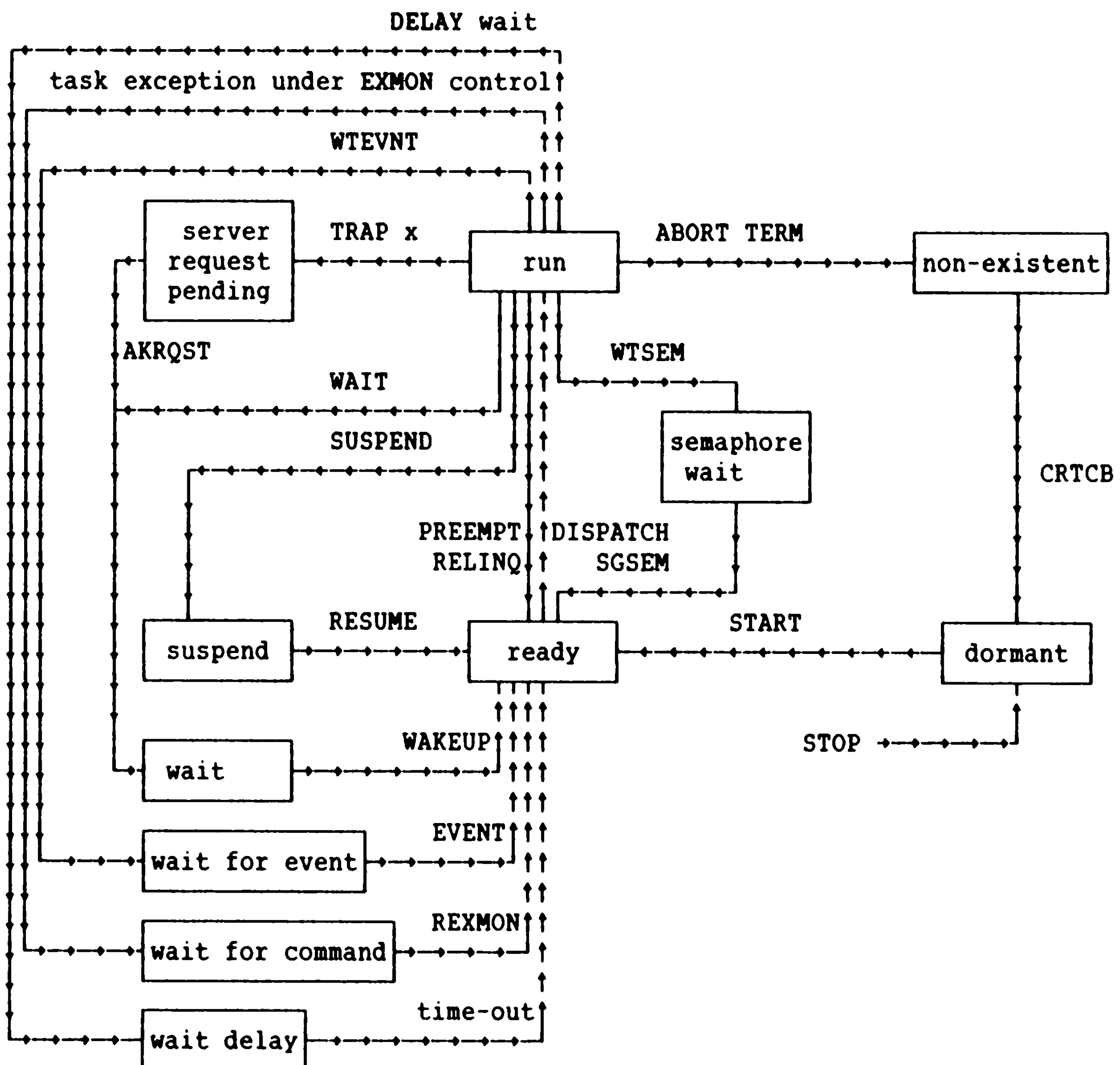
The dispatch cycle of RMS68K is entered at any time a task is removed from the "running" state. The reasons for a task to be removed from the "running" state are listed below:
- task relinquishes execution.
- task changes its own task state.
- an event is placed into any task's Asynchronous Service Queue (ASQ).
- task performs a semaphore wait operation.
- task exceeds the maximum execution time slice allowed.

The task scheduled for execution during a dispatch cycle of RMS68K is the task residing in the "ready" state with the higher priority and since the longest time in case of conflict.
When a task is removed from the "running" state from any reason other than STOP, ABORT, TERMT, or TERM directive, the task resumes execution at the next intruction following the last executed instruction.

Following primitives are available for task management :

- CRTCB : target task is created and goes to "dormant".
- START : target task goes to "ready" from "dormant".
- TERM : task terminates itself.
- ABORT : task aborts itself.
- WAIT : task moves itself to "wait".
- SUSPND : task moves itself to "suspend".
- RELINQ : task moves itself to "ready" from "running".
- WAKEUP : target task goes to "ready" from "wait".
- RESUME : target task goes to "ready" from "suspend".
- SETPRI : target task's priority is modified.
- STOP : target task goes to "dormant" from any state.
- TERMT : target task is terminated from any state.

The following diagram shows out the different states of a task:



### 2.1.1 CRTCB : Create Task Control Block

RMS68K knows that a task exists only if a Task Control Block was created. Each TCB needs 512 bytes of memory <i> . CRTCB allows creation of a new TCB and thus a new task.

When a task is created, it is given :

- a four bytes name (and a session number -- always 1 in the SMACC).
- a limit priority (in the range 0..255, 0 being the lowest priority)
- an initial current priority :
  This priority can be changed at any time to a value less or equal to the task's limit priority.
  A given task cannot affect another task that has a current priority greater than its own limit priority.
- A task entry point :
  the task begins execution at this location when a START directive is issued for this task.
- It can be specified if task is a system task and if system stops on task abort.

### 2.1.2 START : Start Task

RMS68K put the target task from the "dormant" state to the "ready" state, based on its current priority, to await execution. The initial values of the registers can be specified.

### 2.1.3 TERM : Terminate Self

RMS68K halts execution of the requesting task and frees all resources attached to it.
This results in the normal termination of task execution. <ii>

### 2.1.4 ABORT : Abort Self

RMS68K halts the execution of the requesting task and frees all resources attached to it.

---

<i> This point limit the maximum number of tasks that can be created in the system.

<ii> Notice that after a TERM or an ABORT directive, the TCB is lost once the monitoring tasks awaken.

### 2.1.5 WAIT : Enter wait state

RMS68K places the requesting task in the "wait" state until a WAKEUP directive is issued by another task.

### 2.1.6 SUSPND: Suspend self

RMS68K stops the execution of the requesting task and moves it to the "suspend" state. The execution of the task is started again only by a RESUME directive issued by another task. When the requesting task is resumed, the execution proceeds in sequence.

### 2.1.7 RELINQ: Relinquish

RMS68K places the requesting task in the "ready" state so that RMS68K may enter a dispatch cycle.
This directive is useful to force an exchange between tasks of the same priority before a time-slice interrupt <i> .
Notice that the priority of the current task is temporarely reduced to the nearest multiple of 16.

### 2.1.8 WAKEUP: Wakeup

RMS68K moves the specified task from the "wait" state to the "ready" state to await execution. If the specified task is not currently in the "wait" state, a wakeup pending condition is set and takes effect the next time the task goes into the "wait" state.

### 2.1.9 RESUME: Resume a Target Task

RMS68K resumes the execution of a task that is moved from the "suspend" state to the "ready" state to await execution.
If the specified task was not in the "suspend" state, RESUME has no effect.

### 2.1.10 SETPRI: Set Priority

RMS68K changes the current priority of the target task to a value that must be less than its limit priority.
A user task cannot alter priority of a system task.

---

<i> The time-slicer is activated only every 200ms to force a relinquish of the execution of a task and to leave the CPU to another task with the same priority

### 2.1.11 STOP : Stop Target Task

RMS68K stops the execution of the specified task and moves it to the "dormant" state, with all resources still attached. In particular, dynamically allocated memory segments for data (or code) are not released.
To stop a system task, its name must be specified and the requestor must not be a user task.

If no task name is specified, RMS68K select one user task and stops it.
This facility is used to stop all user tasks in a session.

This directive cannot be used to STOP the calling task.

### 2.1.12 TERMT : Terminate Target Task

In the same manner as for STOP directive, RMS68Kforces an ABORT directive for target task.

Note :The TERMT directive may require several milliseconds before the target TCB is eliminated from the the system and that this task can be created again because this work is done only at next dispatch cycle of RMS68K.

## 2.2 MEMORY MANAGEMENT

RMS68K has been designed to handle a Memory Management Unit (MMU) that is non-existent on the SMACC. This implies that there is no difference between a "logical" address and a "physical" address. <i>
The memory management directives have to initiate the MMU contents at each scheduling and from the RMS68K point of view a task can only acces to 4 different segments at a time.

Each time a task issues a directive, RMS68K verifies that all arguments are located in a space that is allowed for this task and reject the call if not.

Each segment is a part of the memory, with any length but beginning at a location witch is a multiple of 256 bytes, and which can receive the following attributes:

- read-only or physical ROM.
- shareable globally (or locally : only between tasks of the same session).
- memory mapped I/O space.

The memory management directives of RMS68K provide in fact two different services:

- a dynamic memory allocation package.
- a control of the use of memory in each directive call <ii> .

The following primitives are available for memory management:

- GTSEG : Allocates a memory segment to a task.
- DESEG : Deallocates a memory segment from a task.
- DCLSHR : Declare a memory segment shareable.
- ATTSEG : Attach a shareable segment to self.
- SHRSEG : Place a shareable segment into target task's address space.
- TRSEG : Transfer a segment from self to a target task.
- RCVSA : Read segment attributes.

### 2.2.1 GTSEG : Allocate a Segment

RMS68K allocates the smallest number of memory pages which satisfies the specified length. Page size is 256 bytes (determined at SYSGEN).
The task to receive the new segment can be the requesting task or another task that must be in the "dormant" state.
The physical address of the segment can be specified to include in the task address space the access to a library, or global variables.

---

<i> This implies also that the option bit 13 (logical address=physical address) must be specified in most of segment management directives.

<ii> Take care about the fact that if your parameter block (or a parameter pointed to by it) is outside the current address space of the calling (or destination) task, the call is rejected (without any error return in some cases).

### 2.2.2 DESEG : Deallocate a Segment

RMS68K deletes the specified segment from the target task's address space. If the permanent option is specified, the permanent status is removed if the segment is shareable. If the segment is currently not shared by any other tasks and its status is not permanent, the memory is added back to the free memory list.
A task cannot delete a segment if its current stack pointer (USP) points within the segment to be detached.
A task can detach another task's segment only if this task is in the "dormant" state.
This directive is assumed on all segments owned by the task when it terminates due to a TERM, ABORT or TERMT directive.

### 2.2.3 DCLSHR: Declare a Segment Shareable

RMS68K makes a segment available for use by other tasks. If globally sharing is specified in the segment attributes, the segment is available to all tasks. The segment can also be declared as a permanent segment with this directive, so that it is not released if no task remains attached to it.

### 2.2.4 ATTSEG: Attach a Shareable Segment

RMS68K adds the specified segment into the address space of the requesting task. The segment must be a globally or locally shareable segment. The segment remains in the address space of all other current sharers.

### 2.2.5 SHRSEG: Grant Shared Access to Another Task

RMS68K adds the specified segment into the address space of the specified task. The segment must be a shareable segment that remains in the address space of all other current sharers.

### 2.2.6 TRSEG : Transfer a Segment

RMS68K transfer the owning of a segment to the target task's address space.

### 2.2.7 RCVSA : Receive Segment Attributes

A description of the specified segment owned by the specified task is returned to the user buffer.

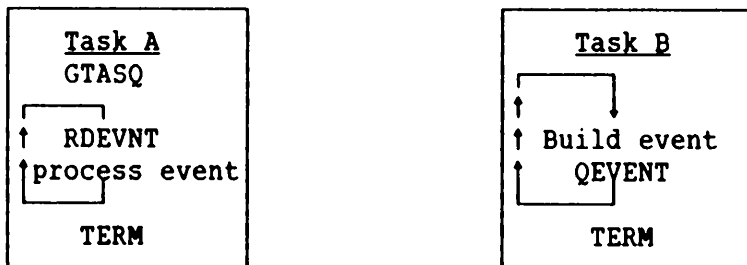
## 2.3 INTER-TASK COMMUNICATION

For inter-task communication, the standard facility offered by RMS68K is the event management.
The events are queued into the asynchronous service queue (ASQ) of the destination task.
The events can be serviced when they arrive in the ASQ by an asynchronous service routine (ASR) that interrupts the task's execution if enabled. <i>

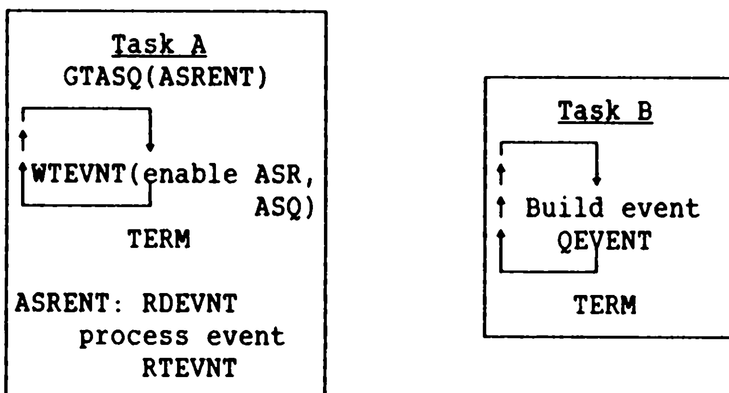
The following directives are available for inter-task communication :

- GTASQ : task allocates its ASQ and defines ASR normal entry point.
- DEASQ : task deallocates its ASQ.
- SETASQ : task changes the status of its ASQ.
- WTEVNT : task moves itself to "wait for event" until an event is placed in its ASQ.
- QEVNT : an event is placed in the ASQ of the target task.
- RDEVNT : task reads an event from its ASQ.
- RTEVNT : return from ASR.

Skeleton of events management without ASR:




Skeleton of events management with an ASR:




---

<i> This facility is not offered to a High level language programmer.

### 2.3.1 GTASQ : Allocate Asynchronous Queue

RMS68K allocates memory for the specified task's ASQ. The ASQ consists of a fixed length ASQ control block and the area for receiving messages, whose maximum length is specified.
This ASQ and an ASR (with its default entry point) can be enabled or not.

### 2.3.2 SETASQ: Set ASQ/ASR Status

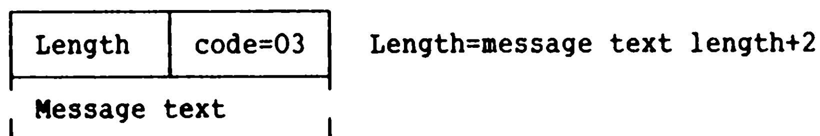
RMS68K replaces the requesting task's current ASQ/ASR status field with that specified above. If the new status indicates ASR enabled and there is an event in the ASQ, an ASR "interrupt" occurs (this is not a real interrupt).

### 2.3.3 DEASQ : Deallocate Asynchronous Queue

The memory allocated to the requestor's ASQ is freed. Any unserviced events in the ASQ are lost. No error code is returned.

### 2.3.4 QEVNT : Queue Event To Task

RMS68K places the specified event into the ASQ of the target task (if the ASQ is enabled). The specified event block must conform to the message event format as follows:




### 2.3.5 WTEVNT: Wait for Event

RMS68K ensures that the ASQ and ASR of the requesting task are enabled, and places the task in the "wait for event" state.
The next incoming event to the task's ASQ or the presence of an event already in the ASQ causes an ASR interrupt. When the ASR returns from event service, control returns to the location immediately following the WTEVNT directive.

### 2.3.6 RDEVNT: Read Event

RMS68K moves the oldest entry of the requesting task's ASQ into the area specified in the call.
If the event moved into the receiving area is a server task message (code 7) and the server task is to receive a parameter block from the task which requested the service, RMS68K moves the parameter block into the receiving area immediately following the event text.

### 2.3.7 RTEVNT: Return from Event Service

This directive must be used to exit from an ASR.
RMS68K restores all registers from the stack and returns control to the location at which point the ASR interrupt occurred. The ASR can specify whether or not it re-enables itself at exit.

## 2.4 TASK SYNCHRONISATION

An event is used for asynchronous interaction between tasks. Data is assumed and exists in the message field of an event. If no data is required, the receiving task can ignore the message field. The event is received by a task in its ASQ which is a part of system memory.

A semaphore is a common data structure used for exchange timing signals between concurrent processes. No data is assumed. If data is required, it can be implemented by a shared data segment which is in the address space of the interacting tasks.

RMS68K offers three types of semaphores to synchronise activities and control resources.

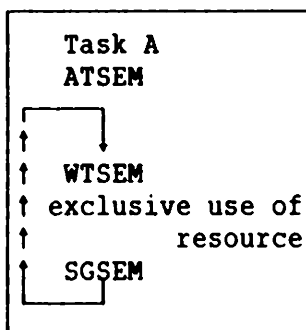
- type 1 :(boolean semaphore) is used when several tasks require exclusive acces to one resource.
- type 2 :is used to control the execution sequence of tasks.
- type 3 :(semaphore with count) is used when a task controls a resource witch another task wishes to use.

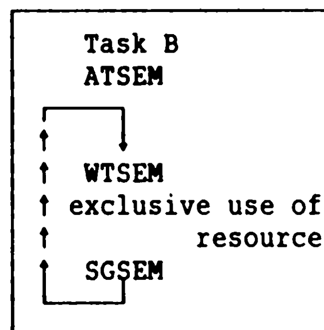
The following directives are available for semaphore management :

- ATSEM : Attach to semaphore
- CRSEM : Create a semaphore
- DESEM : Detach from a semaphore
- DESEMA : Detach from all semaphores
- WTSEM : Wait on a semaphore
- SGSEM : Signal semaphore

The Normal use of semaphores is the following :

Type 1 : implements mutually exclusive access to a single resource.




(necessary in all tasks just to know the KEY number)

Type 2 : Control the execution sequence of two or more tasks.

1) Upon entering a task, attach all semaphores (ATSEM) of tasks which are to be executed prior to this task.

2) If this task is to trigger the execution of other tasks, then create a semaphore (CRSEM) for this task.

3) Before proceeding with the execution of this task, wait on the semaphores (WTSEM) of the tasks which are attached in 1.

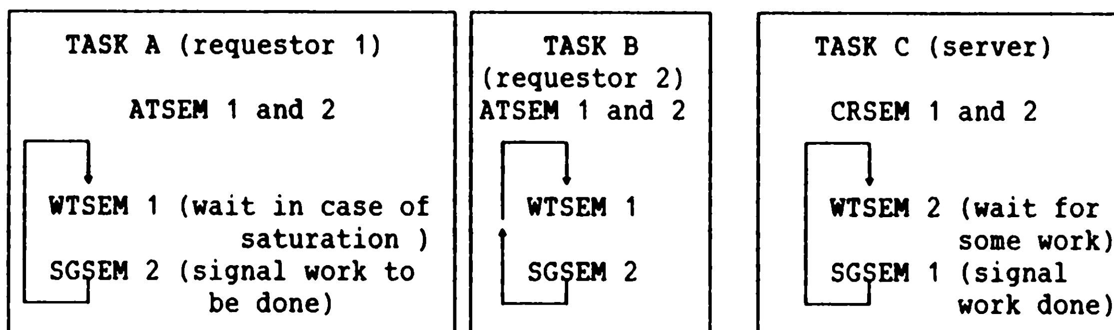
4) Proceed with the execution of this task until another task is to begin execution.

5) Signal this task semaphore. If the task is completed, detach from semaphore.

6) If the task is not completed, wait on the semaphore of the tasks which were attached in 1.

7) Return to 4.

Type 3 : task C has control over a resource which tasks B and A want to use. (the count specified is in fact the work for TASK C queue length)




### 2.4.1 ATSEM : Attach to Semaphore

RMS68K allows the requesting task to use the specified semaphore. If the semaphore already exists no action is taken, else the exact function is determined by the semaphore type:

- type 1 :the semaphore is created with an initial count of one.

- type 2 :the semaphore is created with an initial count of zero.

- type 3 :the requesting task is placed in a "wait on semaphore" state until CRSEM directive is issued by another task on the same semaphore.

### 2.4.2 CRSEM : Create a Semaphore

RMS68K creates or re-initializes the specified semaphore, and allows the requesting task to use it. The exact function of the directive is determined by the semaphore type:

- type 1 :idem as ATSEM
- type 2 :idem as for ATSEM except that an initial count is specified (#0)
- type 3 :if the semaphore already exists, the CRSEM directive is rejected else it is created with the specified initial count and all tasks in the "wait" state resulting of an ATSEM on this semaphore prior to its creation is reactivated.

### 2.4.3 DESEM : Detach from Semaphore

RMS68K detaches the requesting task from the specified semaphore. The task can no longer use that semaphore until an ATSEM is issued. The semaphore is physically removed from the system according to the semaphore type :

- type 1 :when the last user detaches from it.
- type 2 :when the last user detaches from it and the current signal count is equal to the intial count set by the CRSEM directive.
- type 3 :when the task that created it with a CRSEM directive detaches from it.

### 2.4.4 DESEMA: Detach all Semaphores

RMS68K detaches the requesting task from all semaphores to which it is attached. The rules are the same as for DESEM.

### 2.4.5 WTSEM : Wait on Semaphore

The current signal count of the specified semaphore is decremented by 1. If the count is negative, the requesting task is put in "wait for semaphore" and added to the waiting list for that semaphore.
If semaphore type 1, a check is made that WTSEM is issued before SGSEM directive.

### 2.4.6 SGSEM : Signal Semaphore

The current signal count of the specified semaphore is incremented by one. If the count is then zero or negative, the first task waiting for that semaphore is removed from the list and placed in the ready list to await execution. The requesting task continues executing (RMS68K does not enter its dispatch cycle).
If semaphore type 1, a check is made that SGSEM is issued after a WTSEM.

## 2.5 TIME AND DELAY MANAGEMENT

The SMACC owns a MM58176A Microprocessor Real Time Clock that delivers:

- an RT clock with constant frequency (with a minimum interval of 100ms) used by RMS68K to handle delays and to maintain an internal system date and time.
- a calendar including seconds, minutes, hours, days, and years (unused by RMS68K).
- a programmable delay interrupt (yet unused -- but which could be used by RMS68K for delay management if less than 100ms delays must be handled).

For time and delay management, the following directives are available:

- STDTIM : Set system date and time.
- GTDTIM : Get system date and time.
- DELAY : Wait for a delay.
- DELAYW : Wait for an event with a time-out.
- RQSTPA : Request for a periodic activation.

### 2.5.1 STDTIM: Set System Date and Time

RMS68K updates the system date and time. This service is available only for system tasks.

### 2.5.2 GTDTIM: Get System Date and Time

RMS68K returns the current system date and time into the return parameter block specified.
Date is given as the number of days since 01/01/1980.
Time is given as the number of milliseconds in the current day.

Note: The date and time returned by RMS68K are not read from the calendar chip, and thus are not updated when the SMACC is suspended.

### 2.5.3 DELAY : Delay Self

RMS68K delays the execution of the requesting task until the specified amount of time is elapsed, after which execution resumes at the location following the DELAY directive.

This directive does not affect the asynchronous event processing. If the ASR is enabled and an event comes into the ASQ, the delay is consideed to be satisfied and the event is processed.
The delay precision depends on the clock precision.
A zero delay cancels a periodic activation request (cf RQSTPA).

### 2.5.4 DELAYW: Delay and Wait

This directive functions as a combination of the DELAY, WTEVNT, and WAIT directives. If the calling task has an ASQ, RMS68K enables the calling task's ASR and ASQ. It puts the calling task into a "wait" state as a result of any one of the following occurences:

- The specified amount of time has elapsed.
- A WAKEUP is sent to the waiting task or the wakeup pending condition exists at the time the directive is called. The DELAY is cancelled.
- an asynchrounous event arrives or is already present in the caller's ASQ. Both DELAY and WAIT are cancelled. Control is given to the task at its ASR address and when the ASR returns, the execution resumes in sequence.

A zero delay cancels a periodic activation request (cf RQSTPA).

### 2.5.5 RQSTPA: Request Periodic Activation

RMS68K activates the specified task at an initial time and at optional intervals. The task can be activated in four ways, one of which is specified:

- RESUME : the task is activated by RESUME directive at the time that has no effect if the task is not in the "suspend" state.
- WAKEUP : the task is activated by WAKEUP directive at the time and a wakeup pending condition is set for the task if it is not in the "suspend" state.
- Timer event : the task is activated by queueing an event code 4 to the task's ASQ that is serviced either at the default ASR service address or at a specified alternative ASR service address.

If a request to activate a task has the same activation request ID as a previous request, the new parameters replace the old.
To cancel any periodic activation, option bit 10 must be set and an activation ID of zero must be specified.
If option bit 9 is set, an event is sent immediately to the specified task when a request is cancelled, rather than the next scheduled interval time. The activation count field has bit 15 set to 1 to identify it as a cancel event.

## 2.6 SPECIAL FUNCTION CONTROL

Some special features are available for task control:

- Monitor tasks : a monitor task can be set up to automatically receive notification of the termination of another task - referred to as a subtask of this monitor task.

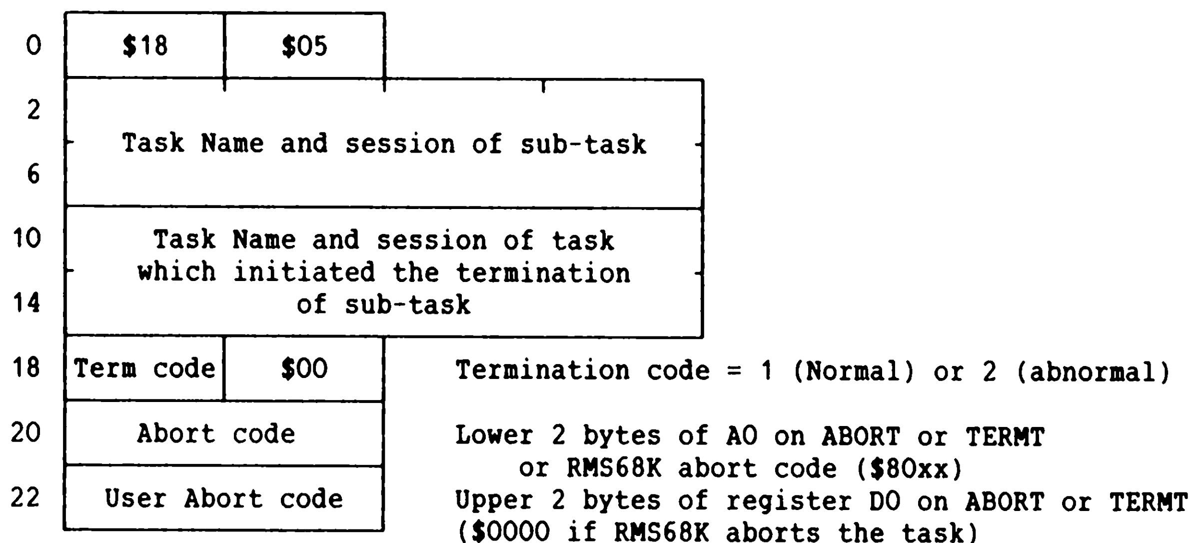
- Exception monitor task : a task can be established as an exception monitor task, which provides execution control over a target task (typically a debugger is an exception monitor task).

- Server tasks : A server task is able to receive and process request from any task in the system. Any task can request the services of a server task by executing a trap instruction. Thus a server task appears as a part of RMS68K.

- Local handling of traps and exceptions : A task can handle its own traps and exceptions.

### 2.6.1 MONITOR TASK

When a task is created (CRTCB) or started (START), a monitor task can be set up for this new task. This monitor task is noticed of the termination of this task by an event queued in its ASQ with following structure:



### 2.6.2 SERVER TASK CONTROL

A server task is a special task that responds to designated trap instructions executed by other tasks (not trap 0 and 1).

A server task specifies which trap instruction it recognizes. A request for services of a server task manifests itself to the server by an event. The server can then process that event synchronously or asynchronously.

A server task can set itself up to receive events on more than one trap instruction and/or each time a task terminates.

#### 2.6.2.1 SERVER: Establish Server

RMS68K establishes the requesting task as a server task of the trap instruction specified. Any task can then request the services of the server task by executing the appropriate trap intruction.

#### 2.6.2.2 AKRQST: Acknowledge Service Request

The receipt or completion of processing of a pending service request is acknowledged by placing the requesting task into an appropriate state (from the "server request pending" state). The acknowledging task need not to be the service task.

#### 2.6.2.3 DERQST: Set User/Server Request Status

RMS68K sets the enable/disable status of the server event entry mechanism according to the parameter.
A service request is made known to a server task trough an entry in the server task's entry. When a request caused by a given trap is entered into the ASQ, further insertion of requests of that type is automatically disabled. The server task can then decide when to re-enable receipt of that request type.
Any request issued when the ASQ is disabled for that request type is queued until its turn for insertion arises.
Unless the request receipt was disabled by the DERQST directive, request receipt for a given request type is automatically re-enabled when an acknowledgement is made for a request of that type.

#### 2.6.2.4 DSERVE: Deallocate Server functions

A server task initiates orderly shutdown of service. Any request still in the ASQ continues to be served by the task, but all new request will be treated as if the server had never existed.

### 2.6.3 EXCEPTION MONITOR TASK CONTROL

A task can be established as an exception monitor task, which provides execution control over a target task. Exception events of interest to the exception monitor task are specified by an exception monitor mask. When an exception event occurs, execution of the target task is halted, and an event is queued to exception monitor task, which indicates the target task identification and the exception event.

#### 2.6.3.1 EXMON : Attach Exception Monitor

RMS68K attaches the target task to the exception monitor task and places the target task in the "wait for command" state. An event with event code 8, indicating the attach, is queued to the ASQ of the exception monitor.
If the target task does not issue the directive, it must be in the "dormant" state.

#### 2.6.3.2 DEXMON: Detach Exception Monitor

RMS68K detaches the target task from its exception monitor. The target task then resumes normal activity according to its current state. A detach message is queued to the ASQ of the exception monitor tsk.

#### 2.6.3.3 EXMMSK: Set Exception Monitor Mask

An exception monitor mask is associated with a target task which is to be controlled by an exception monitor task. This mask specifies which exceptions are to cause the execution of the target task to cease and notification to be sent to the exception monitor.

Each bit of the mask corresponds to a particular exception. If a bit is set, the associated exception is relevant to the target task.

The bits and associated exceptions are listed below:

```
 bits 0 and 1 : must be 0
 bits 2 to 15 : trap 2 to trap 15
 bit 16       : bus error
 bit 17       : address error
 bit 18       : illegal instruction
 bit 19       : zero divide
 bit 20       : CHK instruction
 bit 21       : TRAPV
 bit 22       : privilege violation
 bit 23       : line 1010 emulator
 bit 24       : line 1111 emulator
 bits 25 to 26: reserved
 bits 27 to 31: used by RMS68K for execution control events
```

#### 2.6.3.4 RSTATE: Receive Task State

An exception monitor can receive the current state of a target task. This current state information includes the contents of all registers, the exception monitor mask, task state and execution control fields.

#### 2.6.3.5 PSTATE: Put Task State

An exception monitor can modify the state of a target task by changing the values of the target task's registers and exception monitor mask.

#### 2.6.3.6 REXMON: Run Task under Exception Monitor

An exception monitor task specifies how a target task is to be executed. There are four modes of operation which can be selected:

- Normal execution.

- Execute one instruction.

- Value change trace.

- Value equal trace.

### 2.6.4 LOCAL HANDLING OF TRAPS AND EXCEPTIONS

The normal handling of a trap or an exception results in standard to an abort of the task (except for trap 0 and trap 1 handled by RMS68K).
Some of the traps have been removed from RMS68K handling to solve particular problems:
- Trap 3 is used to mask all interrupts.
- Trap 4 is used to unmask interrupts.
- Trap 5 is used to do a single CAMAC action with a sure CAMAC QX response.

Some other traps and exceptions are handled by some servers, such as the datagram service (trap 6), or the debugger (line 1010 emulator).
It is possible for a task to handle those excpetion and not to go to the default server by the mean of the following directives.

#### 2.6.4.1 EXPVCT: Announce Exception Vectors

A task can handle its own exceptions. This directive is used to specify to RMS68K the nine possible exception handling routine entry points through the following address table :

| |
|---|
| bus error |
| address error |
| illegal instruction |
| zero divide |
| CHK instruction |
| TRAPV instruction |
| privilege violation |
| line 1010 emulator |
| line 1111 emulator |

A value zero in any table entry results in default processing of that exception.
Later the task can dynamically alter exception processing by swapping values in specific table entries without re-issuing an EXPVCT directive.

#### 2.6.4.2 TRPVCT: Announce Trap Vectors

A task can handle its own traps. This directive is used to specify to RMS68K the handling routine entry points for trap 2 to trap 15.

RMS68K uses the trap vector table (that consists of 14 four-bytes entries, each of which is the transfer address for TRAP #2 to TRAP #15) given as argument to handle trap instructions which occur during the execution of the issuing task. A zero value in any table entry results in default processing of the corresponding trap.
After the TRPVCT directive has been executed, the requestor can dynamically alter trap instruction processing by swapping values into the trap vector table.

## 2.7 INTERRUPT SERVICE MANAGEMENT

A task can include one or more Interrupt Service Routines (ISR) which are activated as the result of an external interrupt (or exception), and execute in the MC68000 user hardware state at the priority level of the interrupt. This mechanism is useful in creating device drivers.

The ISR code is a part of a task, totally shares its address space, and executes independently of this task. Therefore an ISR can run concurrently with the task.

When an external interrupt occurs, RMS68K save the current state of the processor and invokes the ISR. When the ISR is active, all task level activities are disabled, but the ISR can be interrupted in favor of an ISR with a higher priority level. Therefore, it is important that an ISR consumes a minimum amount of execution time to avoid system performance degradation and lost interrupts.

Upon exit, the ISR can reactivate the task in which it is included, for the purpose of processing results in background mode.

### 2.7.1 CISR : Connect Interrupt Service Routine

RMS68K offers the possibility for a task to handle some exception vectors with an Interrupt Service routine.

When the interrupt occurs, RMS68K enter the ISR quickly, <i> and give control to the ISR with register A0 containing the vector number in the low order 16 bits, and register A1 set equal to the value of the argument provided in the CISR directive. The contents of all other registers are completely unreliable.

### 2.7.2 RTE : return from interrupt

During ISR execution, only one RMS68K directive is allowed, which is the RTE directive, and thus all ISRs should be coded so as to ensure this and to ensure also that a minimum of time is lost at an interrupt level.
Three possibilities are offered to exit from an ISR:

- D0=0 : simple return from ISR.
- D0=1 : return from ISR and wakeup the task associated to the ISR.
- D0=2 : a 4 bytes message with event code 2 is queued into the associated task's ASQ and the corresponding ASR if any is executed at task level.

---

<i> Take care to TRACEFLAG SYSGEN parameter that can grow-up this time if trace is enabled

The normal sequence of an ISR code is the following:

```
ISRENTRY EQU     *         (Here A1 = parameter passed to CISR call
                                 A0 = vector number
                                 all other registers have any value => do not use
                                     JSR or BSR instructions without initiating
                                     A7 so that the stack is coherent.
         ....
         MOVEQ   #x,D0     (choose the exit mode)
         TRAP    #1
```

### 2.7.3 SINT : Simulate Interrupt

RMS68K activates the ISR as if the actual exception (or interrupt) had occured at the specified level.

## 2.8 TASK OR SYSTEM INFORMATION DIRECTIVES

The following primitives are provided by RMS68K, but i think that you will never use them except if you are writing some system information primitives.

### 2.8.1 TSKATTR : Task Attributes

The target task's user number and attributes are returned to the requestor.

### 2.8.2 MOVELL: Move from Logical Address

A block of data is moved from one logical address to another. A user task may only move data to other tasks within its own session and cannot move data to a system task's address space.
It is forbidden to move from an odd address to an even address or vice versa.

### 2.8.3 MOVEPL: Move from Physical Address

A block of data is copied from a physical address to a logical address within the destination task's address space.
It is forbidden to move from an odd address to an even address or vice versa.

### 2.8.4 TSKINFO: Return Copy of Task Control Block

A copy of the target task's TCB is moved to the requestor's buffer. The requesting task must be a system task.

### 2.8.5 SNAPTRAC: Snapshot of System Trace

The contents of the system trace table are copied into the buffer provided within the address space of the requesting task.

The pointers within the trace table to the next free entry and to the end of the table are adjusted to point to equivalent address within the requestor's buffer.

Refer to chapter 7 for a detailled description of the system trace table.

## 2.9 CDIR : Configure Directive

This directive provides a means by which new system directives can be created for use by specific applications and thus to add code running in the supervisor hardware mode.

## 2.10 Input/Output

To handle the asynchronous lines of the SMACC, the Channel Management Request service of RMS68K is not used.
Instead, the Monica Input Output system (MIOS) has been adapted to RMS68K in order to implement the Monica symbolic debugger.

Two communication are available on the SMACC front panel:
- the first one (RS232/current loop) is normally used by NODAL through MIOS and
  is dedicated to terminal communications and to MacIntosh file access.
- the second one (RS232/RS422) is planned to be used to implement a Local Area
  Network (Appletalk for example).

For the first implementation of the stand-alone MONICA concurrently with RMS68K, the first line is dedicated to MONICA and the second one is used by NODAL.

Since NODAL, a debugger or a LAN management is still using the different channels, access to standard connected display units should be made only through NODAL (or by a call to a subroutine internal to NODAL ).

In a first time, only the following primitive (with standard CERN interface) should be used by user programs to access to the terminal (and thus to the MacIntosh peripherals if connected):

```
PROCEDURE ALERT(RO Logical_unit:INTEGER;RO texte:STRING)
```

- The normal output device is logical_unit 1.
- The MacIntosh devices are accessed through logical_units from 128 to 255.

# 3 PS MEMORY LAYOUT ON THE SMACC

## 3.1 SMACC hardware requirements

The memory board of the SMACC consists of 6 columns of 8 chips.

The first column must be equiped with 64Kb RAM because all interrupts vectors are located at addresses $00000 to $00400 and a part of this column is hardware protected by an access in the user hardware state of the MC68000.
The four following columns can be equiped with either 64Kb RAM or with 128Kb EPROM.
The last column is normally equiped with 256Kb EPROM.

In a first time, only system is located on EPROM, what give us a maximum memory amount of 327Kb RAM and 262Kb EPROM. (For program development and testing purpose only, an extra memory card can be added if necessary).

The programming of different jumpers and of two PAL ensure the following:
- RAM is decoded as locations from $000000 to $04FFFF
- ROM is decoded as locations from $080000 to $0BFFFF
- Locations $000000 to $007000 are hardware protected against write from a task running in the user harware mode of the MC68000.
- The Restart vector can be fetched from location $080000 instead of location $00000.

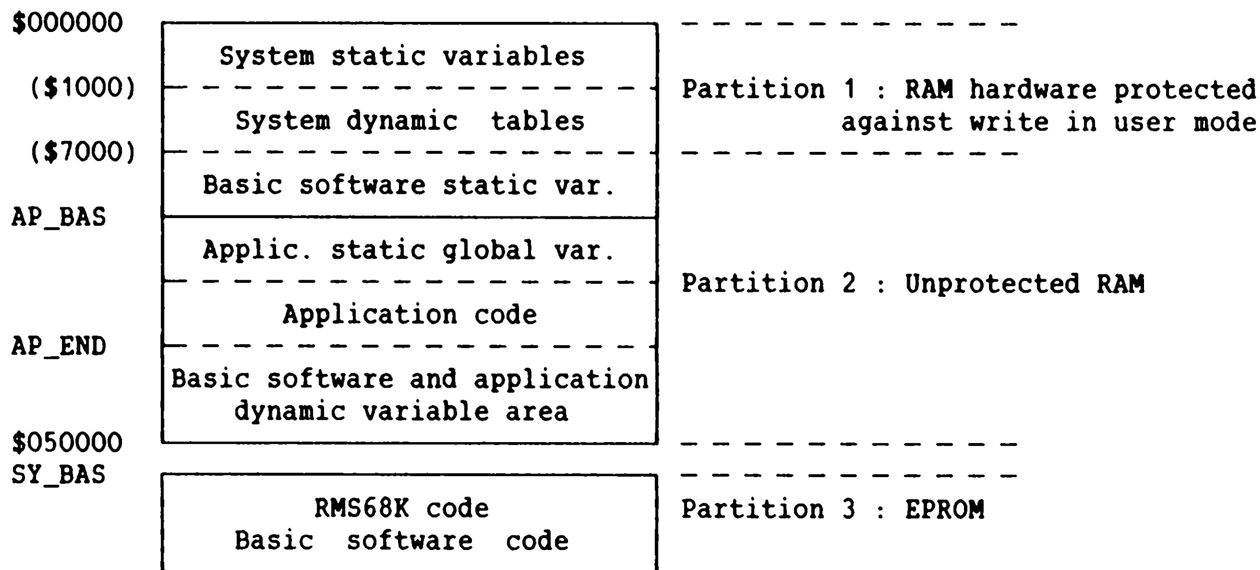
## 3.2 General memory partitioning from the RMS68K point of view.

The various constraints of the 68000 architecture, the SMACC hardware, and RMS68K have led to the following physical memory layout for the SMACC. Memory is divided for RMS68K Memory Management into the following partitions:

- RAM protected area to receive RMS68K data.
- RAM unprotected area to receive all application program and data.
- ROM area to receive invariant code.

[- RAM extension if present.]

The following diagram shows the partition layout:




This memory partitioning must remain as static as possible because it must be known at initialization time from RMS68K and cannot be changed after this time.

### 3.2.1 Partition 0: supervisor data

This memory area consists of RAM used for system data, hardware-protected against write access except in supervisor mode.
It starts at address $00000 to accomodate :

- exception vectors
- RMS68K system parameters
- supervisor stack
- crash save area
- RMS68K's dynamically created tables

The supervisor data memory partition uses 29Kb.

### 3.2.2 Partition 1 : RAM unprotected area

This memory area consists of unprotected RAM and is used by all software running in user mode that require RAM.

This partition is shared between resident basic software and application software as follow:
The begin of this partition is used by all basic software requiring a static memory reservation such as MIOS or Remote procedure call (for the interface with the FEC).
Then a gap is left for further evolution of the basic software static reservations.
Then we will find application global data, and then the application code.

The rest of this partition is under the control of RMS68K for dynamic allocation purpose (take care that this area is reset to zero at start-up) and must not be used without allocating by the GTSEG directive.
The main utilization of this dynamic area will be the following:
- NODAL working area.
- Remote servers working area.
- Datagram service buffer area.
- application tasks stack area.

### 3.2.3 Partition 2: system sofware code

This memory area consists of EPROM starting at address SY_BAS ($80000) and used for pure code and constants (NO variables) of basic software.
The maximum amount of memory that can be installed here is 256Kb EPROM.

The very first item in this partition is the RESET vector, followed immediately by a System Jump Table. <i> This table consists of a fixed sequence of instructions of the form :

```
XYZ:    JMP.L _XYZ
```

Thus application code wishing to call subroutine _XYZ (whose address may vary according to the system version) call in fact XYZ (whose address is constant). This allows an applications package to be loaded into a SMACC without knowing which version of the basic software is installed, as long as it contains all necessary subroutines. Unused entries in the Jump Table jump to an error routine.
This arrangement is insensitive to the RAM/EPROM configuration: either or both of the application code and system code may be in EPROM.
The full contents of Partition 4 is:

| Item | Size |
|---|---|
| RESET vector | |
| System Jump Table | |
| RMS68K kernel, reset code, initial task | (25Kb) |
| MIOS (MoniCa/RMS Input-Output system) | ( 7Kb) |
| Communications software: | |
| Low-level Datagram | ( 1Kb) |
| Datagram services | |
| NODAL | |
| kernel | (52Kb) |
| functions | (12Kb) |
| system functions | ( 3Kb) |
| System call library | ( 3Kb) |
| P+ library | |
| Total used : | 100Kb |
| [MoniCa symbolic debugger - optional] | (64Kb) |

<i> This system jump table is maintened by G.Cuisinier and if you need that some extra jumps to system routines, please contact him.

### 3.3 Cluster notion

The maintenance problem due to the big number of SMACC that will be installed in the PS control system has led to the choice of separating the program production in at least two independant Link-Edit and pushing called "Cluster": <i>

- the EPROM resident part (Sytem software constant part).
- the RAM loaded part (Application software part).

The only link between those 2 "cluster" must be done by a minimum of fixed "magic addresses" that must be fixed at link-edit time or found at start-up by the initialization routines. This will result that the only references to the EPROM part must be either through the trap handling facilities, either through jumps via the system jump table.

The loading of the SMACC will be done in two steps with the LDACC routine:

- loading of basic software cluster (unless system is in EPROM)
- loading of the application software cluster.

### 3.4 The memory environment of a task

When running, a task owns at least the following segments:

a) The library code segment "LIBR" containing all basic software located in partition 2 (EPROM).

b) The global variable segment "PROG" including basic software global variables, Application software global variables and application code.

c) Task's local variable segment (named with task's name) where we find:
   - The current stack (pointed to by A7).
   - The variables of the current routine (pointed to by A6).
   - The variables global to this task (pointed to by A5).

d) Task's code segment if dynamic loading is provided.

The allocation of an ASQ and of other dynamic memory is under the responsibility of the task.

---

<i> In fact this number was planned to be bigger than 2 to allow a partial reload of the SMACC without any stop but this seems not to be possible in a first time.

# 4 START-UP OF THE SMACC

The system can be stopped and then restarted at any moment from the FEC by the use of some CAMAC functions.

The start-up of the SMACC is divided into several steps:
- Hardware Start-up.
- RMS68K initialization and start-up.
- Basic software Start-up.
- Application software Start-up.

## 4.1 Hardware start-up

Power-up, Z or F28.A0 cause an external hardware reset.

The reset provides the highest exception level and is designed for system initialization and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state (and the trace state is forced off).
The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector decoded as location SY_BAS (or $00000 according to the RESTART jumper).
Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register are saved. The reset exception vector contains the initial supervisor stack pointer (SSP) and the initial program counter (PC). Finally, instruction execution is started at the address in the program counter. The power-up/restart code must be pointed by the initial program counter.

The reset signal resets all peripheral circuitry, i.e. CAMAC address counter, CIR, COR,...

Notes:
- In some cases (trap during trap handling) the microprocessor can hang in halt and the only way to restart it is to force a hardware reset.
- The AFTER-RESET jumper selects if, after a reset, the microprocessor continues executing (position CONT), or first goes in the suspend state (position SUSP). The latter possibility has to be used if the reset vector can only be loaded after reset. After loading the program, the microprocessor is put to its executing state by F25.A0.
- The RESET instruction does not cause loading of the reset vector, but does assert the reset line to external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

## 4.2 RMS68K start-up

The RMS68K start-up is divided in two parts :
- Pre-initializer
- System initializer

### 4.2.1 RMS68K Pre-initializer

The vector address SY_BAS branch in the supervisor hardware state to the RMS68K system pre-initializer whose work is to initialize application dependant system initializer parameters:

- It make a copy from EPROM to RAM of all system parameters that must dynamically adjusted before start-up.

- It finds at location $400 a pointer to a Partition description vector (PDV) described as following:

```
        PDV  DC.L   AP_END
             DC.L   0,0,0,0,0,0,0,0,0
             DC.L   TASK_AD
             DC.L   NODAL_C
```

  AP_END is the first free address for dynamic memory management of partition 1 and the end of loaded code.
  NODAL_C is the pointer to the chain of all application defined NODAL headers.
  TASK_AD is the pointer to the task description blocks used by the Application initialization module.
  This PDV is used to initiate the memory partitionning.

- It then clears and disables all possible LAMs in the crate (except for station 18 that should be the SMACC itself).

- It jumps to the RMS68K initializer.

### 4.2.2 RMS68K System Initializer

The system initializer can then initiate all non-SYSGEN fixed parameters <i> by doing the following work :

- It loads system stack pointer from SYSPAR.

- It clears memory partition 0 through the end of SYSPAR

- It initializes exception vectors.

- It constructs the list of memory free for dynamic management and reset it to zero (except for statically reserved area).

- It initializes system parameters and tables used by RMS68K.

---

<i> refer to appendix for SYSGEN parameters

- It initializes serial ports

- It makes resident the following tasks and places them in the ready list :
    - MIOS : MIOS Main task
    - GOGO : basic software initialization task

- It goes to RMS68K exec.

Then the system starts the first task into the ready list.

### 4.2.3 Basic software initialization

The basic software initialization task "GOGO" is the very first task to be dispatched by RMS68K.

- It gets a data segment for its own stack.

- It gets global segment "PROG" containing all static global variables and the application code and then declare this segment shareable.

- It gets global segment "LIBR" containing all basic software code and then declare this segment shareable.

- It connects timer ISR to exception vector 30.

- It enables timer's 100ms interrupts.

- It updates the system date and time from the MM58176A chip current date.

- It calls the application initialization routine which does the following jobs:

    - Initializes the Initialized Global Variables. (This is done by copying the INIT_DATA section into the INIT_VAR section).

    - Creates a Task Control Block (CRTCB) for every Process described in the TASK_INIT_BLOCK section.

    - Give the owning of the global segments ( PROG and LIBR) to those task.

    - Start every task described (attaching them to an exception monitor if specified).

- Then it starts the NODAL monitor task NODL.

- Then it put itself into the "wait" state.

The "NODL" task starts the different servers and the interactive NODAL ("NODI") and acts as monitor task for all those system tasks.
In case of abnormal terminaison of one of those tasks, it restarts it and write an error message on the connected terminal - if there is one. <i>

---

<i> Refer to chapter 6 - logging errors.

### 4.2.4 Application software initilization

Once the basic software is running, the application task with the highest priority can then provide an extra initialization according to application software requirements.

The following problems are not yet solved:

1) Delay management.
   The delay precision is yet 100ms (the minimum value that can be genarated by the clock circuit without any complicated binary->BCD date and time conversion for delay handling). This value could be shorter with a new clock delays management software.

2) Short time power-fails.
   There is no different management between short-time power-fails and an initial memory loading : the software initialization does the full reinitialization in each case.

3) Dynamic memory loading
   The dynamic loading of a task can yet be provided only for a single task written with the restriction that its code is position independant which is not the case except for special assembly-written tasks.

### 4.2.5 Exchange of magic memory addresses between FEC and SMACC

The FEC needs to know several magic addresses in the SMACC for different purpose, but a minimum of system dependant addresses must remain in the FEC.

The SMACC software is split in two parts :
- system independant part (because EPROM resident)
- application dependant part.

Locations from $80000 are reserved for a system jump table (maintened by G.Cuisinier and addresses refering to the system independant part (for example address of some internal to RMS68K tables) must be found by the FEC in that table.

Some application part dependant addresses are found from the PDV, whose address is found at magic address $00400 (for example, the end of statically reserved memory, the begin of the application NODAL headers and the list of task).

For application, another fixed location ($00404 by example) will contain a pointer to a table maintened by application developpers, in which at a constant displacement, the FEC NODAL or P+ programs will find the pointers they require to access to information.

References to application dependant part must be found through the PDV and no absolute magic address must be used in the FEC programs to refer to SMACC addresses.

For example, the chain of NODAL headers is found in memory as follow:
- read the address of the PDV:
```
GETBL(smacc.,[[400,0,tb,2,cc)
```
- read the 12th address in the PDV :
```
GETBL(smacc.,tb(2)+44,tb(1),tb,2,cc)
```
- now (tb(1),tb(2)) is the address inside the smacc for what we were looking.

# 5 STANDARD FACILITIES ON THE SMACC

The following facilities for operational software are available:
- Operating system (plus system initialisation and input/output driver)
- Elementary communication with host minicomputer
- General library
- Interactive Nodal
- CAMAC/LAM access

Macintosh personal computer link:
- not discussed in this document.

The following facilities are in preparation:
- P+ environment (plus MoniCa symbolic debugger)
- Datagram service (simple message passing)
- Remote call services

The following facilities will be provided later:
- Remote file access
- Dynamic task loading

The CERN convention for programming the MC68000 must be used in all programming.

## 5.1 OPERATING SYSTEM, LANGUAGES, AND LIBRARIES

### 5.1.1 Operating System

The operating system RMS68K is available to run in the SMACC with its full set of directives.

CAMAC interrupts (LAMs) and front-panel interrupts do not require special handlers: they can be treated by user-written interrupt service routines as allowed by RMS68K.

Applications may be built of task segments (several tasks on one segment), library segments (many subroutines on one segment), and data segments. Any task or subroutine may call any library segment: virtual addresses are unique throughout user address space.

Tasks are differentiated by priority, with higher priority reserved for short, urgent tasks. Thus the CPU is shared as a result of tasks voluntarily entering wait states for external interrupts, the clock, or input-output. In consequence, to prevent abuse, interactive Nodal must have the lowest priority.

A dynamic down-line loader may be provided later to load a new application task into a running system (This is intended primarily for tests).

### 5.1.2 NODAL-68K

Nodal has been adapted as follows:

1) Input/Output:

   - Terminal I/O uses MIOS. (ODEV=1)

   - the acces to the different peripheral of a MacIntosh personal computer (connected as terminal) and the use its files and line-printer are possible
     (syntax : OLD file-name or SET ODEV=OPEN("W",".PRINTER")).

   - a 64Kwords CAMAC File Module can be used for local file support.
     (syntax : SAVE <n>file-name where n=CAMAC station number)

   - the access to the FEC files will be provided later.
     (syntax : OLD <computer>(user)file-name )

2) New data type 32 bits integer has been added (DIM-LONG A(20)).

3) CERN standard 68K calling sequences are supported (new function types 28, 29, 30 and 31).

4) The edition commands have been extended (lower-case commands are accepted).

5) IMEX/EXEC will be supported.

6) New NODAL functions headers are automatically generated by P+ compiler if compatible with NODAL interface and those new functions are directly callable from NODAL after startup.

7) It is not possible to use compiled LDEF functions. <i>

8) It is possible to start an event-driven or a scheduled non-interactive NODAL for low-priority background activity.

### 5.1.3 Nodal functions to access RMS68K

Nodal is used as an interactive command interpreter for RMS68K: a special module has been added to Nodal for this purpose with interactive commands beginning with "@".

It is possible to run Nodal at extra high priority when it is necessary to abort a looping program during debugging with @SET-PRIORITY "NODI" xx command.

---

<i> A Nodal compiler for MC68000 has been developped at SPS but is uncompatible with our NODAL.

### 5.1.4 Real arithmetic routines

Although the basic software (compiler, code generator, library and interpreter) is made as independent of real-number format as possible, the ND 48-bit floating point format is used. The Nodal-68K software package emulating this format is used.

### 5.1.5 General library routines

These are specified and implemented as the need arises, including:

1) RMS68K system calls <i>

2) Input/Output from a user task (for debug or error handling only): It is yet only possible to use the ALERT primitive for print-out of a string.

3) CAMAC access (SCAM)

4) Datagram service primitives (see below)

5) "Context" routines such as

   - USERC to obtain network user code

   - FROMC to obtain source of remote call

   - LOCAL to obtain local computer identifier

For efficiency, CAMAC access don't use SCAM, but uses MOVE instructions to and from memory-mapped CAMAC. In P+, this effect is achieved by declaring variables equivalenced to memory-mapped CAMAC by ENTRY statements <ii> .

There is no 'process manipulation' or 'global variable' package since task and message facilities are provided by RMS68K and the communications software.

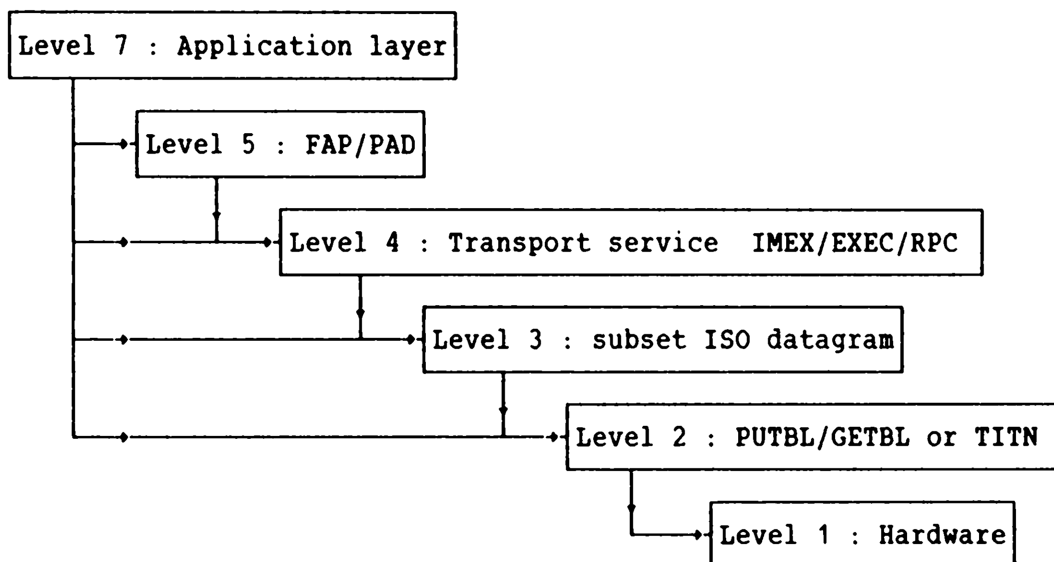
---

<i> Refer to appendix P-PLUS interface to RMS68K and Nodal interface to RMS68K.

<ii> Refer to appendix ACCESS to CAMAC.

## 5.2 COMMUNICATIONS SOFTWARE

Communications software for the SMACC follows the layered philosophy outlined in the diagram below. The layers correspond to those of the OSI Model to facilitate integration of the emerging international protocol standards, but there is no claim to conform to any standards.




RPC= Remote Procedure Call
FAP= File Access Protocol
PAD= Packet Assembler/Disassembler (remote terminal protocol)

SMACC software includes:

Level 2: PUTBL/GETBL/TRACC/STACC and LDACC
Level 3: CERN datagram primitives

High level: Remote Procedure Call (FEC calls SMACC)
IMEX/EXEC protocol (FEC calls SMACC)

The following facilities are not yet provided :

High level: File access protocol (SMACC calls FEC)
Remote Procedure Call (SMACC calls FEC)

### 5.2.1 Level 2 primitives available on the FEC

The following procedure declarations apply to the SMACC. SMACCs are identified by an integer (currently in the range 0..59).

```
PROCEDURE STACC (RO smacc_id      : INTEGER;
                 RO bit_number    : INTEGER;
```

```
                WO completion_code: INTEGER );
```

Returns in the completion code the current value of the indicated software status bit in the specified SMACC and clears it. Guaranteed to be an indivisible operation. Bits 0 to 5 are free for applications, and may be set by appropriate MOVE instructions (see SMACC hardware user's manual; in P+, variables are used as for CAMAC access).

```
PROCEDURE TRACC (RO smacc_id        : INTEGER;
                 RO bit_number      : INTEGER;
                 WO completion_code: INTEGER );
```

Sets the indicated software trigger bit in the specified SMACC. Guaranteed to be an indivisble operation. Bits 0 to 5 are free for applications, and their occurrence is indicated by a 'signal' operation on the corresponding type 2 semaphore SEM0 to SEM5.

```
PROCEDURE PUTBL (RO smacc_id        : INTEGER;
                 RO address_low     : INTEGER;
                 RO address_high    : INTEGER;
                 RO dummy           : INTEGER;
                 RO user_buffer     : ROW [lo..hi:INTEGER] OF INTEGER;
                 RO word_count      : INTEGER;
                 WO completion_code: INTEGER );

PROCEDURE GETBL (RO smacc_id        : INTEGER;
                 RO address_low     : INTEGER;
                 RO address_high    : INTEGER;
                 WO dummy           : INTEGER;
                 WO user_buffer     : ROW [LO..HI:INTEGER] OF INTEGER;
                 RO word_count      : INTEGER;
                 WO completion_code: INTEGER );
```

These write and read 16-bit words to/from a SMACC memory zone from/to the given buffer in the user program. 'address_low' is the bottom two bytes and 'address_high' is the high byte of the address of the buffer in the SMACC. The SMACC address must be even.

```
PROCEDURE LDACC (RO smacc_id        : INTEGER;
                 RO image_file_name: STRING;
                 WO completion_code: INTEGER );
```

This down-loads the specified ACC from the specified image file. As mentioned in the chapter on the memory layout, LDACC loads the basic software image from a standard file (unless it is in EPROM) before loading the specified applications image.

The following calls are used in NODAL to administer the ACC definition table:

```
DFACC(N,L,C,S,CO)    Defines ACC number N in loop L, crate C, station S
                     (CO=0 means OK)
                     If top bit of N=1, the ACC is a SMACC.

DFACC(N,0,0,0,CO)    Clears ACC number N from table

DFACC(-1,L,C,S,N)    Finds number N of ACC in loop L, crate C, station S
```

(N=-1 means no such ACC)
If top bit of N=1, the ACC is a SMACC.

ADACC(N,L,C,S) Returns loop, crate & station of ACC number N

In the ACC definition table, the top bit of the table entry is set to indicate a SMACC. The next-to-top bit is used to indicated that the basic software is in EPROM in an individual SMACC.

### 5.2.2 Datagram service (not yet implemented)

The datagram service provides the transport of complete, self-contained messages of limited size, known as datagrams, between two user programs. An individual datagram is delivered complete and correct or not at all, but there is no mathematical guarantee of delivery or sequence.

The full set of calls and their semantics are defined in the DG2 interim report (CERN/DD/KIK-X/R1) and are not repeated here.

For future compatibility, all fields of the 'internet_address' exist. 'ISO_code' and 'network_number' are fixed, 'station_number' is the SMACC number, and 'socket' is application-dependent. All seven of the CERN standard datagram access primitives are implemented:

| | |
|---|---|
| dg_open | associates a user program with a socket number |
| dg_set | sets up characteristics of a socket |
| dg_close | cancels 'dg_open' |
| dg_send | send a datagram |
| dg_receive | receive a datagram |
| dg_wait | wait for datagram(s) |
| dg_info | obtain information about socket |

In the SMACC, there are no problems of layout. The 'dg_' routines are implemented as P+ routines referenced through the general basic software jump table.

In the FECs, the 'dg_' routines implemented as ICCI procedures. Only SRT programs and ICCI procedures can access the datagram service: HRT programs cannot do so.

TRACC/STACC/PUTBL/GETBL are used to provide a line protocol, with appropriate enhancements to give a 'send packet' and 'get packet' interface for use by the datagram service itself.

The packet header format is as follow :

| | |
|---|---|
| Segment length | 2 bytes |
| Header checksum | 2 bytes |
| Dest addr | 8 bytes |
| Srce addr | 8 bytes |
| Segment offset | 2 bytes |
| Total length | 2 bytes |
| | |
| TOTAL | 24 bytes |

The first version does not switch packets between SMACCs or forward them through TITN to other FECs, but the use of full addresses allows this in the future.

### 5.2.3 Remote Procedure Call (being implemented)

The P+ programmer in the FEC can issue a remote call to any SMACC connected to that FEC.
SMACC identifiers are integers of type COMPUTER.

- TITN network FECs have computer number less than 256.
- SMACCs have numbers of the form : N+256*FEC_number.

The code produced by the P+ (Version A) compiler for an RPC is unchanged. This also means that NORD-PL code in the FEC can execute RPC to a SMACC by following today's COOKBOOK.
The run time code issues a standard P+ call to 1REM with the A-register pointing to a remote call descriptor. The NORD-PL format of the descriptor is as follows:

```
INTEGER ARRAY <remca> := (#AB,#CD,#EF, <computer_number>,
                          <tag>,<param_ref>, <tag'>,<param_ref'>,...,ENDPA);

where            <remca> is an identifier chosen by the user
                "ABCDEF" is the name of the procedure to call remotely
                                (filled with zero if necessary)
      <computer_number> is the remote computer number
                  <tag> is the tag for the following parameter
            <param_ref> is the address of an actual parameter
                           (address of descriptor for STRING parameters)
and    <tag> ::= <acces> + <type>
where <acces> ::= RO ! WO ! RW
       <type> ::= REMI ! REMR ! REMS ! REMRI ! REMRR
```

Alternatively, <tag> is defined by:

```
      CASE INTEGER OF 0 : RO INTEGER;
                      1 : RO REAL;
                      2 : RO STRING;
                      3 : RO ROW [INTEGER] OF INTEGER;
                      4 : RO ROW [INTEGER] OF REAL;
                      8 : WO INTEGER;
                      9 : WO REAL;
                     10 : WO STRING;
                     11 : WO ROW [INTEGER] OF INTEGER;
                     12 : WO ROW [INTEGER] OF REAL;
                     16 : RW INTEGER;
                     17 : RW REAL;
                     18 : RW STRING;
                     19 : RW ROW [INTEGER] OF INTEGER;
                     20 : RW ROW [INTEGER] OF REAL;
            END CASE;
```

If the tag indicates a ROW, it is exceptionally followed in the descriptor by the array size (number of elements).
If the tag is 63 (77 octal), it is a dummy and there is no parameter.
With the restriction that only 16-bit integers can be used in remote calls to a SMACC, the above descriptor remains unchanged. Note that record types are tagged as integer arrays, and variants can be sent as 2 ordinary parameters.

RPC datagrams are in the format:

```
 <call_datagram>::= <header><usercode><6_byte_name> {<tag>[<parameter>]} ENDPA
<reply_datagram>::= <header>    <error_code>       {<tag>[<parameter>]} ENDPA

     <header>::= datagram header as above
   <usercode>::= 32-bit network usercode
<6_byte_name>::= remote procedure name in ASCII, null-filled
 <error_code>::= 32-bit error code for RPC protocol
        <tag>::= as above
  <parameter>::= <row_size> [<array>] | <string_descriptor> [<string_buffer>] |
              <16_bit_integer_value> | <48_bit_real value>
```

The tags for all parameters are transmitted in both directions. The parameters are transmitted only if necessary, i.e.:

```
RO parameters: only in call datagram
RW parameters: always
WO parameters: in reply datagram: all transmitted
               in call  datagram: only row size & string descriptor
```

The call datagram is sent to an agreed socket number in the SMACC. The reply datagram is returned to the source address of the call datagram. 1REM reports failure either on time-out or when it receives a non-zero RPC error code.

The RPC server in the SMACC is a permanently active program listening to the agreed socket number. It hold a table of known remote procedures and their entry-point addresses. Its algorithm is (without error cases):

```
dg_open   {attach to socket number}

LOOP dg_wait   {until a datagram arrives}
     dg_receive
     look up procedure name & address
     build actual parameter list
     call procedure
     build reply
     dg_send
END LOOP
```

Note: To start several RPC at the same time in different SMACCs the best way is to make a first RPC that starts a remote process and a second RPC later to read the results of this process.

### 5.2.4 IMEX/EXEC (not yet implemented)

The general scheme is similar to that for RPC. Message formats are specific to NODAL and are not detailed here.

# 6 SPECIAL FACILITIES ON THE SMACC

## 6.1 Camac LAM and front-pannel interrupts handling

The SMACC is intended to handle a CAMAC hardware with hard real-time constraints and has to respond to many application dependant interrupts.

Except for timer, Serial control interface and communication registers, the basic software offers a default handling of interrupts that only clear them.

The normal way for the applications to handle those external interrupts is to write (in assembly language) an Interrupt Service Routine to service them.
The treatement done at interrupt level should be as quick as possible while during this time all other LAMs are masked and also the communications with the FEC.

Notice that an ISR is always under the control of a task and that the connexion disappears whith the task.

The normal skeleton of a task controlling one or more ISR is the following:

- allocation of an ASQ (GTASQ) to be able to receive end of ISR code 2 messages and to be noticed of any exception occuring while ISR execution.
- connexion of the ISR code (CISR) to the corresponding vector <i>
- wait for the event.
- process the event
- return in the wait for event state - except in case of exception.

## 6.2 Error handling

Several kinds of errors can be detected during execution :

- Software predicted errors: at this time - especially for debugging - the programmer want to post a text to signal an error condition (or a normal completion).
- Software fatal errors : the programmer can have not predicted some conditions that even occurs and wants to ABORT its own task to signal this condition.
- Hardware fatal errors : some operations can produce an hardware error during the execution of a task : for example a "bus error" exception can occurs if a read is made for a write camac function.

---

<i> Refer to appendix Absolute variables for the SMACC

### 6.2.1 Monitor task

An exception monitor task gets control over the execution of any task it controls. In particular, such a task is woken up on any fail in a task and is able to logg this to the terminal.

Frank di Maio did develop a basic exception monitoring task that can yet be used to get a minimum of informations when a task does crash.

### 6.2.2 Logging errors on terminal

In any case it is useful to log all errors on the local terminal - if connected - for maintenance purpose: the link with the FEC can be broken or non-existant and at this time local messages must be displayed on the local terminal.

### 6.2.3 Logging errors to the FEC

To be able see the connexion between errors detected and the general environment, and also to keep a paper trace of all detected errors, some errors (or perhaps all) have to be logged on the FEC error device. <i>

## 6.3 Error number convention

RMS68K error codes <ii> are in the form $08XX00YY where XX is the directive number and YY a general mean error code.

RMS68K abort codes are in the form $80XX, where XX is an exception numbered from 0 to 31. <iii>

MONICA, stack handling and pushing errors are handled via Axxx instructions.

The result of the SMACC integration meetings have hold to choose the following error numbering convention:

- 2 bits for severity level
- 4 bits for error type (type 0 reserved for NODAL)
- 10 bits for error number in this type
- 16 bits to precise error source.

G.Cuisinier holds the list of the error codes and will develop a FLIRT function to transform RMS error code to PS error code.

---

<i> An error message primitive must be provided for end-users,

<ii> Refer to appendix RMS68K ERROR CODES.

<iii> Refer to appendix RMS68K EVENTS STRUCTURE- Exception monitor events

### 6.4 Power-fail interrupt

In case of power fail, the SMACC is able to continue its execution after power-restart without reloading it because its RAM has a power-fail supply battery. <i>

The internal RMS68K clock is not updated during the power-fail but the MM57167A internal calendar is updated during this time.

No standard power-fail/power-restart handling is done: the system is yet only restarted.
This results in the following actions:

1) current programs running under NODAL (interactive or not) are lost.

2) dynamically allocated memory is given back to RMS68K and is cleared.

3) statically initialized data are set back to their initial value.

4) current situation is mainly lost.

---

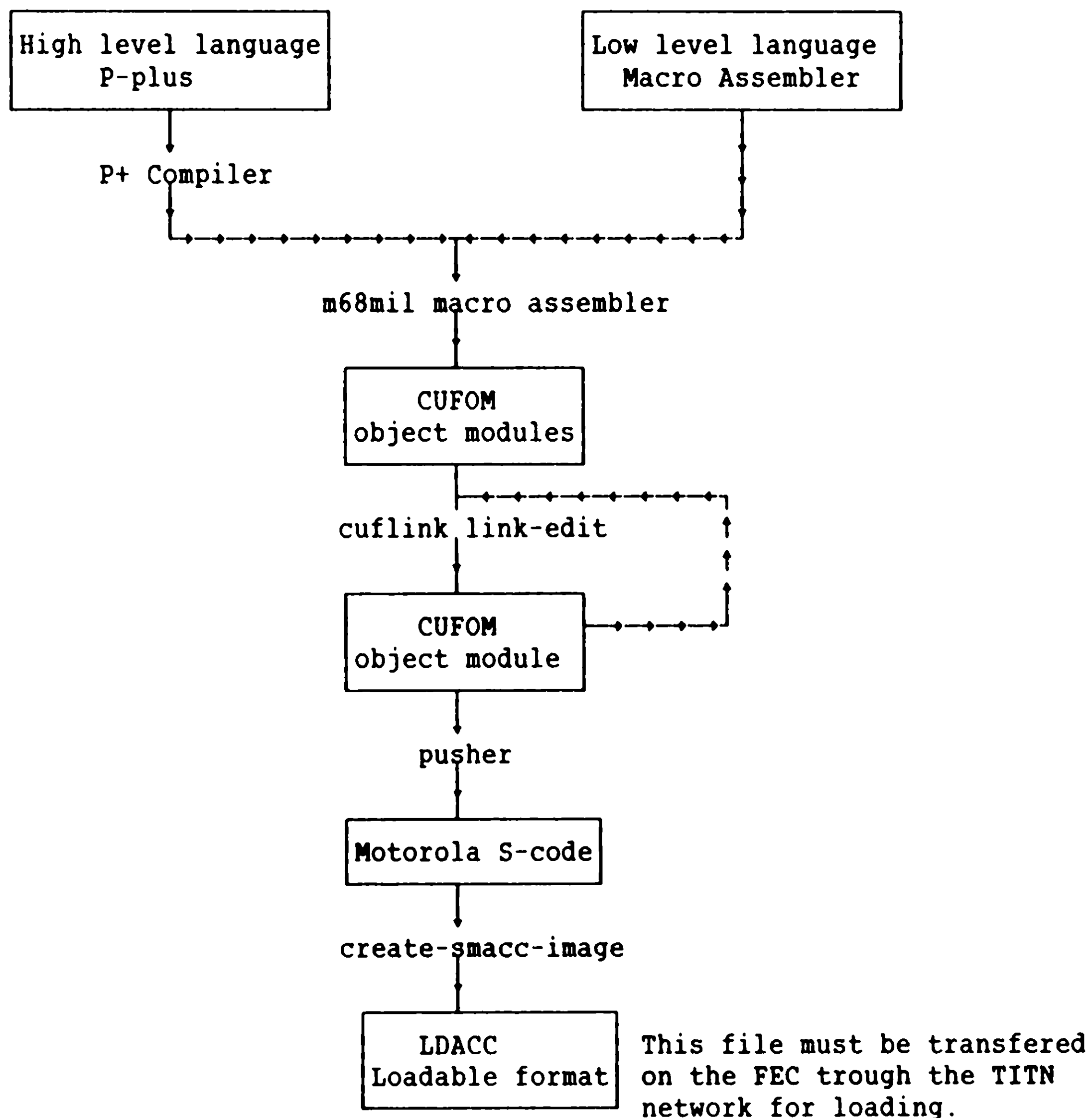
<i> the low consumption allows up to 4 years!

## 7 PROGRAM PRODUCTION FOR THE SMACC

The standard PS programming environment offers the following languages to program on the SMACC:

- Interpreted NODAL
- Compiled P+
- Macro-assembler M68MIL

Except for interpreted NODAL, program production is on the PRDEV computer following this diagram :



## 7.1 Software Architecture.

The SMACC memory is divided in 3 partitions:

- 0 : RMS68K reserved RAM (write protected from an user program).
- 1 : unprotected RAM
- 2 : Permanent software resident EPROM (or RAM in a first time).

The memory partitionning is done at system generation, but adjusted dynamically at system start-up for the end of the static unprotected RAM partition.

## 7.2 Existing program development tools.

The following software are available for program development :

- INCLUDER: utility to produce a new source file from another by replacing the $INCLUDE <file> lines by the contents of the specified line (include commands can be nested to a deapth of 10 files calling another file).

- P-PLUS : P+ compiler to produce M68MIL source code from P+ source files.

- M68MIL : Motorola 68000 assembler that produce CUF binary.

- PLANC-MC68 : PLANC compiler producing NRF code.

- NRF-CUFOM : utility translating NRF code in CUFOM binary format with the restriction to 2 sections (DATA and PROG) allowing the use of the PLANC-MC68 compiler.

- CUFMERG : utility to concatenate several CUF files in one CUF file.

- CUFLINK : link-editor producing CUF with a maximum of resolved external references.

- PREPUSH : utility to prepare the work for the pusher: all sections are evaluated in length and an implantation starting address is proposed to the user for each implantation section.

- UNIPUSH : utility to produce from the CUF an absolute memory image in the standard motorola S-code format (MSC).

- CREATE-IMAGE : utility to produce the loadable format from the S-code format.

- DATAIO-LOAD : utility to load the contents of EPROM into a DATAIO model 29A EPROM programmer and to produce also a loadable image from the non romable part of the application.

## 7.3 Program production procedures.

For the SMACC program production a set of XCOM procedures is available under M68K-CROSS-SOFT file space called as follow:

```
@XCOM (M68K) <file-name> <print-listing>
```

The following file naming convention is used for <file-name>:

```
xxx-yyyyyyyy:SYMB or  yyyyyyyy:Sxxx
```

where xxx = procedure to apply on this source file:

PPL = P+
M68 = M68MIL macro-assembler
PLC = Planc
TSK = List of CUF files link together to produce a new CUF with a maximum of resolved external references
IMA = List of CUF files to link together and to push to produce a SMACC loadable memory image.

### 7.3.1 Assembly language source files.

The documentation of the macro-assembler used is the CERN 83- 12 yellow report: M68MIL CROSS MACRO ASSEMBLER. (Horst von Eicken - DD - 22.12 .1983).
It is recommended to the user to read the chapter 5, that describes a set of general purpose macros, and to list the contents of thoses macros found in the file: (M68K-CROSS-SOFT)SYSTEM-MACROS:SYMB.
The rules used at CERN by the compilers for data generation and for stack handling conventions are described in the CERN 84-12 yellow report: CERN CONVENTION FOR PROGRAMMING THE MC68000 FAMILY (PS - 20.11.1984)

The name an assembly source file must be M68-xxxxxxxxxx:SYMB or xxxxxxxxxx:SM68. To assembly it, you just have to specify:

```
@XCOM (M68K)  <source> <list>
```

- The use of the INSERT pseudo instruction results in nothing because this facility is offert by the INCLUDE utility called in a systematic way to treat the following commands:
  $INCLUDE (<user>)<file>

- The use of the SYSTEXT pseudo-instruction results in the including of the file (M68K-CROSS-SOFT)SYSTEM-MACROS:SYMB.

- If more than 5000 lines are to be assembled, the switche /S1 (or /S2 if more than 10000 lines) must be specified as follow:
  @XCOM (M68K)/S1 <source> <list>

- The result of the assembly is a :CUF file.

The following sections must be used by the user to allow a further link-edit without any problem:

#### 7.3.1.1 code sections.

- TASK_INIT_BLOCK: Section receiving the task's descriptor used by the initialisation procedure to start all tasks after system start-up.
  Its contents is :

```
DC.L    '!INB'       Eye catcher
DC.L    0            link-list (not yet used)
DC.L    'xxxx'       Task Name
DC.L    1            Task Session
DC.L    xxxx         Entry point
DC.B    50           Initial priority  [0..255]
DC.B    100          Limit priority    [0..255]
DC.W    $8000        Task attributes (system task)
DC.W    0            Global segments map
DC.L    'ERLG'       Exception monitor name and session
DC.L    1
DC.L    $1FFFFFC     Exception monitor mask
DC.W    xx           Task code language (1=assembly,
                     2=Pascal, 4=P+, 6=C, 9=Planc) used by
                     error logging task.
```

- NODAL_HEADER : Section containing only the NODAL header part:
- INIT_DATA: Section receiving the initial values to be copied to the INIT_VAR section at restart time.
- CONSTANT_DATA: Section containing only Read-only constants.
- PROGRAM : Section receiving the program part.

#### 7.3.1.2 Static variables reservation sections.

- GLOBAL_VAR : Section containing static global variables without any initialisation (not reset to zero at restart time).
- INIT_VAR : Section containing static variables that will be reinitiated by the P+ initialisation program by copy of the contents of the INIT_DATA section.

### 7.3.2 P+ source files.

The name an P+ source file must be PPL-xxxx:SYMB or xxxx:SPPL. To compile it, you just have to specify:

```
@XCOM (M68K) <source> <list>
```

The P+ compiler used is the new P+ version B generating code for the MC68000.

### 7.3.3 Planc source files.

The name of a PLANC source file must be in the form PLC-xxxx:SYMB or xxxx:SPLC. To compile it and to produce CUFOM, you just have to specify:

```
@XCOM (M68K) <source> <list>
```

The PLANC-MC68 compile is able to produce code on 2 separate sections: data and program. The NRF-CUFOM translater transform the NRF and produce code on 2 sections: PROGRAM and INIT_VAR.

### 7.3.4 First link-edit.

A lot of :CUF files can be link-edited in one single resulting :CUF file to reduce EXTERN/ENTRY conflicts with the rest of the system. You are allowed to do such a link-edit by furnishing a file with a name TSK-xxxx:SYMB or xxxx:STSK that contains a list of all files to be linked together.

Lines beginning with a % character are considered as comments.
No blank lines are allowed.

```
example:     file (CONS-MAC)TASK-TEST:STSK

% link-descriptor of task test:
(cons-mac)test       % main program
(cons-mac)subr1      % 1st subroutine...
(cons-mac)subr2      % 2nd subroutine...
% end of list...
```

The resulting :CUF file will get a name with the same convention as for a source file.

### 7.3.5 How to create a loadable image of the SMACC.

A list of the application :CUF files to be linked together must be provided in a file with the name IMA-xxxx:SYMB or xxxx:SIMA. The rules are the same as for previously described :STSK files.

To produce the application loadable memory image of the SMACC (:BIN), you just have to specify:

```
@XCOM (M68K) <description file> <list>
```

A link-edit is done, including the following files :

- (LIBRARY-PPL-M68K)PPL-HDR-BEGIN:CUF that will fix the order of all existing sections and the description of all sections.
- the list of your CUF modules
- (LIBRARY-PPL-M68K)PPL-HDR-END:CUF that will define the end of all sections.
- (SMACC-SYSTEM)M68-SYSTEM-TABLE:CUF that will be used to define all symbols used from the system part.

From this link-edit, no undefined external reference must persist.

After this link-edit, the pusher is called, with the magic addresses specified in the memory partitionning.
This will result in the production of an MSC file.

To load the application on the SMACC, two commands must be issued:

- LDACC(smacc,"(P-PLUS-TEST)SMACC-SYSTEM",cc)
- LDACC(smacc,"<file>",cc)

### 7.3.6 Remarks on the program production procedures.

By default all program production procedures are submitting a job to the batch number 1. You don't need to specify your current user name and password because XCOM is able to find them. However, it is possible but not recommended to execute those job as MODE files instead of batch jobs by specifying @XCOM (M68K)/MODE.

Notice that XCOM needs at least 50 pages free in your user file space to work.

XCOM checks the existence of the file and finds the full file name.

The following files are dynamically created under VOLATILE:

```
        yyyyyyyyy:Bxxx  = Job batch file to compile
        yyyyyyyyy:Cxxx  = Second batch file to edit the result of this job.
        yyyyyyyyy:Dxxx  = Job  auxiliary  data file if needed (for example by
                          CUFLINK).
```

The listing output file is created by default under VOLATILE :

```
        yyyyyyyyy:Lxxx  = Resulting listing
```

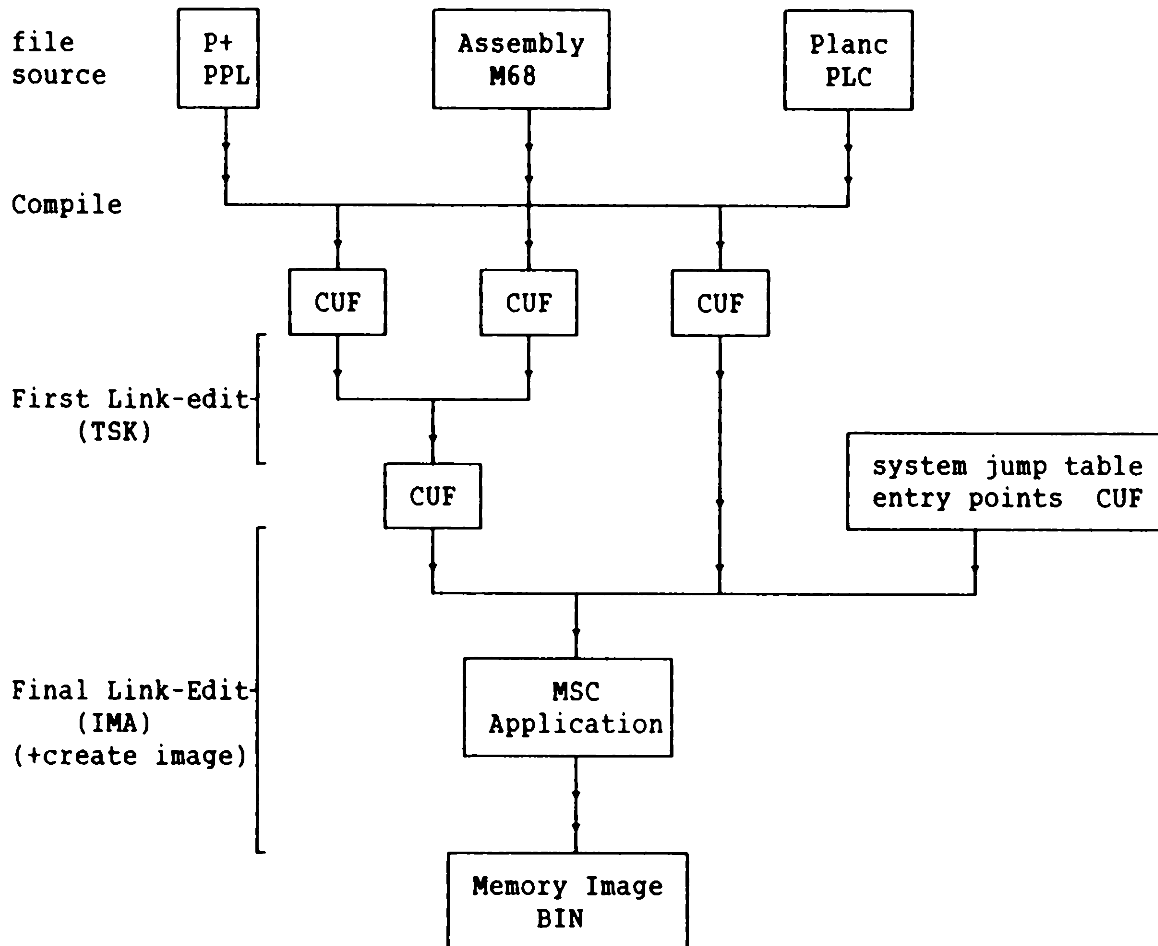
The asked print listing option can be one of the following :

```
        L = Line-printer
        P = Philips
        F = Laser-printer horizontal format
        W = Laser-printer vertical format
        X = Central DD Xerox 8700 laser printer (55 lines/page, 4 page/page)
        R = PS xerox 2700 laser printer
        N = No print out of the listing file.
        T = direct print out on terminal (only if /MODE switch is specified)
```

The following parameters can be modified for each user by creating a file USER-PARAMETERS:XCOM whose contents is following:

```
↑AP_PASSW:=='xxxx' % Application project password  (default: none)
↑MAX_TIME==100     % Maximum job time              (default: 10)
↑VOL_LIST:=='user' % File-space to keep LIST files (default: VOLATILE)
↑JNAM:=='XC#yyyyy' % Title for IBM laser printer   (default:XC#user)
                   %   XC -> delivery point 'XC' F1 6/R.012
```

7.3.7 Annexe: Supported Program production shema.




Notice than other program productions chains can be used, for example the PRIAM VAX chain can be used to use other languages as C, FORTRAN or PASCAL whose produce CUF files, that can then be transported through CERNET to the PRDEV and then integrated to this program production.

# 8 DEBUGGING A PROGRAM ON THE SMACC

Now that your know how to produce an application image, you want to debug it.

To debug a program on the SMACC, different ways of operation are possible and complementary:

- From remote computer a low-level debug facility is available. (This will be principally used by the operation people, because it does not need to connect any terminal to the SMACC).

- On the smacc interactive NODAL, it is possible to use the system commands and also to make a low-level debug of a NODAL function using the TILT, DUMP and LOOKAT functions to place break-points.

- The stand-alone MONICA monitor and debugger can be used to control RMS68K and the whole system, but the result of any call to MONICA is to stop the whole system until an action is taken by the operator.

- MONICA debugger running under the control of RMS68K will be implemented later to allow to place a break-point in any task, resulting in the stop of only that task.

The first debugging problem is to load the SMACC memory and to start it up.
The best thing to do is to start the FEC remote debugger and to use it to load the SMACC.

For that purpose, the questions asked by this NODAL program are the following:
- SMACC number ?
- Application to be loaded.

You can now load your SMACC by typing DO90.
In case of error 132, you must unbypass the serial crate by typing DO91.

After that your application is loaded and the interactive NODAL must be available on the connected terminal - if any (or the MONICA system if used).

In case of problems, the following steps must be followed:

1) Test if the system part does work alone without your application: restart the remote debugger and specify no application.

2) If this does not work, check the hardware connexion between SMACC and terminal.

3) If the link seems to be correct, check the communication parameter setting of the terminal (or of the MacIntosh): 4800 bauds, 8 bits, 1 start bit, 2 stop bits, No parity.

4) Now that you are sure of the fact that the hardware works, reload your application and check with the remote debug tools that all your tasks are started and that tasks MIOS and NODI are in wait state.
   If no one of the application tasks is started, check the contents of the task init blocks.
   If your tasks are started, you can now check the connexion of your ISR from the FEC and check for the current state of all tasks.

## 8.1 FEC Remote debugger

A low level remote debug facility is available on the FEC.

This program is called from RT-NODAL by the command:
RUN SMACC-DEBUG

It asks you first for the smacc number on which you are working, and then on the application to be loaded on the smacc after the loading of the system image (what is the normal case if you follow chapter 8 for program production.

The most command available are the following:

- D02 : re-edit the menu
- D040 : edit menu for CAMAC basic functions (usefull in case of hardware problems only).
- D098 : activate <PRDEV>TRANSFER-FILE utility to read your image from the PRDEV to the local computer.
- D091 : unbypass serial crate (this results also in a system restart if the power was not cut on the crate).
- D090 : load the system image and then the application image.
- D050 : patch some words into memory.
- D051 : dump 128 words from any even memory location
- D052 : dump system debug area
- D053 : dump system crash save area (and system stack)
- D060 : look at Task control blocks.
- D061 : list memory partitionning (usefull to check what dynamic area is yet unused).
- D062 : list all Interrupt Service routines connexions.
- D063 : list contents of semaphore table.
- D064 : dump contents of system trace table: this is significant only if the system is crashed.
- D065 : look at task's segments tables
- D066 : list contents of global segment table
- D067 : dump contents of periodic activation and delay table
- D068 : set the system trace flag (after start-up)
- D071 : put an irrecoverable break-point in memory (destroying 6 words) and wait for the user action). Then try to dump the registers contents at this location.

## 8.2 Using NODAL tools

Once interactive NODAL started, you can now use it to debug your programs.

The set of RMS68K commands allows you to issue any RMS68K call <i> and thus to check the state of any task.

To debug a NODAL function, a primary debug facility exists that allows you to place a breakpoint at 1 location (function TILT(address))- that will save the contents of all registers and then restart NODAL at the interactive level.
The contents of the registers and of the memory can then be checked through the DUMP and LOOKAT functions.

## 8.3 MoniCa debugger

The DD-supported RMS68K environment includes MoniCa debugger, an interactive symbolic debugger allowing such operations as:

1) tracing

2) breakpoints

3) inspection/modification of variables

4) single stepping/single procedure calls

based on information output by the compiler in addition to the object code.

The Monica debugger can be used in different ways:
- in stand-alone mode (outside RMS68K): It is possible to load RMS68K as a MONICA application and to use Monica debugger on the 2nd line provided on the SMACC. Take care to the fact that in this case, any breakpoint will make a full stop of the system.
- as an application running under RMS68K control: in this mode it will be possible to put a breakpoint inside a task, halting only this task and not the whole system.

No final documentation exists yet on MONICA, but you can get a draft on the IBM under WYLBUR from the file $CG.HVE.LIB(MONICA) and print it out by following the first lines of this file.

## 8.4 Other debug informations

---

<i> Refer to appendix Nodal interface to RMS68K

### 8.4.1 System does not start

If interactive NODAL does not, the system sofware initilization programs includes some debug points:

1) The basic software initialization programs write on the CAMAC station 1, for system debug purpose.

2) The system sofware use location "Debug_Area" - cleared initially by PRE-INIT - to notice some events for tasks 'INIT', 'GO ', 'MIOS', and for the line driver.

### 8.4.2 System crashes

There is no fail indicator on the SMACC, so - except the running level indicator lamp, the only indication that the running system may have crashed is that there is no response to operator input. When this happens, memory must be examined to determine the cause of the system crash.

#### 8.4.2.1 System crash area

If the system did detect an error condition and call its crash procedure (subroutine KILLER in RMS68K), the registers are saved in the CRASHSAV area as follow:

| Offset | | | | |
|---|---|---|---|---|
| 0 | PC | SR | | |
| 8 | D0 | D1 | D2 | D3 |
| $18 | D4 | D5 | D6 | D7 |
| $28 | A0 | A1 | A2 | A3 |
| $38 | A4 | A5 | A6 | A7 |

If the program counter and the status register displayed at CRASHSAV are zero, then the system did not crash.

#### 8.4.2.2 Nothing at crashsav

The symbol RUNNER is at offset $C from the start of SYSPAR. This is the current running task's TCB address.

Next display the contents of the running task's TCB (512 bytes).

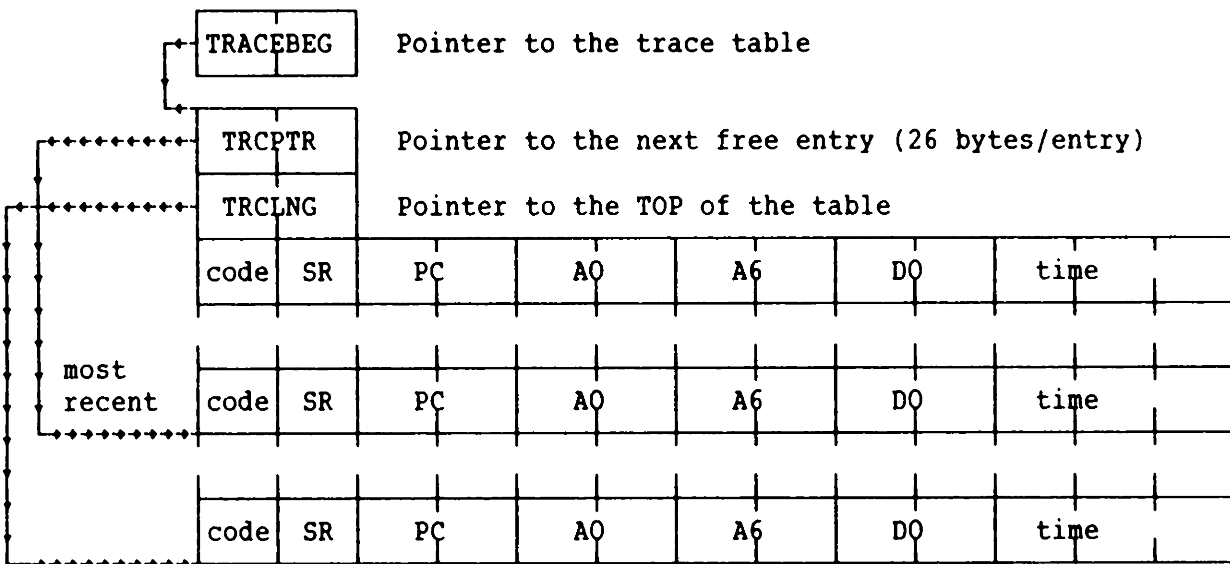
#### 8.4.2.3 System trace table

The system trace table can provide information about the most recent events that have occured while the system was running.

Entries are built in the system trace table when various events occur. The setting of the SYSGEN parameter TRACEFLG determines which events cause an entry to be built.

| Bit number in TRACEFLG | Event | Trace code |
|---|---|---|
| 15 | TRAP #1 | $FF15 |
| 14 | Interrupt not serviced | $EE14 |
| 13 | Timer interrupt | $FF13 |
| 12 | User trap (#2 - #15) | $AA12 |
| 11 | Exception | $AA11 |
| 10 | Dispatch | $FD10 |
| 9 | Interrupt serviced | $EE09 |
| 8 | Return from LOADMMU | $DD08 |
| 7 | SINT interrupt | $DD07 |
| 6 | SYSFAIL interrupt | $EE07 |

At TRACEBEG address is a pointer to the start of the trace table.



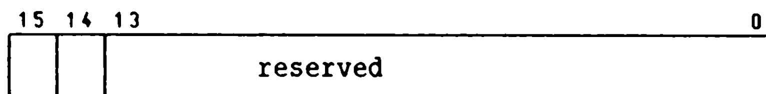
# A P P E N D I X   A

## P-PLUS INTERFACE TO RMS68K

```
MODULE PROVIDING rms_tools AS LIBRARY:
     PROVIDE ALL WITH
```

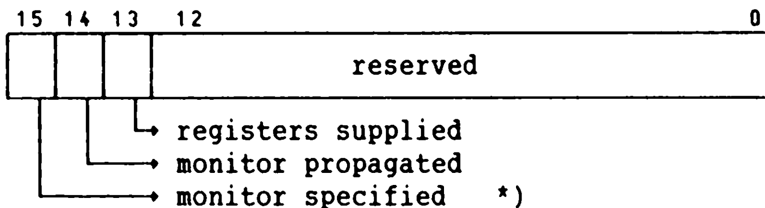
1) Task control directives

```
PROCEDURE Rms_crtcb        (* Create Task Control Block *)

(RO p1: INTEGER;      (* name    of new task *)
 RO p2: INTEGER;      (* session of new task *)
 RO P3: INTEGER;      (* options *)
 RO p4: INTEGER;      (* Monitor  task  name: This field is used only if
                         options bit 15=1. If this field is 0,  the  new
                         task's  monitor  will  be  the requesting task.
                         otherwise,  the  monitor  will  be   the   task
                         specified in this field *)
 RO p5: INTEGER;      (* Monitor task session: This field used only when
                         options bit 15=1 and the requesting task  is  a
                         system task.  If the field has the value 0, the
                         session  of  the  new  task's  monitor  is  the
                         requestor's  session.  Otherwise,  it  will  be
                         assigned to this specified value *)
 RO p6: INTEGER;      (* Initial priority to be assigned to the new task.
                          This priority can be changed at any time to a
                         value  less  or  equal  to  the  task's  limit
                         priority.  (A given task cannot affect another
                         task that has a current priority greater  than
                         its own limit priority.) *)
 RO p7: INTEGER;      (* Highest  priority  which can be assigned to the
                         new task. *)
 RO p8: INTEGER;      (* Task attributes :
        Bit 15=1 New task is a system task.
        Bit 13=1 Crash system if new task terminates abnormally.
        Bit 12=1 Task dump    if new task terminates abnormally.
        Bit 11=1 Relocatable task running with no MMU.  Entry
                 address will be adjusted when task is started. *)
 RO p9: address;      (* task entry point *)
 RO pA: INTEGER)      (* user  generated ID [-32767..32678]: Not used by
                              RMS68K,  but  is  for the user's information
only. *)
 ENTRY '_CRTCB';

(* options :
```




```
   └→ New task's monitor is the same as for requesting task if=1
──────→ New task's monitor p3 if=1 *)
```

```
 FUNCTION Rms_start        (* start task *)

 (RO p1: INTEGER;      (* target task's name: If the field is 0, look
                          for a task to re-start. *)
  RO p2: INTEGER;      (* target task's session *)
  RO p3: INTEGER;      (* options *)
  RO p4: INTEGER;      (* monitor task's name (only if options bit 15)
                          if 0 monitor = requesting task. *)
  RO p5: INTEGER;      (* monitor task's session (only if options bit 15)
                                 if 0 session = same as requesting task *)
  RO p6: ROW[1..15]OF INTEGER)  (* This  field contains the init. value
                                     of D0-D7, A0-A6 if option bit13=1   *)
         INTEGER        (* if start was called with TASKNAME=0, the      *)
ENTRY '_START';        (* Taskname of the started Task will be returned *)

(* options:
```

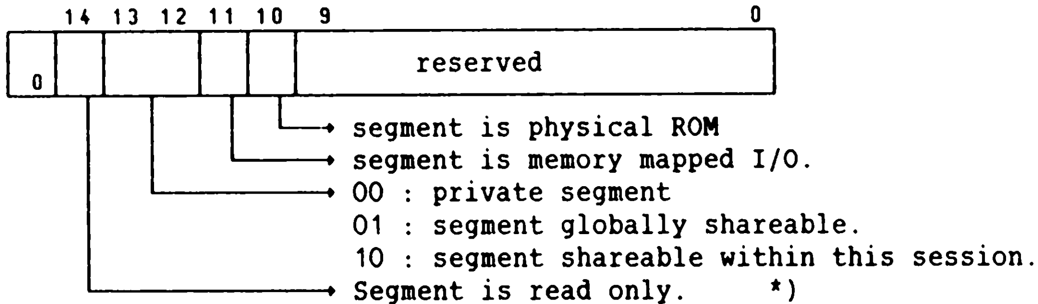



```
 PROCEDURE Rms_relinq      (* relinquish *)
     ENTRY '_RELINQ';

 PROCEDURE Rms_wait        (* enter wait state *)
     ENTRY '_WAIT';

 PROCEDURE Rms_suspnd      (* suspend self *)
     ENTRY '_SUSPND';

 PROCEDURE Rms_term        (* terminate self *)
     ENTRY '_TERM';

 FUNCTION Rms_stop         (* stop target task *)

 (RO p1: INTEGER;      (* target task's name *)
  RO p2: INTEGER)      (* target task's session*)
        INTEGER        (* Name of stopped task returned if stop Session *)
     ENTRY '_STOP';              (*  mode used *)

 FUNCTION Rms_termt        (* terminate target task *)

 (RO p1: INTEGER;      (* target task's name *)
  RO p2: INTEGER;      (* target task's session*)
  RO p3: INTEGER)      (* abort code 16 bits *)
        INTEGER        (* Name of terminated Task returned if  *)
 ENTRY '_TERMT';       (*          "Terminate" session used.   *)
```

```
FUNCTION Rms_setpri      (* set priority *)

(RO p1: INTEGER;       (* target task's name *)
 RO p2: INTEGER ;      (* target task's session*)
 RO p3: INTEGER)       (* target task's new priority *)
        INTEGER        (* If error code 10 is returned in DO, then the *)
    ENTRY '_SETPRI'; (* target task's limit priority will be returned *)

PROCEDURE Rms_abort         (* Abort Self *)

(RO p1 : INTEGER)      (* abort code *)
    ENTRY '_ABORT';

PROCEDURE Rms_wakeup     (* wakeup target task *)

(RO p1: INTEGER;       (* target task's name *)
 RO p2: INTEGER)       (* target task's session *)
    ENTRY '_WAKEUP';

PROCEDURE Rms_resume     (* resume target task *)

(RO p1: INTEGER;       (* target task's name *)
 RO p2: INTEGER)       (* target task's session*)
    ENTRY '_RESUME';
```

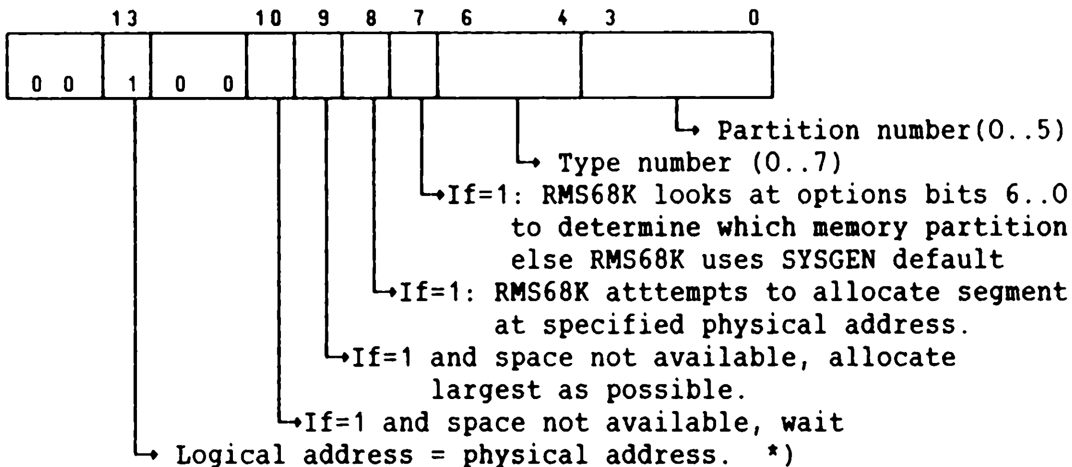
2) Memory management directives

(* Common to all segment management directives *)
(* attributes :

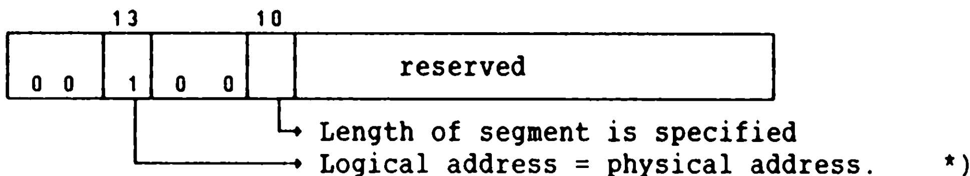



```
PROCEDURE Rms_gtseg        (* Allocate a segment *)

(RO p1: INTEGER;     (* Name    of task to receive the segment *)
 RO p2: INTEGER;     (* Session of task to receive the segment *)
 RO p3: INTEGER;     (* directive options (16 bits) *)
 RO p4: INTEGER;     (* segment attributes *)
 RO p5: INTEGER;     (* Name of new segment *)
 RO p6: address;     (* logical or physical address if specified *)
 RO p7: INTEGER;     (* segment length in bytes *)
 WO p8: address;     (* Physical address of new segment (reg A0) *)
 WO p9: INTEGER)     (* size  of  largest  free  block  available  if
                        directive  is  rejected,  Number  of  bytes
                        allocated if option bit 9=1 *)
    ENTRY '_GTSEG';
```

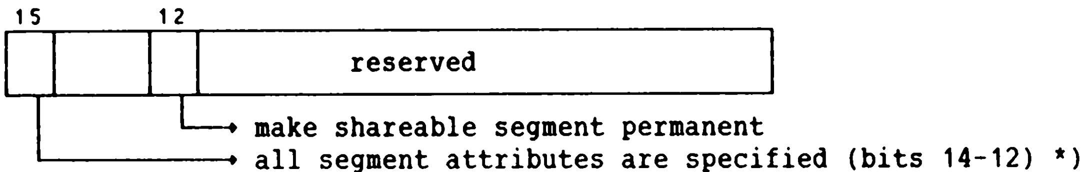
(* options :



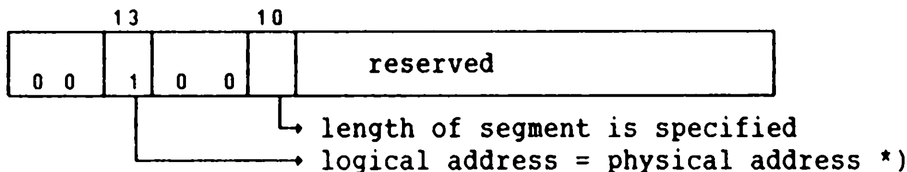
```
FUNCTION Rms_attseg        (* Attach a shareable Segment *)

(RO p1: INTEGER;        (* N/A *)
 RO p2: INTEGER;        (* N/A *)
 RO p3: INTEGER;        (* directive options *)
 RO p4: INTEGER;        (* segment attributes (16 bits):
                           Bit 13=1  segment  to  attach  is  a  locally
                                     shareable segment.
                           Bit 12=1  segment  to  attach  is  a  globally
                                     shareable  segment.
                           Either  bit 12 or 13 (but not both) must be set
                           to 1. *)
 RO p5: INTEGER;        (* name of the required segment *)
 RO p6: address;        (* Logical  address  of  segment  within  task's
                           address space. Not applicable  if  options  bit
                           13=1 *)
 RO p7: INTEGER)        (* Length  of  segment  to be attached. Applicable
                           only if options bit 10=1. The  value  specified
                           must  be  <  or  =  to the actual length of the
                           segment *)
       INTEGER          (* return physical address of allocated segment *)
   ENTRY '_ATTSEG'

(* options:
```



```
PROCEDURE Rms_dclshr       (* Declare a segment shareable *)

(RO p1: INTEGER;       (* N/A *)
 RO p2: INTEGER;       (* N/A *)
 RO p3: INTEGER;       (* directive options (16 bits):
 RO p4: INTEGER;       (* segment attributes (16 bits):
          Bit 13=1 segment to attach is a locally shareable segment.
          Bit 12=1 segment to attach is a globally shareable segment.
              Either  bit 12 or 13 (but not both) must be set to 1. *)
 RO p5: INTEGER;       (* name of the required segment *)
 RO p6: address;       (* Logical   address   of  segment0  within  task's
                          address space. Not applicable  if  options  bit
                          13=1 *)
 RO p7: INTEGER)       (* Length  of  segment  to be attached. Applicable
                          only if options bit 10=1. The  value  specified
                          must  be  < or  =  to the actual length of the
                          segment *)
    ENTRY '_DCLSHR';

(* options :
```

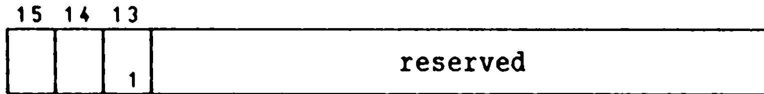



```
FUNCTION Rms_shrseg      (* grant shared acces to another task *)

(RO p1: INTEGER;       (* target task's name *)
 RO p2: INTEGER;       (* target task's session *)
 RO p3: INTEGER;       (* options *)
 RO p4: INTEGER;       (* segment attributes *)
 RO p5: INTEGER;       (* segment name *)
 RO p6: address;       (* logical address *)
 RO p7: INTEGER)       (* segment length in bytes *)
        address        (* Physical address of segment *)
    ENTRY '_SHRSEG';

(* options :
```

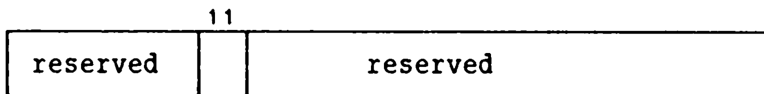



```
FUNCTION Rms_trseg       (* transfer a segment *)

(RO p1: INTEGER;       (* target task's name *)
 RO p2: INTEGER;       (* target task's session*)
 RO p3: INTEGER;       (* options: *)
 RO p4: INTEGER;       (* attributes :
                          Bit 14=1  Segment is to be read-only.
                                =0  Segment is to be read-write. *)
 RO p5: INTEGER;       (* Name of segment to be transferred. *)
```

```
 RO p6: address)       (* Logical address if bit 14=1 *)
        INTEGER        (* Physical address of segment *)
    ENTRY '_TRSEG';

(* options:
```




```
PROCEDURE Rms_deseg        (* Deallocate a Segment *)

(RO p1: INTEGER;       (* Name of target task to loose the segment *)
 RO p2: INTEGER;       (* session of target task to loose the segment *)
 RO p3: INTEGER;       (* directive options *)
 RO p4: INTEGER;       (* N/A *)
 RO p5: INTEGER)       (* name of the required segment *)
    ENTRY '_DESEG';

(* options:
```



```
PROCEDURE Rms_rcvsa        (* receive segment attributes *)

(RO p1: INTEGER;      (* name    of task owning the segment *)
 RO p2: INTEGER;      (* session of task owning the segment *)
 RO p3: INTEGER;      (* options : *)
 RO p4: INTEGER;      (* N/A : directive attributes *)
 RO p5: INTEGER;      (* Segment name if Bit 14=0 *)
 RO p6: address;      (* logical address if bit 14=1 *)
 RO p7: INTEGER;      (* N/A : segment length *)
 WO p8: ROW[1..5] OF INTEGER)   (* destination buffer address *)
   ENTRY '_RCVSA';

(* options:
```

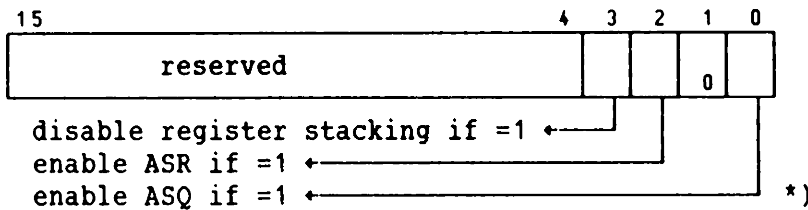
```
                 No information is returned in the caller's buffer.
                 Segment is identified by the specified address.

The returned buffer has the following structure :
       p8(1) = segment name
       p8(2) = segment attributes (16 usefull bits)
       p8(3) = segment begin logical address
       p8(4) = segment ending logical address
       p8(5) = segment begin physical address *)
```

3) Semaphore management directives

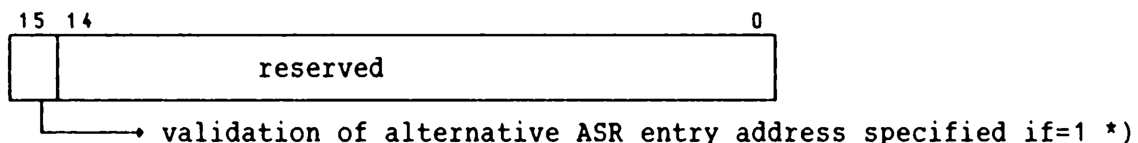
```
FUNCTION Rms_crsem          (* Create a semaphore *)

(RO p1: INTEGER;        (* semaphore name *)
 RO p2: INTEGER;        (* semaphore key *)             (* N/A *)
 RO p3: INTEGER;        (* initial count used for type 2 & 3
                           Must be non-negative value *)
 RO p4: INTEGER)        (* semaphore type [1,2,3] *)
        INTEGER         (* returned semaphore key *)
    ENTRY '_CRSEM';

PROCEDURE Rms_wtsem         (* wait on semaphore *)

(RO p1: INTEGER;        (* semaphore name *)
 RO p2: INTEGER)        (* semaphore key *)
    ENTRY '_WTSEM';

PROCEDURE Rms_sgsem         (* signal semaphore *)

(RO p1: INTEGER;        (* semaphore name *)
 RO p2: INTEGER)        (* semaphore key  *)
    ENTRY '_SGSEM';

FUNCTION Rms_atsem          (* Attach to semaphore *)

(RO p1: INTEGER;        (* semaphore name *)
 RO p2: INTEGER;        (*  N/A *)
 RO p3: INTEGER;        (*  N/A *)
 RO p4: INTEGER)        (* semaphore type [1,2,3] *)
        INTEGER         (* returned semaphore key *)
    ENTRY '_ATSEM';

PROCEDURE Rms_desem         (* detach from semaphore *)
(RO p1: INTEGER;        (* semaphore name *)
 RO p2: INTEGER)        (* semaphore key *)
    ENTRY '_DESEM';

PROCEDURE Rms_desema        (* Detach from all semaphores *)
    ENTRY '_DESEMA';
```

4) Event management directives

```
PROCEDURE Rms_gtasq        (* Allocate ASQ *)

(RO p1: INTEGER;       (* target task name or 0 if own task *)
 RO p2: INTEGER;       (* target task session *)
 RO p3: INTEGER;       (* ASQ status *)
 RO p4: INTEGER;       (* max message length in bytes *)
 RO p5: INTEGER;       (* max queue length in bytes *)
 RO p6: address)       (* ASR entry address *)
    ENTRY '_GTASQ';

(* ASQ status :
```




```
                                                         *)

PROCEDURE Rms_setasq        (* Set ASQ/ASR Status *)

(RO p1: INTEGER)       (* new ASQ status : (only bit 0 bit 2 used)
                          Bit 0 : enable ASQ if =1
                          Bit 2 : enable ASR if =1 *)
    ENTRY '_SETASQ';

PROCEDURE Rms_deasq        (* Deallocate ASQ *)
    ENTRY '_DEASQ';

PROCEDURE Rms_qevnt        (* queue event to task *)

(RO p1: INTEGER;       (* destination task name *)
 RO p2: INTEGER;       (* destination task session *)
 RO p3: INTEGER;       (* directive options *)
 RO p4: event_sended;(* buffer address *)
 RO p5: address)       (* Alternate ASR entry point if specified *)
    ENTRY '_QEVNT';

(* options:
```

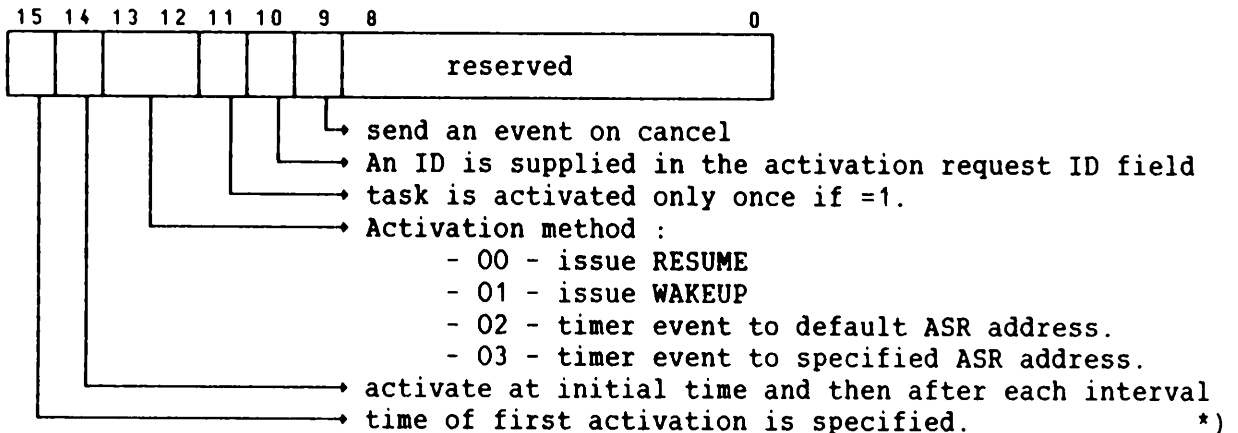



```
PROCEDURE Rms_rdevnt       (* read event *)

 (WO p1: event_received)     (* event receiving area *)
    ENTRY '_RDEVNT';

PROCEDURE Rms_wtevnt       (* wait for event *)
    ENTRY '_WTEVNT';
```

5) Time and delay management routines

```
PROCEDURE Rms_stdtim      (* set system date and time *)

(RO p1: INTEGER;          (* New System Date  *)
 RO p2: INTEGER)          (* New System Time  *)
    ENTRY '_STDTIM';

PROCEDURE Rms_gtdtim      (* get system date and time *)

(WO p1: INTEGER;          (* number of days since 01/01/1980 *)
 WO p2: INTEGER)          (* number of milliseconds *)
    ENTRY '_GTDTIM';

PROCEDURE Rms_delay        (* Delay self *)

(RO p1: INTEGER)          (* Number of millisecondes to Delay  *)
    ENTRY '_DELAY';

PROCEDURE Rms_delayw       (* Wait for event or Delay *)

(RO p1: INTEGER)          (* Number of millisecondes to Delay  *)
    ENTRY '_DELAYW';

PROCEDURE Rms_rqstpa      (* request periodic activation *)

(RO p1: INTEGER;          (* target task's name *)
 RO p2: INTEGER;          (* target task's session*)
 RO p3: INTEGER;          (* options :
 RO p4: INTEGER;          (* Initial activation time *)
 RO p5: INTEGER;          (* Period in milliseconds *)
 RO p6: address;          (* ASR entry address (only if bits 13-12=1) *)
 RO p7: INTEGER)          (* Activation request ID used to identify this
                             request *)
    ENTRY '_RQSTPA';
```

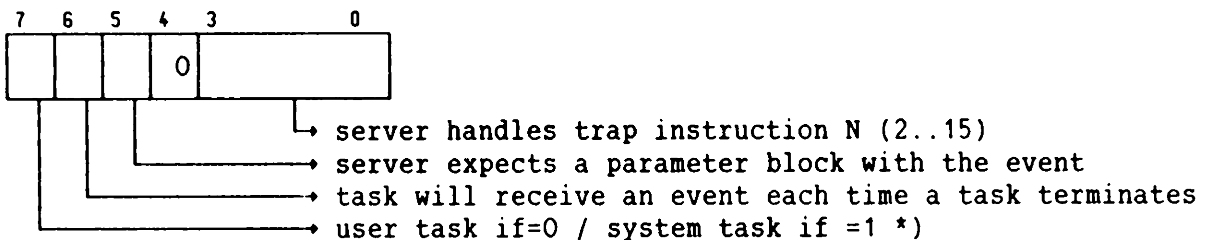
(* options:



6) Server directives

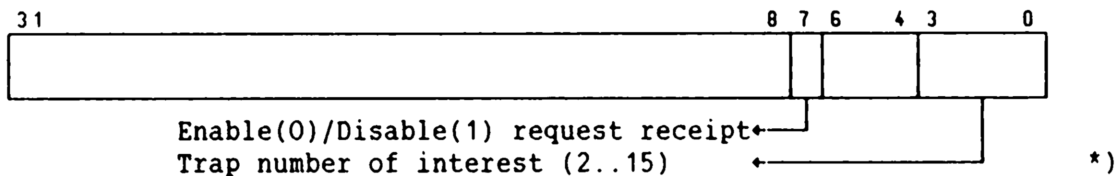
```
PROCEDURE Rms_server        (* establish server *)
(RO p1: address;        (* TRAP ASR address *)
 RO p2: INTEGER;        (* Trap instruction identifier: *)
 RO p3: INTEGER)        (* Parameter block size in bytes, Required if
                           TRAP instruction identifier bit 5=1.
                           Specifies size of parameter block which will
                           be given to the task with the event *)
    ENTRY '_SERVER';
```

(* trap instruction identifier:



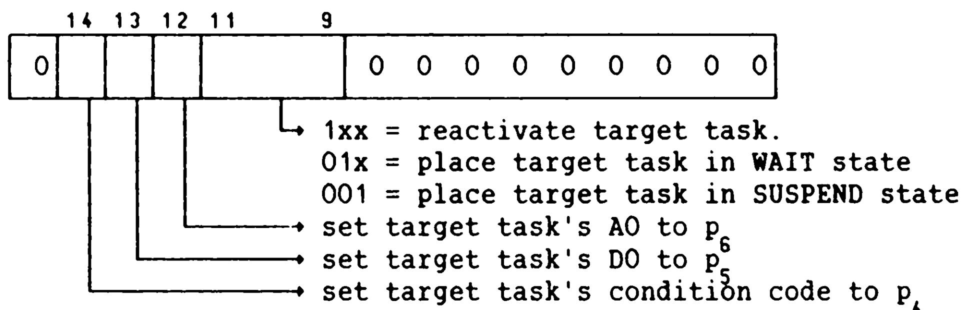

```
PROCEDURE Rms_derqst        (* set user/server request status *)

(RO p1: INTEGER)        (* trap number & status *)
    ENTRY '_DERQST';
```

(* parameter :




```
PROCEDURE Rms_akrqst        (* Acknowledge service request *)

(RO p1 : INTEGER;       (* target task name    *) (* whose request is *)
 RO p2 : INTEGER;       (*    "        "   session *) (* being acknowledged *)
 RO p3 : INTEGER;       (* directive options (16 bits) :
 RO p4 : INTEGER;       (* Trap number being acknowledged [2..15] *)
 RO p5 : INTEGER;       (* condition codes supplied if options bit 14=1 *)
 RO p6 : INTEGER;       (* D0 register, supplied if options bit 13=1 *)
 RO p7 : INTEGER)       (* A0 register, supplied if options bit 12=1 *)
    ENTRY '_AKRQST';
```

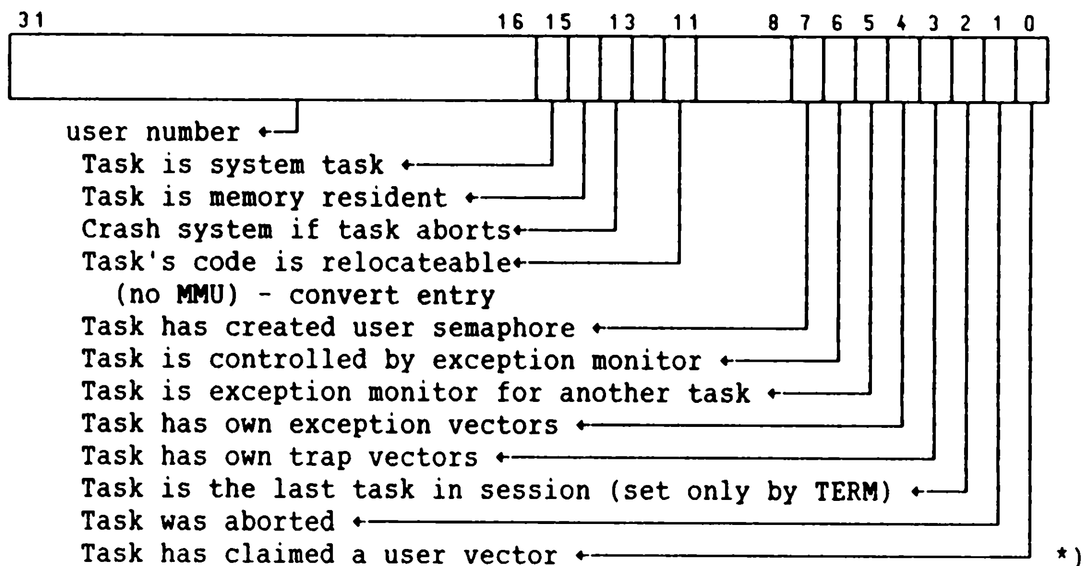
(* options:



```
PROCEDURE Rms_dserve      (* Deallocate server functions *)

(RO p1 : integer);   (* trap number (2..15) *)
    ENTRY '_DSERVE';
```

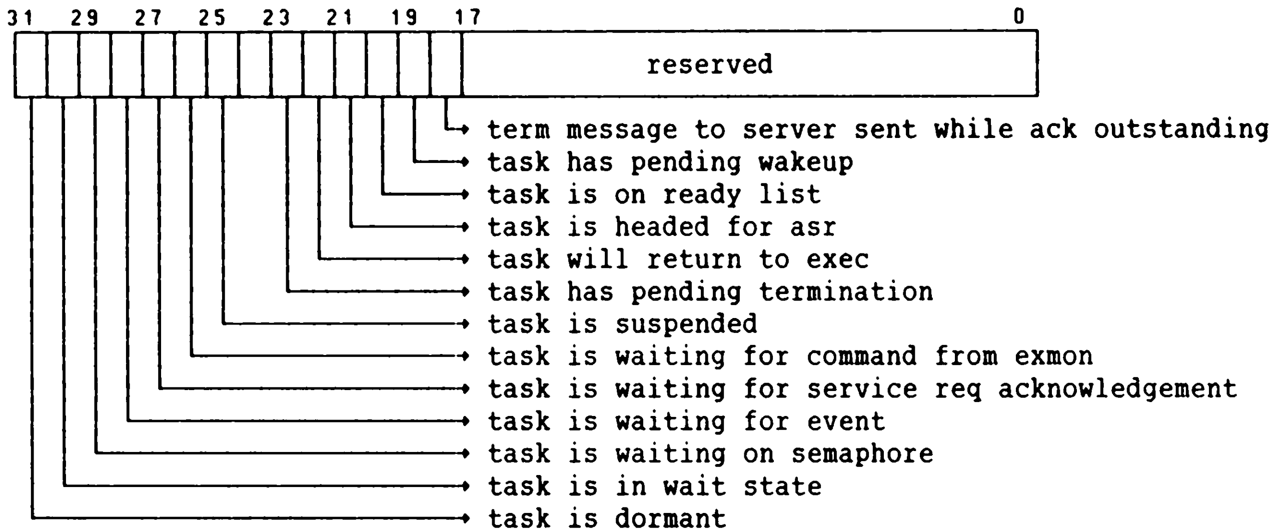
7) Exception monitor handling

```
PROCEDURE Rms_exmon       (* Attach Exception Monitor *)

(RO p1: INTEGER;      (* target task name *)
 RO p2: INTEGER;      (* target task session *)
 RO p3: INTEGER;      (* exception monitor task name *)
 RO p4: INTEGER)      (* exception monitor task session *)
    ENTRY '_EXMON';

PROCEDURE Rms_exmmsk      (* set exception monitor mask *)

(RO p1: INTEGER;      (* target task's name *)
 RO p2: INTEGER;      (* target task's session*)
 RO p3: INTEGER)      (* exception monitor mask *)
    ENTRY '_EXMMSK';

FUNCTION Rms_tskattr      (* read task attributes *)

(RO p1: INTEGER;      (* target task's name *)
 RO p2: INTEGER)      (* target task's session*)
        INTEGER       (* Task attributes (32 bits) *)
    ENTRY '_TSKATTR';

(* returned attributes :
```




```
PROCEDURE Rms_pstate      (* put task state *)

(RO p1: INTEGER;      (* target task's name *)
 RO p2: INTEGER;      (* target task's session*)
 RO p3: ROW[1..25] of INTEGER)  (* task state *)
    ENTRY '_PSTATE';
```

```
(* P3 contents :
       p3(1..15) = registers D0..A6
       p3(16)    = User stack pointer
       p3(17)    = Program counter
       p3(18)    = status register (only 16 bits used)
       p3(19)    = exception monitor mask  *)

 PROCEDURE Rms_rstate      (* receive task state *)

 (RO p1: INTEGER;     (* target task's name *)
  RO p2: INTEGER;     (* target task's session *)
  RO p3: task_state)  (* buffer address *)
     ENTRY '_RSTATE';

(* P3 contents :
       p3(1..19) = idem as for Pstate
       p3(20) = task status
       p3(21) = execution options
       p3(22) = value location
       p3(23) = value
       p3(24) = value mask
       p3(25) = max.instruct.count  *)

(* task_status:
```




```
 PROCEDURE Rms_tskinfo     (* return a copy of tcb *)

 (RO p1: INTEGER;     (* target task's name *)
  RO p2: INTEGER;     (* target task's session*)
  RO p3: INTEGER;     (* options :
                         Bit 15=1  Return copy of target task's TCB.
                         Bit  15=0   Do  not  return copy of target task's
TCB. *)
  RO p4: TCB_table)   (* 512-byte  buffer  where  a  copy of the target
                         task's TCB will be moved. *)
     ENTRY '_TSKINFO';

 PROCEDURE Rms_rexmon      (* run task under exception monitor *)
```

```
(RO p1: INTEGER;      (* target task's name *)
 RO p2: INTEGER;      (* target task's session*)
 RO p3: ROW[1..5]OF INTEGER)   (*      array reference *)
    ENTRY '_REXMON';

        (* p3(1) = execution options
           p3(2) = value location
           p3(3) = value
           p3(4) = value mask
           p3(5) = max.instruct.count  *)

PROCEDURE Rms_dexmon      (* Detach exception monitor *)

(RO p1: INTEGER;      (* Target   task  being  detached *)
 RO p2: INTEGER)      (* Target task session *)
    ENTRY '_DEXMON';

PROCEDURE Rms_expvct      (* Announce exception vectors *)

(RW p1: ROW[1..9] OF ADDRESS)
    ENTRY '_EXPVCT';

(* Exception vector table structure :
      p1(1) = bus error handling address
      p1(2) = address error handling address
      p1(3) = illegal instruction handling address
      p1(4) = zero divide
      p1(5) = CHK instruction
      p1(6) = TRAPV instruction
      p1(7) = privilege violation
      p1(8) = line 1010 emulator
      p1(9) = line 1111 emulator  *)
```

8) Miscellaneous directives

```
PROCEDURE Rms_trpvct      (* Announce trap vectors *)

(RW p1: ROW[2..15] OF ADDRESS)  (* trap vector table *)
    ENTRY '_TRPVCT';

PROCEDURE Rms_cisr        (* Configure Interrupt Service Routine *)

(RO p1: INTEGER;      (* target task name *)
 RO p2: INTEGER;      (* target task session *)
 RO p3: INTEGER;      (* directive options *)
 RO p4: INTEGER;      (* vector   number:  The  vector  number  of  the
                         exception vector being  allocated,  deallocated
                         or switch. Values can be $00 to $ff *)
 RO p5: address;      (* ISR entry point address *)
 RO p6: INTEGER)      (* A user-defined value that will be loaded into
                         address register A1 when an interrupt occurs*)
    ENTRY '_CISR';

(* options:
```

000 : Allocate exception vector to target task's ISR

```
        001 : Dissolve an existing ISR vector
              (only vector number must be specified)
        010 : Switch an exception vector to a new ISR.
 PROCEDURE Rms_sint        (* simulate interrupt *)

  (RO p1 : integer1;   (* Hardware interrupt priority *)
   RO p2 : integer1)   (* Exception vector number (0..$FF) *)
     ENTRY '_SINT';

 PROCEDURE Rms_movell

  (RO p1: INTEGER;     (* source task name *)
   RO p2: INTEGER;     (* source task session *)
   RO p3: address;     (* source logical address *)
   RO p4: INTEGER;     (* destination task name *)
   RO p5: INTEGER;     (* destination task session *)
   RO p6: address;     (* destination logical address *)
   RO p7: INTEGER)     (* Length of data block in bytes *)
     ENTRY '_MOVELL';

 PROCEDURE Rms_movepl

  (RO p1: address;     (* source physical address *)
   RO p2: INTEGER;     (* destination task name *)
   RO p3: INTEGER;     (* destination task session *)
   RO p4: address;     (* destination logical address *)
   RO p5: INTEGER)     (* Length of data block in bytes *)
     ENTRY '_MOVEPL';
```

# A P P E N D I X B

## Nodal interface to RMS68K

The following commands are available from SMACC nodal as system commands and can be used to test the different calls to RMS68K and to find the right values to be specified as parameters.

The @HELP function can be used for online help (but do not give any explanation on the use of the different arguments).

Task, semaphore, segment names parameters are given as 4 character strings. Session parameter should be equal to 1 when specified.

On error, NODAL prints always an error message in the form:
Rms directve failure : xx.
The value of xx can be tested via the RMS_DO nodal variable and interpreted according to the RMS68K error codes appendix.

```
@ABORT-SELF p1                                               (abort)
 p1 : abort code          (integer: 2 bytes)

@ACKNOWLEDGE-SERVICE-REQUEST p1,p2,p3,p4,p5,p6,p7            (akrqst)
 p1 : name of Target task whose request is being acknowledged.
 p2 : session
 p3 : directive options
           Bit 14=1  Set  target  task's  condition  codes in status register as
                     specified.
           Bit 13=1  Set target task's register D0 to value specified.
           Bit 12=1  Set target task's register A0 to value specified.
           Bit 11-9  =1xx    Reactive target task.
                     =01x    Place target task in WAIT    state.
                     =001    Place target task in SUSPEND state.
 p4 : Trap number being acknowledged.
 p5 : condition codes supplied if options bit 14=1.
 p6 : register  d0    supplied if options bit 13=1.
 p7 : register  a0    supplied if options bit 12=1.

@ALLOCATE-A-SEGMENT p1,p2,p3,p4,p5,p6,p7,p8,p9               (gtseg)
 p1 : Name of task to receive segment.
 p2 : Session        N/A if requestor is user task.
 p3 : Directives option
         bit 13 = 1 must be specified (logical address=physical)
         bit 10 = 1 : wait if space not yet available
         bit  9 = 1 : allocate as much as possible if space not available
         bit  8 = 1 : allocate segment at specified address
         bit  7 = 1 : do not use default(1) partition for allocation,but
                      use bits 6..0 for partition type and number.
 p4 : Segment Attributes
         bit 14 = 1 : segment is read only
         bit 13..12 = 00 : private segment
                      01 : globally shareable segment
                      10 : locally shareable segment
         bit 11 = 1 : segment is memory mapped I/O
         bit 10 = 1 : segment is physical ROM
 p5 : Name of new segment.
 p6 : Address of segment Logical or Physical
 p7 : Segment length (in bytes)
 p8 : returned physical address of segment
 p9 : returned size of largest block available

@ALLOCATE-ASYNCHRONOUS-SERVICE-QUEUE p1,p2,p3,p4,p5,p6,p7  (gtasq)
 p1 : Task name
```

```
 p2 : session
 p3 : ASQ status
 p4 : max mess. length
 p5 : queue length
 p6 : ASR service vector
 p7 : receiving area  address

@ANNOUNCE-EXCEPTION-VECTORS  p1                              (expvct)
 p1: exception vector table consists of 9 Long integer entries, each of which is
     the Transfer Address for the appropriate Exception.
                  p1(1) :     bus error
                  p1(2) :     address error
                  p1(3) :     illegal instruction
                  p1(4) :     zero    divide
                  p1(5) :     chk     instruction
                  p1(6) :     trapv   instruction
                  p1(7) :     privilege violation
                  p1(8) :     line    1010 emulator
                  p1(9) :     line    1111 emulator

@ANNOUNCE-TRAP-VECTORS  p1                                   (trpvct)
 p1  : trap vector table consists of 14 4-byte entries, each of which is the the
       transfer address for the appropriate trap instruction.  The table covers
       TRAP 2 through TRAP 15.

@ATTACH-A-SEGMENT p1,p2,p3,p4,p5,p6,p7,p8                    (attseg)
 p1 : task name
 p2 : session
 p3 : directive options
         Bit 13=1  RMS68K supplies logical address of segment equal to physical
                   address of segment.
         Bit 10=1  length of segment to be attached specified in the segment
                   length field.
 p4 : segment attributes   (value      : 2 bytes)
         Bit 13=1  segment to attach is a locally  shareable segment.
         Bit 12=1  segment to attach is a globally shareable segment.
                   Either bit 12 or 13 (but not both) must be set to 1.
 p5 : name of desired segment
 p6 : logical address of segment within task's addresses
      Not applicable if options bit 13=1.
 p7 : length of segment
      Applicable only if options bit 10=1. The value specified must be < or = to
      the actual length of the segment.
 p8 : returned physical address of segment

@ATTACH-EXCEPTION-MONITOR p1,p2,p3,p4                        (exmon)
 p1 : target task name
 p2 : target task session
 p3 : exception monitor task name
 p4 : exception monitor task session

@ATTACH-TO-SEMAPHORE p1,,,p4                                 (attsem)
 p1 : semaphore name
 p2 :      ** N/A **
 p3 :      ** N/A **
 p4 : semaphore type   1,2 or 3
 p5 : returned semaphore key

@CONFIGURE-INTERRUPT-SERVICE-ROUTINE p1,p2,p3,p4,p5,p6       (cisr)
 p1 : target task name or  requesting task if 0
```

```
 p2 : session   n/a if requestor is a user task
 p3 : directive options
      Bit 15-13 =000    Allocate exception vector to target task's ISR
                =001    Dissolve an existing  ISR vector connection. If this
                        option is specified, only the vector numberfiels must be
                        supplied.
                =010    Switch an exception vector to new ISR.
 p4 : vector number
 p5 : ISR address
 p6 :  argument  to pass in A1 at ISR entry

@CREATE-A-SEMAPHORE p1,,p3,p4,p5                           (crsem)
 p1 :    Name of semaphore to create.
 p2 :     ** N/A **
 p3 : initial count for type 2 & 3.
 p4 : semaphore type 1, 2, or 3
 p5 : returned semaphore key

@CREATE-TASK-CONTROL-BLOCK p1,p2,p3,p4,p5,p6,p7,p8,p9,p10  (crtcb)
 p1 : Name of new task
 p2 : session n/a if requestor is a user task
 p3 : directive options
       Bit 15=1 New task's monitor is specified in monitor fields of TCB block.
       Bit 14=1 New task's monitor will be requesting task's monitor.
 p4 : monitor task name (if p3 bit 15=1)
 p5 : monitor session
 p6 : initial priority
 p7 : limit   priority
 p8 : task    attributes
         Bit 15=1 New task is a system task.
         Bit 13=1 Crash system if new task terminates abnormally.
         Bit 12=1 Task dump    if new task terminates abnormally.
         Bit 11=1 Relocatable task running with no MMU.  Entry
                  address will be adjusted when task is started.
 p9 : task    entry point
 p10: user generated i.d.    (16 bits)

@DELAY-SELF p1                                             (delay)
 p1    : Number of millisecondes to Delay

@DELAY-AND-WAIT p1                                         (delayw)
 p1    : Number of millisecondes to delay

@DECLARE-A-SEGMENT-SHAREABLE ,,p3,p4,p5,,,                 (dclshr)
 p1 : ** N/A **
 p2 : ** N/A **
 p3 : directive options
 p4 : segment attributes
 p5 : Name of segment to be shareable.
 p6 : ** N/A  **
 p7 : ** N/A  **

@DELALLOCATE-A-SEGMENT p1,p2,p3,p4,p5,,,                   (deseg)
 p1 : name of target task to lose segment
 p2 : session          n/a if requestor is a user task
 p3 : directive option
         Bit 11=1 Remove permanent status of a shareable segment.
 p4 : ** N/A **
 p5 : name of segment to be deallocated
 p6 : ** N/A **
```

```
 p7 : ** N/A **

@DEALLOCATE-ASYNCHRONOUS-SERVICE-QUEUE                          (deasq)

@DEALLOCATE-SERVER-FUNCTIONS  p1                                (dserve)
 p1 : BIT 3-0 relevant Trap Instruction Number

@DETACH-ALL-SEMAPHORE                                           (desema)

@DETACH-EXCEPTION-MONITOR p1,p2                                 (dexmon)
 p1 : name of target task to detach from exception monitor
 p2 : session     n/a if requestor is user task.

@DETACH-FROM-SEMAPHORE p1,p2,,,                                 (desem)
 p1 : semaphore name
 p2 : semaphore key
 p3 : ** N/A **
 p4 : ** N/A **

@ESTABLISH-SERVER p1,p2,p3                                      (server)
 p1 : Address at which specified TRAP instruction is to be serviced.
 p2 : trap instruction identifier
        Bit 6=1  task elects to receive an event each time a task terminates.
        Bit 5=1  server expects a parameter block with the event.
        Bit 3-0  specify the number of the TRAP instruction which the
                 requesting TASK will serve at the above request service
                 address.  Valid values are 2 through 15.
 p3 : Parameter block size
         Required if TRAP instruction identifier bit 5=1.  Specifies size of
        parameter block which will be given to the task with the event.

@GET-SYSTEM-DATE-AND-TIME p1,p2                                 (gtdtim)
 p1 : returned current system date
 p2 : returned current system time

@GRANT-SHARED-ACCESS-TO-ANOTHER-TASK p1,p2,p3,p4,p5,p6,p7,p8 (shrseg)
 p1 : task name
 p2 : session
 p3 : directive option
 p4 : segment attributes
 p5 : segment name
 p6 : logical address
 p7 : segment length
 p8 : returned physical address of segment

@MOVE-FROM-LOGICAL-ADDRESS p1,p2,p3,p4,p5,p6,p7                 (movell)
 p1 : source task
 p2 : source session
 p3 : source logical address
 p4 : destination task
 p5 : destination session
 p6 : destination logical address
 p7 : length of data block

@MOVE-FROM-PHYSICAL-ADDRESS p1,p2,p3,p4,p5                      (movepl)
 p1 : source physical address
 p2 : destination task
 p3 : destination session
 p4 : destination logical address
 p5 : length of data block
```

```
@PUT-TASK-STATE  p1,p2,p3                                   (pstate)
 p1 : task name
 p2 : session
 p3 : Long-integer array containing task state
         p3(1) = D0
         p3(2) = D1
         ...
         p3(8) = D7
         p3(9) = A0
         ...
         p3(16) = A7
         P3(17) = PC
         p3(18) = SR
         p3(19) = exception monitor mask

@QUEUE-EVENT-TO-TASK p1,p2,p3,p4,p5                         (qevnt)
 p1 :destination task name
 p2 :session
 p3 :directive options
 p4 :event (Long integer array)
 p5 :alternate service vector

 WARNING : It is not possible to determine the data contents.
           It is the user's responsability to use the contents of the provided
           buffer

@READ-EVENT p1                                              (rdevnt)
 p1 : event (long integer array)

@RECEIVE-SEGMENT-ATTRIBUTES p1,p2,p3,,p5,p6,,p8             (rcvsa)
 p1 : taskname
 p2 : session
 p3 : directive options
         Bit 14=1  segment identified by logical address in logical address
                   field.
         Bit 14=0  Segment identified by segment name.
         Bit 13=1  No information will be returned in the user's buffer The
                   logical address of segment referenced will be returned in A0.
         Bit 13=0  All information about segment will be returned in caller's
                   buffer.
 p4 : ** N/A **
 p5 : segment name           if bit 14=0
 p6 : logical address        if bit 14=1
 p7 : ** N/A **
 p8 : return buffer  address                      (long words array )

@RECEIVE-TASK-STATE p1,p2,p3                                (rstate)
 p1 : task name
 p2 : session
 p3 : long-integer array receiving task state:
         p3(1) = D0
         p3(2) = D1
         ...
         p3(8) = D7
         p3(9) = A0
         ...
         p3(16) = A7
         P3(17) = PC
         p3(18) = SR
```

```
        p3(19) = exception monitor mask
        p3(20) = task status          (4 bytes)
        p3(21) = execution options    (2 bytes)
        p3(22) = value location       (4 bytes)
        p3(23) = value                (4 bytes)
        p3(24) = value mask           (4 bytes)
        p3(25) = max.instruct.count   (4 bytes)

@RELINQUISH                                               (relinq)

@RESUME-A-TARGET-TASK p1,p2                               (resume)
 p1  : task name
 p2  : session

@RETURN-A-COPY-OF-TASK-CONTROL-BLOCK                      (tskinfo)
 p1:   name of target task
 p2:   session
 p3:   options
          Bit 15=1  Return copy of target task's TCB.
          Bit 15=0  Do not return copy of target task's TCB.
 p4:   128 long integer buffer address  to receive TCB

@REQUEST-PERIODIC-ACTIVATION p1,p2,p3,p4,p5,p6,p7         (rqstpa)
 p1 : name of task to be activated
 p2 : session    n/a if requestor is a user task
 p3 : directive options
        Bit 15=1 Time of specified activation is specified in initial time
                 field.
              =0 Time of 1st activation = interval+calling sequence time.
        Bit 14=1 The interval is specified in interval field. Task will be
                 activated at initial time , initial time + interval, initial
                 time + 2*interval, etc...
              =0 Task is activated at initial time only.
        Bits 13-12  Activation method.
                    00 - issue resume.
                    01 - issue wakeup.
                    10 - timer event to default ASR service addr.
                    11 - timer event to ASR service addr. specified in service
                         addr. field.
        Bit 11=1 Task will be activated only one time (not required if Bit
                 14=0).
              =0 Task will be activated as option bit 14.
        Bit 10=1 An argument is supplied in Activation Request ID field.
              =0 Activation Request ID of all zeros is assumed.
        Bit  9=1 Send an event to target task when the activation is cancelled.
 p4 : initial time         time of day, in milliseconds.
 p5 : interval             period of time, in milliseconds
 p6 : ASR service address (only if bits 13-12=1).
 p7 : activation id used to identify this request

 Nota : if bits 15-14 = 0, this is a request to cancel a currently-active
        periodic activation.

@RUN-TASK-UNDER-EXCEPTION-MONITOR p1,p2,p3                (rexmon)
 p1 : task name
 p2 : session
 p3 : long integer array :
       p3(1) = execution options            (2 bytes)
       p3(2) = value location               (4 bytes)
       p3(3) = value                        (4 bytes)
```

```
       p3(4) = value mask                    (4 bytes)
       p3(5) = max.instructions count        (4 bytes)

@SET-ASQ/ASR-STATUS p1                                    (setasq)
 p1 : new asq status
        Bit 0 :  ASQ disable if =0, enable if =1
        Bit 2 :  ASR disable if =0, enable if =1

@SET-EXCEPTION-MONITOR-MASK p1,p2,p3                      (exmmsk)
 p1 : task name
 p2 : session
 p3 : exception monitor mask

@SET-PRIORITY p1,p2,p3,p4                                 (setpri)
 p1 : name of target task with changing priority.
 p2 : session n/a if requestor is a user task.
 p3 : New current priority
 p4 : returned priority if error 10

@SET-SYSTEM-DATE-AND-TIME p1,p2                           (stdtim)
 p1 : New System Date
 p2 : New System Time

@SET-USER/SERVER-REQUEST-STATUS p1                        (derqst)
 p1 : trap number & status

@SIGNAL-SEMAPHORE p1,p2,,,                                (sgsem)
 p1 : semaphore name
 p2 : semaphore key
 p3 : *** N/A ***
 p4 : *** N/A ***

@SIMULATE-INTERRUPT p1,p2                                 (sint)
 p1 : interrupt priority
 p2 : vector number

@SNAPSHOT-OF-SYSTEM-TRACE p1,p2                           (snaptrac)
 p1    : buffer where the System Trace table will be copied

@START-TASK p1,p2,p3                                      (start)
 p1 : task name
 p2 : session
 p3 : directive options
         Bit 15=1 the monitor of the target task is specified in monitor field.
         Bit 14=1 the monitor of the target task is the requesting task's
                  monitor.
         Bit 13=1 the registers of the task being started are to be initialized
                  to the values in the registers field of the parameter block.
 p4 : monitor task name   (only if options bit 15)
 p5 : monitor session     (only if options bit 15)
 p6 : init value of registers  D0-D7,A0-A6 (only if options bit 13)
 p7 : return parameter : if start was called with taskname=0, the taskname of the
            started Task will be returned

@STOP-TASK p1,p2,p3                                       (stop)
 p1 : task name
 p2 : session
 p3:  Name of stopped task returned if stop Session mode used

@SUSPEND-SELF                                             (suspnd)
```

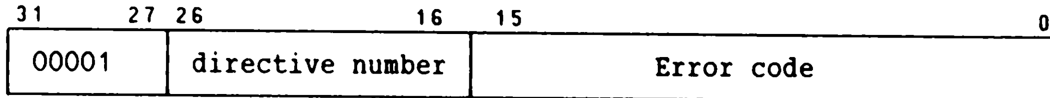
```
@TASK-ATTRIBUTES p1,p2,p3                                    (tskattr)
 p1 : Task name
 p2 : Session
 p3 : returned task attributes

@TERMINATE-SELF                                              (term)

@TERMINATE-TARGET-TASK p1,p2,p3,p4                           (termt)
 p1 : task name
 p2 : session
 p3 : abort code
 p4 : Name of terminated Task returned if Terminate session used.

@TRANSFER-A-SEGMENT p1,p2,p3,p4,p5,p6,,p8                    (trseg)
 p1 : Target task to receive segment.
 p2 : session
 p3 : directive options
        Bit 15=1  the attributes of the segment are changed according to the
                  segment attribute field.
        Bit 14=1  logical address supplied by requestor in segment block.
        Bit 13=1  RMS68K supplies logical address equal to physical address.
         If options bits 13 & 14 both are equal to zero, the logical address of
        the segment will be the same as currently assigned in the requestor's
        address space.
 p4 : directive attributes
        Bit 14=1  Segment is to be read-only.
              =0  Segment is to be read-write.
 p5 : Name of segment to be transferred.
 p6 : logical address         if bit 14=1
 p7 : ** n/a **
 p8 : returned Physical address of segment

@WAIT-STATE                                                  (wait)

@WAIT-FOR-EVENT                                              (wtevnt)

@WAIT-ON-SEMAPHORE p1,p2,,,                                  (wtsem)
 p1 : Semaphore Name
 p2 : Semaphore Key
 p3 : *** N/A *****
 p4 : *** N/A *****

@WAKEUP-A-TARGET-TASK p1,p2                                  (wakeup)
 p1: taskname
 p2: session
```

# A P P E N D I X C

## RMS68K ERROR CODES

# ˜RMS68K ERROR CODES˜

On return from most RMS68K directives, a return code is given as follow :



| 31 – 27 | 26 – 16 | 15 – 0 |
|---|---|---|
| 00001 | directive number | Error code |

1) error codes

$01 invalid directive number.
$02 parameter block out of requesting task's address space.
$03 target task does not exist.
$04 required table does not exist.
$05 table is full.
$06 duplicate request : function cannot be performed again.
$07 entry not found in table or list
$08 memory space not available
$09 requesting task is not a system task
$0A invalid target task state.
$0B request conflicts with existing
$0C address out of requesting task space
$0D address out of requesting task space
$0E function is not enabled
$0F invalid options specified
$10 invalid count or length field

2) directives number in error codes

| | | |
|---|---|---|
| $01 | GTSEG | Allocate a segment |
| $02 | DESEG | Detach a segment |
| $03 | TRSEG | Transfer a segment to another task |
| $04 | ATTSEG | Attach a shareable segment |
| $05 | SHRSEG | Grant shared segment acces |
| $06 | MOVELL | Move from logical address |
| $07 | DCLSHR | Declare a segment shareable |
| $08 | SNAPTRAC | Snapshot of system trace |
| $09 | RCVSA | Receive segment attributes |
| $0B | CRTCB | Create task control block |
| $0D | START | Start a target task |
| $0E | ABORT | Task aborts itself |
| $0F | TERM | Task terminates itself |
| $10 | TERMT | Abort target task |
| $11 | SUSPND | Task moves itself to suspend state |
| $12 | RESUME | Move target task from suspend to ready state |
| $13 | WAIT | Task moves itself to suspend state |
| $14 | WAKEUP | Move target task from wait to ready state |
| $15 | DELAY | Task moves itself to delay state |
| $16 | RELINQ | Task moves itself from run to ready state |
| $17 | TSKATTR | Receive task user number and attributes |
| $18 | SETPRI | Change priority of a task |
| $19 | STOP | Move target task from any state to dormant |
| $1A | EXPVCT | Announce exception vectors |
| $1B | TRPVCT | Announce trap vectors |
| $1C | TSKINFO | Receive a copy of the TCB |
| $1D | RQSTPA | Task is set up for periodic activation |
| $1E | DELAYW | Wait for event or delay |
| $1F | GTASQ | Allocate ASQ |
| $20 | DEASQ | Detach ASQ |

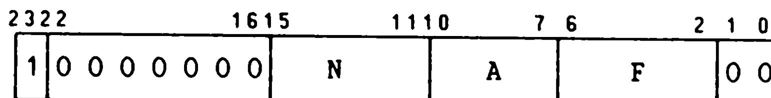
| | | |
|---|---|---|
| $21 | SETASQ | Task changes its ASR/ASQ status |
| $22 | RDEVNT | Task reads an event from the ASQ |
| $23 | QEVNT | Queue an event to a target task's ASQ |
| $24 | WTEVNT | Task moves itself to wait for event state |
| $29 | ATSEM | Attach to semaphore |
| $2A | WTSEM | Wait on semaphore |
| $2B | SGSEM | Signal semaphore |
| $2C | DESEM | Detach from semaphore |
| $2D | CRSEM | Create semaphore |
| $2E | DESEMA | Detach from all semaphores |
| $33 | SERVER | Task is made server task |
| $34 | DSERVE | Detach server function |
| $35 | DERQST | Set user/server request status |
| $36 | AKRQST | Server acknowledge request |
| $3A | CDIR | Configures a new directive |
| $3D | CISR | Configures ISR |
| $3E | SINT | Simulate interrupt |
| $40 | EXMON | Attach exception monitor |
| $41 | DEXMON | Detach exception monitor |
| $42 | EXMMSK | Set exception monitor mask |
| $43 | RSTATE | Receive task state |
| $44 | PSTATE | Modify task state |
| $45 | REXMON | Run task under exception monitor control |
| $48 | MOVEPL | Move from physical to logical addresses |
| $49 | STDTIM | Set system date and time |
| $4A | GTDTIM | Get system date and time |

# A P P E N D I X D

## ACCES TO CAMAC FROM THE SMACC

This Appendix explains how to access the CAMAC on the SMACC from a P+ programmer point of view, or from an assembly language programmer point of view.

A CAMAC function is generated by writing or reading into the CAMAC address space (locations $800000 to $80FFFC).




From the programmer point of view, the CAMAC address space is seen as a big memory buffer declared in P+ as follow:

```
VARIABLE CAMAC ENTRY '_CAMAC': ROW[0..32767] OF INTEGER[WORD];
VARIABLE CSR   ENTRY '_CSRREAD': INTEGER[WORD]
```

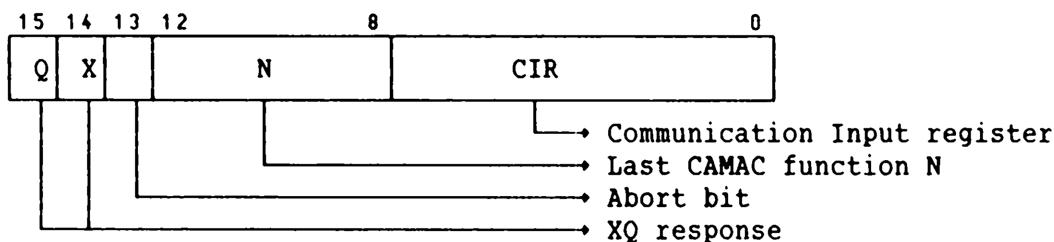
The different access to the CAMAC are performed as follow:

```
val:=CAMAC[N*1024+A*64+F*2];    for a CAMAC read function

CAMAC[N*1024+A*64+F*2]:=val;    for a CAMAC write function

QX:=CAMAC[N*1024+A*64+F*2];     for a CAMAC control function
```

The QX response from a read or write function can be read from the Communication Status Register (CSR) as follow:

```
QX:=CSR;
```

The QX response (read from CSR or as result of a CAMAC control function) has the following structure :




But after a read or a write function, the CSR read can be not the right one (because for example, an interrupt has started an ISR that does access to the CAMAC). To solve this problem, an extra P+ subroutine has been added:

```
PROCEDURE CAMAC_TRAP(RO NAF:INTEGER; (* N*512+A*64+F  to execute *)
                     RW val:INTEGER; (* value to read or to write *)
                     WO qx :INTEGER) (* camac QX response *)
```

In assembly language, the programmer works as follow:

```
       EXTERN    _CAMAC                 * refer to the base of the CAMAC area
       EXTERN    _CSRREAD               * refer to the CSR

       MOVEA.L   #_CAMAC,A0              * CAMAC basis in A0
       MOVE.W    #N<<11+A<<7+F<<2,D1     * NAF*4 in D1

       MOVE.W    #val,D0                * value to write
       MOVE.W    D0,0(A0,D1.W)          * to perform a CAMAC write function

       MOVE.W     0(A0,D1.W),D0           *  to perform a CAMAC read or control
function
                                        *  D0  =  value  read  if  CAMAC  read
function
                                        *  D0  =  CSR  read  if  CAMAC  control
function

       MOVE.W    _CSRREAD,D2            * read the CSR register in D2
```

If a read or write CAMAC function (or also a control function) needs a safe qx response, the TRAP #5 can be used, but requires more CPU time:

```
       MOVEA.L   #N<<11+A<<7+F<<2,A0    * A0=NAF to execute (bits 2 to 15)
       MOVE.W     #val,D0                    *  D0=  value  to  write  if write
function
       TRAP      #5
```

at exit, A0 holds always NAF, but the high word is destroyed, D0 holds the 16 bits value read in bits 0..15 if the function was a camac read function, and holds the qx response (CSR) in the bits 16..31 independantly of the performed function.

# A P P E N D I X E

## Abolute variables for the SMACC

The following magic values are known as entry-point so that they can be referenced either from high level languages, either from assembly language. For P+ users, a definition module SMACC__HARDWARE will be provided.

```
_CAMAC       EQU     $800000  Camac address space begin
*
*    _CORSET  defines a byte array and a write to any of its location will force
*             the corresponding bit in the COR and trigger the FEC
*    _CSRREAD defines a byte or a word used to read either only the CiR, either
*             the CSR and the camac QX+N response
*    _CSRTSET defines a byte used only to reset the communication input register
*
_CORSET      EQU     $C000C0  Set Bit 0 of COR
_CSRREAD     EQU     $C00120  Read Communication status register
_CSRTSET     EQU     $C00100  Test and reset CSR
*
*     following variables define bytes and any write to those locations will
*            result in the corresponding action
*
_FPI1RST     EQU     $C00140  Reset Front Panel Interrupt 1
_FPI2RST     EQU     $C00142  Reset Front Panel Interrupt 2
_FPI3RST     EQU     $C00144  Reset Front Panel Interrupt 3
_FPI4RST     EQU     $C00146  Reset Front Panel Interrupt 4
_FPI4ON      EQU     $C00180  Enable Front Panel Interrupt 4
_FPI4OFF     EQU     $C001C0  disable Front Panel Interrupt 4
*
*  following symbols are used to define vector numbers when  connecting
*            interrupt service routines:
*
_V_FPI1  EQU      $67      vector number to connect to fpi1
_V_FPI2  EQU      $66      vector number to connect to fpi2
_V_FPI3  EQU      $65      vector number to connect to fpi3
_V_FPI4  EQU      $64      vector number to connect to fpi4
_V_IRQ1  EQU      $61      vector number to connect to Int.RQ1
_V_IRQ2  EQU      $60      vector number to connect to Int.RQ2
_V_LAM1  EQU      $41      vector number to connect to LAM1
_V_LAM2  EQU      $42      vector number to connect to LAM2
_V_LAM3  EQU      $43      vector number to connect to LAM3
_V_LAM4  EQU      $44      vector number to connect to LAM4
_V_LAM5  EQU      $45      vector number to connect to LAM5
_V_LAM6  EQU      $46      vector number to connect to LAM6
_V_LAM7  EQU      $47      vector number to connect to LAM7
_V_LAM8  EQU      $48      vector number to connect to LAM8
_V_LAM9  EQU      $49      vector number to connect to LAM9
_V_LAM10 EQU      $4A      vector number to connect to LAM10
_V_LAM11 EQU      $4B      vector number to connect to LAM11
_V_LAM12 EQU      $4C      vector number to connect to LAM12
_V_LAM13 EQU      $4D      vector number to connect to LAM13
_V_LAM14 EQU      $4E      vector number to connect to LAM14
_V_LAM15 EQU      $4F      vector number to connect to LAM15
_V_LAM16 EQU      $50      vector number to connect to LAM16
_V_LAM17 EQU      $51      vector number to connect to LAM17
_V_LAM18 EQU      $52      vector number to connect to LAM18
_V_LAM19 EQU      $53      vector number to connect to LAM19
_V_LAM20 EQU      $54      vector number to connect to LAM20
_V_LAM21 EQU      $55      vector number to connect to LAM21
_V_LAM22 EQU      $56      vector number to connect to LAM22
_V_LAM23 EQU      $57      vector number to connect to LAM23
```

The CERN standard programming convention must be used each time it is possible.

The NODAL for the MC68000 was designed and written before the definition of the CERN programming convention for the MC68000 and does not use this standard.

Four new NODAL function types have been added to handle the STANDARD call interface with NODAL:

- Type 28 : standard call
- type 29 : call with return D0 = returned integer value.
- type 30 : call with implicit returned real parameter
- type 31 : call with implicit returned string parameter

A variable or function can only be found by NODAL only if a NODAL header with the following structure does exists:

```
DC.W  10         * length (=10 words for function)
DC.W  28         * function type = 28, 29, 30 or 31
DC.B  'name'     * name (6 char padded with nulls)
DC.L  $xxxxxxxx  * parameters description
JMP.L yyyy       * JMP to entry point
```

Parameter descriptors are eigth 4-BIT fields, rigth justified
Parameter types are given by field values, ZERO terminating

1 = (RV) address of a 48 bits read-only real value
2 = (IV) address of a 32 bits read-only integer value
3 = (SV) address of a string value, (can take concatenation)
4 = (NR) address of a NODAL header
5 = (RR) address of a real
6 = (IR) address of a read-write 32 bits integer
7 = (RA) starting address of a 48 bits real array
8 = (IA) starting address of a 16 bits integer array
9 = (NM) address of a 3 words NODAL name
10= (LA) starting address of a 32 bits integer array

The normal way of parameter passing is always a "call by reference" either for P+, either for NODAL.

A string consists of a string header and a buffer to contain the the information representing the string. The format of the string header is as follows:

```
HEADER  DC.L  start address of the buffer holding the string
        DC.W  current read pointer, relative to startaddress
        DC.W  current write pointer, relative to startaddress
        DC.W  length of buffer allocated to hold string
```

# A P P E N D I X G

## Example of writing a NODAL compatible routine

We try to write an assembly routine callable from NODAL by:

```
SET a=DEMO(p1,p2,p3)
```

doing the following work:

```
SET k=p1*2          all comuptation done on 32bits integer values
SET p3=K+p2
SET VALUE=k         result=k
```

This function get 3 parameters 32 bits integer p1,p2 and p3.
- p1 and p2 are read-only values while p3 is a read/write parameter.
- a 32 bits integer value is returned.

A standard CERN calling sequence can be used.

The program looks at follow:

```
        IDENT     demo       * identification of resulting binary
        TTL       'Example of NODAL function'
        SYSTEXT              * define macros for CERN standard  calling  *
**  Integer  function  DEMO(p1,p2,p3)      :  NODAL  header * NODAL_HEADER
section r
        dc.w      10           * header length = 10 words
        dc.w      29           * function type = 29
        dc.w      'DE','MO',0  * function name
        dc.l      $00000622    * p3 = RW integer, p1 and p2 = RO integer
        jmp.l      DEMO          *  jump to the effective entry point * **
Program part: * PROGRAM  section  r  *  **  declare  parameter  names  and
volumes : *
        S_param   p1,4,p2,p4,p3,4 * ** declare local variables : *
        S__local   k,4 * ** entry point declaration and stack reservation
for local variables: *
        S__enter    DEMO * * parameters and local variable access is done
through A6 *
        movea.l   p1(a6),a0 * read p1 value
        move.l    (a0),d0

        asl.l     #1,d0      * compute p1*2
        move.l    d0,k(a6)   * save result in k local variable

        movea.l   p2(a6),a0 * compute k+p2
        add.l     (a0),d0

        movea.l   p3(a6),a1 * store result in p3
        move.l    d0,(a1)

        move.l    k(a6),d0   * restore k value in D0 for return

        S_return             * standard exit of the reoutine
        end
```

# A P P E N D I X   H

## Example of testing an ISR from NODAL

The following example can be followed to write and test an ISR:

The Assembly program looks as follow:

```
NODAL_HEADER SECTION R
             dc.w     10,29,'V_','FP','I1',0,0
             jmp.l    get_V_fpi1

PROGRAM      SECTION R
             S_ENTER  get_V_fpi1      * this integer nodal function returns
             extern   _V_FPI1         * the value of the FPI1 vector number
             MOVEQ    #_V_FPI1,DO
             S_RETURN

NODAL_HEADER SECTION R
             dc.w     10,29,'I_','FP','I1',0,0
             jmp.l    get_I_fpi1

PROGRAM      SECTION R
             S_ENTER  get_I_fpi1      * this integer nodal function returns
             MOVE.l   #ISR_FPI1,DO    * the value of the ISR entry point
             S_RETURN

PROGRAM      SECTION R
             entry    ISR_FPI1        * at input only AO=vector number
ISR_FPI1     equ      *               *           and A1=parameter of CISR
             ...                      *           are valid
             ...
             MOVEQ    #x,DO           * choise the ISR exit mode
             TRAP     #1              * exit from ISR
```

We can now test this ISR, starting it from interactive NODAL:

Attach interactive NODAL (NODI) to the global segment where ISR is located:

```
1.20 SET AD=0 ;% init return physical address
1.21 @ATTACH-A-SEGMENT,,,[[2000,,'PROG',,,AD
```

Allocate an ASQ to NODAL to be able to receive bus error return message and also end of ISR messages code 2.

```
1.30 @ALLOCATE-ASYNCHRONOUS-SERVICE-QUEUE,,,1,24,5*24,0
```

Connect Interrupt service routine :

```
1.40 SET arg.=xxxx     % input parameter for the ISR
1.41 @CONFIGURE-INTERRUPT-SERVICE-ROUTINE,,,0,V_FPI1,I_FPI1,arg.
```

You can now read an event - if any - with the following NODAL code:

```
10.10 DIM-LONG EVT.AR(10)
10.11 SET EVT.AR(0)=0
10.12 @READ-EVENT EVT.AR
10.13 IF EVT.AR(0)=0; TYPE 'No event recived';RETURN
10.20 ... event treatement
10.99 GO 10.11    ;% try to loop on the ASQ if another message did arrive
```

You can also wait for such event or for a time-out with the command:

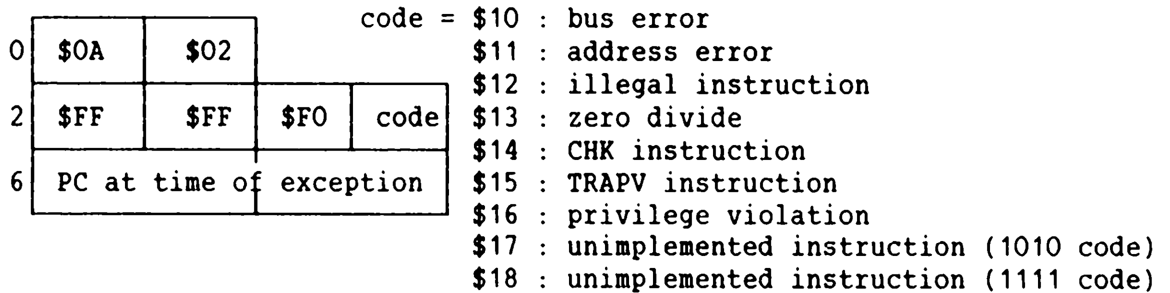
```
2.10 SET TO=n*1000 ;% define time-out = n seconds
2.11 @DELAY-AND-WAIT TO
2.12 DO10          ;% try to read and treat the message
2.13 GO 2.11       ;% loop to wait for next message
```

# A P P E N D I X  I

## STANDARD EVENTS STRUCTURE

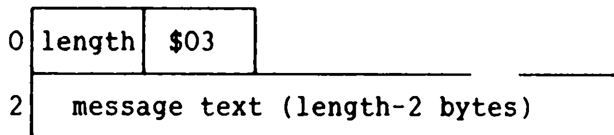
This Appendix shows the detailled format of RMS68K defined event messages wich a task can receive in its ASQ. One of the event messages (code $03) originates in a user task. In this latter case, RMS68K adds some fields to the event so that the format is different for the sender and the receiver.

- Return from ISR Event (code $02) This event is returned to a task when one of its ISR's has encountered an exception (bus error,...)

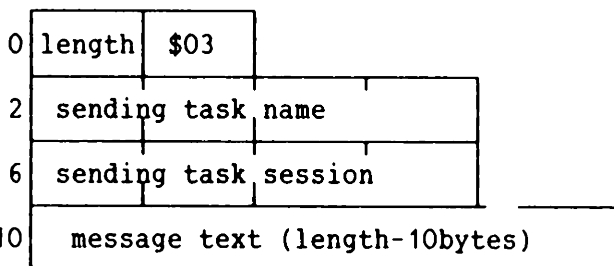



- User task created event (code $03) This message originates in a user task and is sent to a user task. The text of the message can be any format which has been agreed upon by sending and receiving tasks.

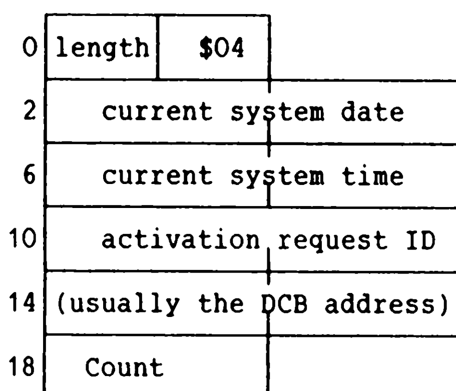
Event sent:




Event received:




- Time-out event (code $04) This event originates in RMS68K when a task is to receive an event as the result of the previously issued RQSTPA directive.




If no request ID was supplied, the message is restricted to 10 bytes.

length=$0A or $10

The activation count is incremented by 1 each time an event is queued.

- Sub-task termination event (code $05) This event is sent to the sub-task's monitor when a task terminates.

| Offset | Field | | Description |
|---|---|---|---|
| 0 | $18 | 05 | |
| 2 | Task Name and session of sub-task | | |
| 10 | Action initiating Task Name & session | | |
| 18 | Term code | $00 | Termination code = 1 (Normal) or 2 (abnormal) |
| 20 | Abort code | | Lower 2 bytes of A0 on ABORT or TERMT or RMS68K abort code ($80xx) |
| 22 | User Abort code | | Upper 2 bytes of register D0 on ABORT or TERMT ($0000 if RMS68K aborts the task) |

- Server task request Event (code $07) When a task request the service of a server task through the use of a trap instruction, this event is sent to the server task:

| Offset | Field | |
|---|---|---|
| 0 | $18 | $07 |
| 2 | trap | prior |
| 4 | requesting task's name | |
| 8 | requesting task's session | |
| 12 | user gen. ID | (see CRTCB procedure) |
| 14 | requesting task's D0 | |
| 18 | requesting task's A0 | |
| 22 | PBST | PBSZ |

PBST = parameter block status (0=total moved, 1=partially moved, 2=bad parameter block 3=no parameter block requested).
PBSZ = parameter block size in bytes for the block which follow.

- Exception monitor event (code $08) This event is sent to the exception monitor task when a task is attached or detached from it, or when a task is halted.

| Offset | Field | |
|---|---|---|
| 0 | $0C | $08 |
| 2 | target task's name | |
| 6 | target task's session | |
| 10 | code | type |

Type = $01 : attach
$02 : detach
$03 : exception event (see code)
code = $00 to $0F : trap #n
$10 to $18 : idem as for event code $02
$1B : maximum count reached
$1C : traced 1 instruction
$1D : value change occured
$1E : value equal occured

# A P P E N D I X  J

## RMS68K usefull internal tables

The following appendix describe the structure of some internal tables of RMS68K useful for debug:

- Task Control Block
- Asynchronous Service Queue
- User Semaphore table
- Periodic activation table.

1) <u>TCB : Task control block</u> The TCB is used to control the execution of the relevant task

| Offset | | | | |
|---|---|---|---|---|
| $00 | ! T C B | | Next TCB addr | |
| $08 | Next TCB of the group | | Next ready TCB addr | |
| $10 | Task Name | | Task session | |
| $18 | Monitor task name and session | | | |
| $20 | Semaphore wait link | | CPRI | LPRI | RPRI | IOCNT |
| $28 | attributes | abort code | current state | |
| $30 | task's TST semaphore | | | |
| $36 | task's TST pointer | | | |
| $3A | task'a ASQ semaphore | | | |
| $40 | task's ASQ pointer | | unused | |
| $48 | Exception vector ptr | | trap vector pointer | |
| $50 | unused | | | |
| $58 | addr of delay in PAT | | UPDO | ISRS |
| $60 | unused | | | |
| $68 | unused | | task's entry point | |
| $70 | User number | SSP | UTRP | |

CPRI current priority
LPRI limit priority
RPRI priority to enter task in ready list
IOCNT count of pending I/O

see RSTATE primitive for state
see TSKATTR primitive for attributes

UPDO upper 1/2 of DO on trap 1's
ISRS ISR error code for wakeup

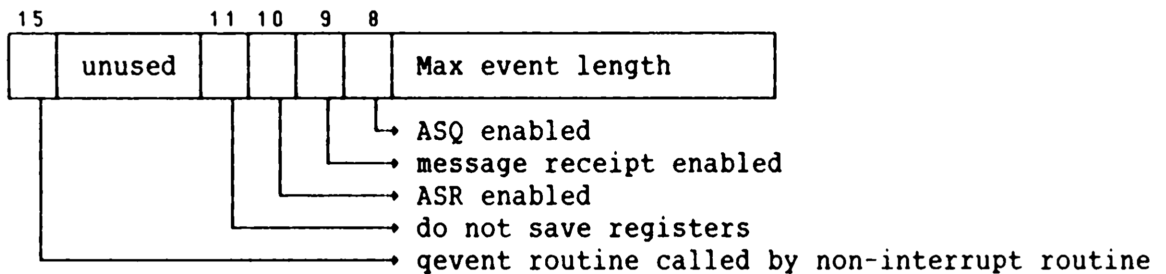
SSP exec stack depth
UTRP user trap number set when trap occurs

$74 registers DO..A6 to be retored when the task will be dispatched
$BO Termination task name and session
$B8 BERR info placed on stack by bus or address error
$FA user's SR
$FC user's PC
$100 registers DO..D7,AO..A7 save
$140 Exception monitor parameters
$160 Begin of TST (task's segments table)

2) Asynchronous event queue

| Offset | | |
|---|---|---|
| $00 | ! A S Q | ASQ status |
| $06 | ASR entry address | reserved |
| $0E | USP when enter ASR | current service addr |
| $16 | queue begin addr | queue end addr |
| $1E | Get pointer | Put pointer |
| $26 | EVCNT | EVCNT number of events currently in queue |

ASQ status:




3) Periodic activation table

Header:

| Offset | | |
|---|---|---|
| $00 | ! P A T | 1st free entry |
| $08 | 1st in task list | 1st in exec list |
| $10 | table length | |

PAT entry definition:

| Offset | | | | |
|---|---|---|---|---|
| $00 | Ptr to next entry | TCB addr of task | | |
| $08 | Time of next activ | activ interval | | |
| $10 | ASR address | options | | see RQSTPA options |
| $16 | activ request ID | count | IT level | IT level only for RMS |

4) <u>User semaphore table</u>

Header:

| | | | | |
|---|---|---|---|---|
| $00 | ! U S T | | next table link | |
| $08 | NSEG | NPAGE | MENT | CENT |
| $10 | 1rst entry addr | | | |

NSEG number of segments in table
NPAGE number of pages - this segment
MENT max number of entries
CENT current number of entries

UST entry definition:

| | | | | | |
|---|---|---|---|---|---|
| $00 | originator's task name and session | | | | |
| $08 | semaphore name | | UCNT | xcnt | type |
| $10 | semaphore or ptr to sem. | | | | |

UCNT # users of semaphore or
-1 if pointer entry
XCNT semaphore count
TYPE semaphore type (1,2 or 3)

# A P P E N D I X K

## RMS68K SYSGEN PARAMETERS

The system initializer task contains a data area, which receives SYSGEN parameters.

ASN = 4 : Number of address spaces

GST = 1 : Global segment table length (14 entries/page)

UST = 4 : User semaphore table length (11 entries/page)

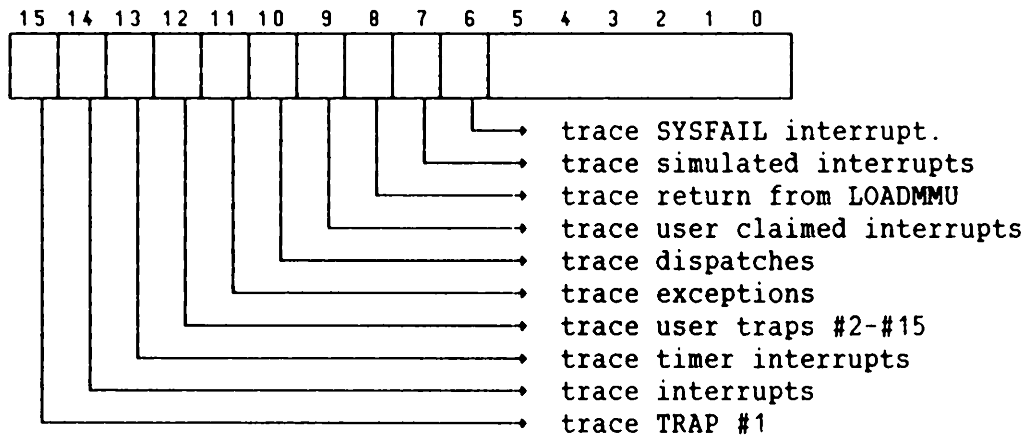
UDR = 1 : User directive table length (25 entries/page)

PAT = 2 : Periodic activation table length ( 8 entries/page)

IOV = 6 : I/O vector table length (12 entries/page)

TRACE =16 : Trace table length (20 entries/page)

TRCFLAG =$0040 : System trace flag <i> .




TIMINTV =100 : the time interval (in milliseconds) between timer interrupts <ii> .

TIMSLIC = 2 : The number of timer interrupts allowed before a task is forced to relinquish the processor.

BUGTRAC = 0 : The address of the trace routine in the firmware debugger.

TCBLST = 0 : maximum number of TCB

TCBRDY = 0 : maximum number of TCB in the "ready" state.

The memory partitionning is the following :

- type = RAM addresses : $00000-$07000 clear addresses : $01000-$07000
- type = RAM addresses : $07000-$50000 clear addresses : $XXXXX-$50000

---

<i> The user must be able to change this value.
<ii> This value should be 10 or 20ms.

- type = RAM addresses : $80000-$C0000 clear addresses : none

# A P P E N D I X L

## REFERENCES

1) MC68000 16-Bit Microprocessor User's Manual
3rd Edition, MC68000UM(AD3), Motorola Inc, 1982.

2) M68000 16/32 bit microprocessor Programmer's reference manual.
(MOTOROLA).
This manual describes the general hardware and software environnement due to the fact the central processor is a MC68000 microprocessor.

3) User's Manual of the SMACC - A MC68000 Based Autonomous Crate Controller.
W.Heinze. (CERN/PS/CO/Note 84-24 - 22.1.1985).
This note is a complete description of the hardware of the SMACC and serves as basic document for maintenance as well for systems and applications programmers.

4) M68000 Family : Real-time Multi-tasking Software User's Manual.
(MOTOROLA M68KRMS69K/D8)

5) P-plus version B Reference Manual and MC68000 implementation.....

6) DG2 interim report.
(CERN/DD/KIK-X/R1).

7) Prototype Implementation of Datagram service.
B.Carpenter. (CERN/DD/KIK-X/W10)

8) MC68000 Cross Software.
K.Osen. (CERN/PS/CO/Note-84-09 - 12.12.1984).

9) Software support for Motorola 68000 microprocessor at CERN - M68MIL Cross Macro Assembler.
H. von Eicken (CERN 83-12)
[also available through WYLBUR by by typing HELP WRITEUP M68MIL]

10) MC68000 Cross Macro Assembler Reference Manual
3rd Edition, M68KXASM(D3), Motorola Inc, 1979.

11) A simple SYSGEN facility for RMS68K.
T.Sharning. (CERN/PRIAM - RMS68K/M1 - 04.07.1984).

12) CUFOM The CERN universal format for object modules.
J.Montuelle. (CERN/DD/US/84 - 16.02.1982).

13) The P+ Cluster Layout in MC68000.
K.Osen. (CERN/PS/CO/WP-84-043 - 25.06.1984).

14) The CUFOM PROCESSORS User's Manual.
J.Montuelle. (CERN/DD/US/83 - 16.02.1982).

15) MoniCa, A Symbolic Debugging Monitor for the M68000
H. von Eicken (CERN/DD/???).

16) Super ACC protocol using serial CAMAC.
(CERN/PS/CO/WP-82-30 - 17.05.1985).

17) Auxiliary Controllers and their role in the PS Control System.
P.N.Clout. (CERN/PS/CO/Note-82-11 - 15.04.1982).

18) Implantation via NODAL d'un interpreteur de commandes systemes oriente RMS.
G.Cuisinier. (CERN/PS/CO/NOTE-85-??? - 16.07.1985 ).

19) Logging des erreurs fatales.
G.Cuisinier. (CERN/PS/CO/WP 85-??? 15.03.1985).

20) RMS68K dans le SMACC : Taches en temps reel critique.
G.Cuisinier. (CERN/PS/CO/WP-85-031 - 11.04.1985)

21) Mesures a l'oscilloscope des temps de traitement du mecanisme ISR sous RMS.
G.Cuisinier. (CERN/PS/CO/WP-85-042 - 14.05.1985)

22) Service datagrammes : design du protocol de bas niveau.
G.Cuisinier. (CERN/PS/CO/WP-85-047 - 20.05.1985).

23) MTT15 EXORMACs OPERATING SYSTEMS COURSE.
(MOTOROLA - 05.1984).

24) MacIntrotte.
F.di Maio. (CERN/PS/CO/Note 85-01 - 26.02.1985)
This note describes the use of the MacNodal system on the Macintosh computer for test or control of CAMAC equipments in the stand-alone mode.

# APPENDIX M

## GLOSSARY

ACC :auxiliary crate controler.

An :Address register number n. There are eight address registers: A0,A1,A2,A3,A4,A5,A6,A7. Each register has 32 bits.

A7 :This address register is referred to as the system stack pointer, SP. The following instructions has implicit references to A7: JSR, BSR, RTS, LINK and UNLK. There are two versions of A7. One version is visible while the processor is in the user state, and the other is visible when the processor is in the supervisor state. Events which change the execution state of the processor therefore also appearently change the contents of A7. Such events include the TRAP and RTE instructions.

Assembly :This is a symbolic program specification from which the generated bit pattern is 100% predictable (a one to one correspondence).

Asynchronous service queue (ASQ) :A FIFO used for management of event messages between RMS68K and a task. The ASQ can be used by a task to treat events synchronously or asynchronously.

Asynchronous service routine (ASR) :A part of a task's program code which asynchronously processes event messages in the task's ASQ. The ASR operates in a software interrupt mode.

Autovector :MC68000 family microprocessor interrupt caused exception vector. There are seven autovectors, corresponding to seven levels of interrupt priority.

Call by reference :This is a parameter passing method which copies the address of the actual parameter into the parameter list. Call by reference must be used if the called routine shall write into the actual parameter. It is also convenient to use call by reference on read only variables if they are very large.

CUF :File type of CUFOM files in SINTRAN.

CUFOM :CERN Universal Format for Object Modules.

Dn :Data register n. There are eight data registers: D0,D1,D2,D3,D4,D5,D6,D7. Each register has 32 bits.

Dynamic link :The dynamic link of the current stack frame points to the dynamic link in the stack frame belonging to the invocation of the routine which called the currently executing routine. The dynamic link is the data equivalent to the return address.

Exception monitor task :A task which can monitor one or more other tasks and be notified by any excption which occur whitin those tasks.

Exception vector :A memory location from which the MC68000 family fetches the address of a routine which is to handle an exception.

Executive directive :A request by a task for services of monitor.

Global variables :These variables are situated on a memory segment by themselves. They are not placed in the working area. The global variables must not be confused with the local variables in the global stack frame.

Interrupt Service Routine (ISR) :A part of a task's program code which handles interrupts. The ISR operates in an asynchronous mode with the task.

Local variables :These variables are allocated inside a stack frame in the working area.

MC68000 :This is a 16 bit microprocessor standardised at CERN. The MPU has 16 multipurpose registers each 32 bits wide. The address bus has 24 bits, allowing the access of 16777216 bytes.

Monitor task :A task which receives automatic notification upon the termination of one or more other tasks, called sub-tasks of the monitor task.

MSB :Most significant bit. The bit number of MSB depends on the operand size.

MSC :File type of Motorola S-code files in SINTRAN.

M68MIL Assembler :A DD/CERN assembler for MC68000 which generates CUFOM output.

Multitasking :An operation mode where more than one functionally bound task is beeing precessed concurrently.

PC :Program counter. This special register has 32 bits. During the execution of an instruction, PC points to the next instruction.

Procedure :A routine without a function value.

Process :Code which can be executed by a task.

Rn :Register number n. This notation is used when any address or data register can be used.

Routine :A routine is a common term for procedure and function.

Segment :A block of memory which can be used for data, program code, or an ASQ. Every task can consist up to four segments. Segments may be shared by more than one task.

Semaphore :A unit representing a count of signals wich is used for synvhronizing task activity or controlling the use of ressources.

Server task :A task which operates much like an extension of RMS68K, and which can provide a service to any task in the system upon request.

Session :A group of related tasks, identified by a session number.

SP :System stack pointer. This another name for address register A7, both in user and supervisor state.

SSP :Supervisor stack pointer. This is another name for A7 while the computer is in the supervisor state.

Static level :In a nested block-structured language such as Pascal, the level of nesting of a routine in the source file. For practical reasons, the outermost level is 2, so a doubly-nested routine is at level 4. Static levels 3 to 31 are valid (numbering is from 1, 2 is not necessary).

Static vector :Pointer to the currently relevant stack frame for a certain static level. 29 such pointers are stored in the global area; when a routine on static level N is entered, it saves the previous static vector N and replaces it with A6.

Variables which are neither global nor local are accessed via the static vectors.

**Supervisor hardware state** :A privileged state of MC68000 family microprocessor execution. This state is used by the operating system, and allows use of the privileged instructions. All exception handling takes place in the supervisor state.

**System task** :A task which operates in the user hardware state of the MC68000 family microprocessor, but is protected from user tasks.

**Task** :A functionnaly bound group of one or more modules which can operate concurrently whith other tasks.

**Task Name** :A means of identifying a particular task within a session : in the PS use of RMS68K, a task name is a 4 characters name.

**Task Priority** :A relative level of importance given to a task. Task which are more urgent are assigned higher priorities.

**Trap vector** :A particlar type of MC6800 family microprocessor exception vector, corresponding to MC68000 TRAP instructions.

**User hardware state** :Normal programs execute in the user state of the MPU. This state forbids use of the privileged instructions.

**User task** : A task which operates in the user harware state of the MC68000 family microprocessor.

**user vector** : A particular type of the MC68000 family microprocessor exception vector. They may be assigned by the user.

**USP** :User stack pointer. This is another name for A7 while the computer is in the user state.

# Index

ABORT . . . 4-6, 8, 10, 23, 38, 45, 46, 64, 65.
AKRQST . . . 20, 74.
ASQ . . . 3, 4, 11, 12, 14, 17-21, 24, 31, 45, 72, 108, 111, 129, 131.
ASR . . . 11-13, 17, 18, 24, 72-74, 129.
ATSEM . . . 14-16, 71.
ATTSEG . . . 9, 10, 67.
CISR . . . 24, 45, 77.
cluster . . . 31, 125.
CRSEM . . . 14-16, 71.
CRTCB . . . 4-6, 19, 34, 63.
CUFLINK . . . 49, 54.
CUFMERG . . . 49.
CUFOM . . . 49, 52, 125, 126, 129, 130.
datagram . . . 23, 30, 37, 39, 40, 42, 44, 125.
DCLSHR . . . 9, 10, 68.
DEASQ . . . 12, 72.
delay . . . 4, 17, 18, 35, 57, 73.
DELAYW . . . 4, 17, 18, 73.
DELIVERY . . . 42, 54.
DERQST . . . 20, 74.
DESEG . . . 9, 10, 69.
DESEM . . . 14, 16, 71.
DESEMA . . . 14, 16, 71.
DEXMON . . . 21, 77.
dormant . . . 4-6, 8-10, 21.
DSERVE . . . 20, 75.
event . . . 4, 11-14, 17-21, 24, 45, 72-74, 108, 111, 112, 116, 129.
existent . . . 4.
EXMMSK . . . 21, 75.
EXMON . . . 21, 75.
EXPVCT . . . 23, 77.
GETBL . . . 41.
GTASQ . . . 12, 45, 72.
GTDTIM . . . 17, 73.
GTSEG . . . 9, 30, 66.
IMAGE . . . 41, 49, 50, 52, 56, 57.
ISR . . . 24, 25, 34, 45, 56, 77, 95, 105, 107, 108, 111, 126, 130.
LAM . . . 45.
LDACC . . . 31, 40, 41.

M68MIL . . . . . . . . . . . . . . . . . . . . . . . . . . . 48-50, 125, 130.
monitor . . . . . . . . . . . . . . . . . . . . . . . . . . . 2, 19, 21, 22, 34, 46, 51, 56, 63, 64, 75-77, 112, 126, 130.
MOVELL . . . . . . . . . . . . . . . . . . . . . . . . . . . 26, 78.
MOVEPL . . . . . . . . . . . . . . . . . . . . . . . . . . . 26, 78.
NODAL . . . . . . . . . . . . . . . . . . . . . . . . . . . 1, 2, 27, 30, 33-39, 41, 44, 46-48, 51, 56, 58, 59, 79, 81-88, 100, 101, 103, 105, 107, 108, 126.
periodic activation . . . . . . . . . . . . . . . . . . . . 4, 17, 18, 57, 73, 115, 116, 121.
PPL . . . . . . . . . . . . . . . . . . . . . . . . . . . 50.
PPLUS . . . . . . . . . . . . . . . . . . . . . . . . . . . 39, 49, 61, 63-78, 125.
PREPUSH . . . . . . . . . . . . . . . . . . . . . . . . . . . 49.
PSTATE . . . . . . . . . . . . . . . . . . . . . . . . . . . 22, 75, 76.
PUSHER . . . . . . . . . . . . . . . . . . . . . . . . . . . 49, 53.
PUTBL . . . . . . . . . . . . . . . . . . . . . . . . . . . 41.
QEVNT . . . . . . . . . . . . . . . . . . . . . . . . . . . 12, 72.
RCVSA . . . . . . . . . . . . . . . . . . . . . . . . . . . 9, 10, 70.
RDEVNT . . . . . . . . . . . . . . . . . . . . . . . . . . . 12, 72.
ready . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-7, 16, 34, 121.
RELINQ . . . . . . . . . . . . . . . . . . . . . . . . . . . 5, 7, 64.
reset . . . . . . . . . . . . . . . . . . . . . . . . . . . 30, 32, 33, 51.
RESUME . . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 5, 7, 18, 65.
REXMON . . . . . . . . . . . . . . . . . . . . . . . . . . . 22, 77.
RPC . . . . . . . . . . . . . . . . . . . . . . . . . . . 43, 44.
RQSTPA . . . . . . . . . . . . . . . . . . . . . . . . . . . 17, 18, 73, 111.
RSTATE . . . . . . . . . . . . . . . . . . . . . . . . . . . 22, 76.
RTEVNT . . . . . . . . . . . . . . . . . . . . . . . . . . . 13.
running . . . . . . . . . . . . . . . . . . . . . . . . . . . 3-5, 26, 28, 29, 31, 35, 37, 47, 56, 58-60, 63.
semaphore . . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 14-16, 41, 57, 71, 81, 115, 117, 121, 131.
SERVER . . . . . . . . . . . . . . . . . . . . . . . . . . . 12, 19, 20, 23, 44, 74, 75, 112, 131.
SETASQ . . . . . . . . . . . . . . . . . . . . . . . . . . . 12, 72.
SETPRI . . . . . . . . . . . . . . . . . . . . . . . . . . . 5, 7, 65.
SGSEM . . . . . . . . . . . . . . . . . . . . . . . . . . . 14, 16, 71.
SHRSEG . . . . . . . . . . . . . . . . . . . . . . . . . . . 9, 10, 68.
SINT . . . . . . . . . . . . . . . . . . . . . . . . . . . 25, 78.
SNAPTRAC . . . . . . . . . . . . . . . . . . . . . . . . . . . 26.
STACC . . . . . . . . . . . . . . . . . . . . . . . . . . . 40.
START . . . . . . . . . . . . . . . . . . . . . . . . . . . 5, 6, 19, 34, 38, 44, 51, 56, 59, 60, 64, 100.

STDTIM . . . . . . . . . . . . . . . . . . . . . . . . . . 17, 73.
STOP . . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 5, 8, 31, 56, 58, 64.
SUSPND . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 5, 7, 64.
TCB . . . . . . . . . . . . . . . . . . . . . . . . . . . 3, 6, 8, 26, 59, 76, 115, 121.
TERM . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-6, 10, 64, 131.
TERMT . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 5, 8, 10, 64.
TRACC . . . . . . . . . . . . . . . . . . . . . . . . . . 41.
TRPVCT . . . . . . . . . . . . . . . . . . . . . . . . . . 23, 77.
TRSEG . . . . . . . . . . . . . . . . . . . . . . . . . . 9, 10, 69.
TSK . . . . . . . . . . . . . . . . . . . . . . . . . . . 21, 50.
TSKATTR . . . . . . . . . . . . . . . . . . . . . . . . . 26, 75.
TSKINFO . . . . . . . . . . . . . . . . . . . . . . . . . 26, 76.
wait . . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 5, 7, 16, 18, 34, 37, 56, 64.
    for . . . . . . . . . . . . . . . . . . . . . . . . . 4, 12, 16, 17, 21, 42, 45, 57, 72, 73, 108.
    on . . . . . . . . . . . . . . . . . . . . . . . . . . 14-16, 71.
wakeup . . . . . . . . . . . . . . . . . . . . . . . . . . 4, 5, 7, 18, 24, 65.
    pending . . . . . . . . . . . . . . . . . . . . . . . 7, 18.
WTEVNT . . . . . . . . . . . . . . . . . . . . . . . . . . 12, 18, 72.
WTSEM . . . . . . . . . . . . . . . . . . . . . . . . . . 14-16, 71.