powerpc

Sam Jordan

COLLABORATORS			
	TITLE : powerpc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Sam Jordan	February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

powerpc

1.1 autodocs for powerpc.library

powerpc.library/AllocSignalPPC

powerpc.library/AllocVecPPC

powerpc.library/AllocXMsgPPC

powerpc.library/AttemptSemaphorePPC

powerpc.library/ChangeMMU

powerpc.library/ClearExcMMU

powerpc.library/CmpTimePPC

powerpc.library/CopyMemPPC

powerpc.library/CreateMsgPortPPC

powerpc.library/CreateTaskPPC

powerpc.library/DeleteMsgPortPPC

powerpc.library/DeleteTaskPPC

powerpc.library/EndSnoopTask

powerpc.library/EnqueuePPC

powerpc.library/FindNamePPC

powerpc.library/FindPortPPC

powerpc.library/FindSemaphorePPC

powerpc.library/FindTagItemPPC

powerpc.library/FindTaskPPC

powerpc.library/FindTaskByID

powerpc.library/FreeAllMem

powerpc.library/FreeSemaphorePPC

powerpc.library/FreeSignalPPC

powerpc.library/FreeVecPPC

powerpc.library/FreeXMsgPPC

powerpc.library/GetHALInfo

powerpc.library/GetInfo

powerpc.library/GetMsgPPC

powerpc.library/GetSysTimePPC

powerpc.library/GetTagDataPPC

powerpc.library/InitSemaphorePPC

powerpc.library/InsertPPC

powerpc.library/LockTaskList

powerpc.library/ModifyFPExc

powerpc.library/NextTagItemPPC

powerpc.library/ObtainSemaphorePPC

powerpc.library/PutMsgPPC

powerpc.library/PutXMsgPPC

powerpc.library/ReleaseSemaphorePPC

powerpc.library/RemExcHandler

powerpc.library/RemHeadPPC

powerpc.library/RemovePPC

powerpc.library/RemPortPPC

powerpc.library/RemTailPPC

powerpc.library/RemSemaphorePPC

powerpc.library/ReplyMsgPPC

powerpc.library/Run68K

powerpc.library/SetCache

powerpc.library/SetExcHandler

powerpc.library/SetExcMMU

powerpc.library/SetHardware

powerpc.library/SetNiceValue

powerpc.library/SetReplyPortPPC

powerpc.library/SetScheduling

powerpc.library/SetSignalPPC

powerpc.library/SetTaskPriPPC

powerpc.library/Signal68K

powerpc.library/SignalPPC

powerpc.library/SnoopTask

powerpc.library/SPrintF

powerpc.library/SubTimePPC

powerpc.library/Super

powerpc.library/UnLockTaskList

powerpc.library/User

powerpc.library/WaitFor68K

powerpc.library/WaitPortPPC

powerpc.library/WaitPPC

powerpc.library/WaitTime

1.2 allocvec32

powerpc.library/AllocVec32 powerpc. ↩
library/AllocVec32

```
  NAME
AllocVec32 - allocates memory which is correctly aligned (V7)

  CPU
680x0

  SYNOPSIS
memblock = AllocVec32(memsize, attributes)
d0                    d0       d1

void *AllocVec32(ULONG, ULONG);

  FUNCTION
This function allocates memory via exec/AllocVec and aligns the
memory block properly, so that this memory block can be shared
with PPC tasks. The minimal alignment of the memory block is 32.

  INPUTS
memsize - size of memory to be allocated
attributes - the desired memory attributes (see exec/AllocMem for
       a description of these attributes)

  RESULT
memblock - The address of the allocated memory block

  NOTES
```

```
Memory blocks allocated with 'AllocVec32' must be freed using
'FreeVec32'.

  SEE ALSO

              FreeVec32
              , exec/AllocMem
```

1.3 allocxmsg

```
              powerpc.library/AllocXMsg                                powerpc. ↩
                  library/AllocXMsg

  NAME
AllocXMsg - allocates a message for Inter-CPU communication (V12)

  CPU
680x0

  SYNOPSIS
message = AllocXMsg(bodysize, replyport)
d0                  d0        a0

struct Message *AllocXMsg(ULONG, struct MsgPort *);

  FUNCTION
This function allocates memory for a message which can be used
for Inter-CPU communication. Some fields of the message are
initialized.
After this function was called, the message body must be
created before sending this message.

  INPUTS
bodysize - the size of the message body (max. 65535-MN_SIZE)
replyport - the reply port

  RESULT
message - The address of an initialized message (except for
    the message body, which must be initialized by the
    programmer).

  NOTES
Calling this function is the only way allowed to create a
message which can be sent to a PPC task.

A message allocated with 'AllocXMsg' should be freed using
'FreeXMsg' if it is not used anymore. Since V14, it is
allowed to free the messsage using 'FreeXMsgPPC' on the
PPC side (which is internally done using a cross call).

An Inter-CPU message must be sent with 'PutXMsg' to a PPC
task.

It is possible not to specify a replyport (simply set
```

```
replyport to NULL).

If you want to be compatible to earlier versions, you
shouldn't free InterCPU messages by the foreign task.

  SEE ALSO

              FreeXMsg
              ,
              PutXMsg
```

1.4 freevec32

```
              powerpc.library/FreeVec32                              powerpc. ↩
                 library/FreeVec32

  NAME
FreeVec32 - frees memory allocated with 'AllocVec32' (V7)

  CPU
680x0

  SYNOPSIS
FreeVec32(memblock)
    a1

void FreeVec32(void *);

  FUNCTION
This function frees a memory block which was allocated using
'AllocVec32'.

  INPUTS
memblock - The address of the allocated memory block

  SEE ALSO

              AllocVec32
```

1.5 freexmsg

```
              powerpc.library/FreeXMsg                               powerpc. ↩
                 library/FreeXMsg

  NAME
FreeXMsg - frees a message allocated with 'AllocXMsg' (V12)

  CPU
680x0
```

```
  SYNOPSIS
FreeXMsg(message)
   a1

void FreeXMsg(struct Message *);

  FUNCTION
This function frees a memory allocated using 'AllocXMsg'.

  INPUTS
message - a message allocated by 'AllocXMsg'.

  NOTES
There were some restrictions in earlier versions using
FreeXMsg. Since V14, a XMessage created by AllocXMsg can be
freed either by another 68K task (if the message isn't used
anymore) or by a PPC task using FreeXMsgPPC.

  SEE ALSO

              AllocXMsg
              ,
              PutXMsg
```

1.6 getcpu

```
powerpc.library/GetCPU                                  powerpc.library/GetCPU

  NAME
GetCPU - gets the PowerPC CPU type (V7)

  CPU
680x0

  SYNOPSIS
CPUType = GetCPU
d0

ULONG GetCPU (void);

  FUNCTION
This function reads the PowerPC CPU type. A longword is returned with
one specific bit set (see the include file 'powerpc/powerpc.i'
resp. 'powerpc/powerpc.h' for a description of the different
CPU types)

  RESULT
CPUType - A longword with one specific bit set.
```

1.7 getppcstate

```
powerpc.library/GetPPCState                          powerpc.library/GetPPCState

    NAME
  GetPPCState - returns the state of the PPC and PPC applications (V13)

    CPU
  680x0

    SYNOPSIS
  PPCState = GetPPCState
  d0

  ULONG GetPPCState (void);

    FUNCTION
  This function returns the current state of the PPC processor and
  the state of custom applications. A bitmask is returned with the
  values defined in 'powerpc.i' resp. 'powerpc.h'.

    RESULT
  PPCState - A bitmask. The following bits are supported (the
       description is valid if the bit is 1):

       PPCSTATEF_POWERSAVE  - PPC is currently in power save mode.
       PPCSTATEF_APPACTIVE  - PPC application tasks are currently
          - active resp. installed in the system.
       PPCSTATEF_APPRUNNING - At least one PPC application task is
            ready or running.
```

1.8 powerdebugmode

```
powerpc.library/PowerDebugMode                    powerpc.library/PowerDebugMode

    NAME
  PowerDebugMode - sets the level of debugging output (V7)

    CPU
  680x0

    SYNOPSIS
  PowerDebugMode (debuglevel)
      d0

  void PowerDebugMode (ULONG);

    FUNCTION
  The powerpc.library has a built-in debugging system which prints out
  many informations to the serial port. The main purpose of this
  function is to improve the maintenance of this library. If problems
  occur with the powerpc.library then it will help the author of this
  library a lot to fix the problems. Try to reproduce the problems with
  debugging output enabled and send the debugging output to the author.
  Use a program which captures the data transferred to the serial port
  (for example Sushi) to save the debugging output.
```

```
  INPUTS
debuglevel - Debugging level (0-3). All other values are ignored.
       The higher the debugging level, the larger the debugging
       output. 0 means no debugging output.

  NOTES
The powerpc.library operates with debugging level 0 by default (no
debugging output). You can change the default value with the
environment variable 'powerpc/debug' (set values from 0 to 3).
```

1.9 putxmsg

```
              powerpc.library/PutXMsg                                    powerpc. ↩
                  library/PutXMsg

  NAME
PutXMsg - sends an Inter-CPU message to a PPC task (V12)

  CPU
680x0

  SYNOPSIS
PutXMsg(MsgPortPPC, message)
  a0          a1

void PutXMsg(struct MsgPortPPC *, struct Message *);

  FUNCTION
This function sends an Inter-CPU message allocated by 'AllocXMsg'
to a PPC task.

  INPUTS
MsgPortPPC - a PPC message port
message - a message allocated by 'AllocXMsg'.

  NOTES
Inter-CPU must NOT be used for internal communication. They can
only be used for communication between tasks on different processors.

Inter-CPU messages get a different node type, if they are
sent. If you want to filter out Reply-Messages from standard
or Inter-CPU messages, compare the LN_TYPE field to NT_REPLYMSG.
Replied Inter-CPU messages still get the same node type
(NT_REPLYMSG). Any assumptions about the value of the new
node type are ILLEGAL!!

As soon as an Inter-CPU message is sent, the 68K looses ownership
over the message. No access to the message is allowed until the
reply has been arrived. If no replyport was specified, it's
allowed to free the message, after it was read from the other side.

Inter-CPU messages can be reused if they have been replied.

Inter-CPU messages are read and replied using the standard
```

```
message handling mechanisms (exec/WaitPort,exec/GetMsg,exec/ReplyMsg
for 68K, powerpc/WaitPortPPC, powerpc/GetMsgPPC, powerpc/ReplyMsgPPC
for PPC).

Don't call exec/ReplyMsg with an InterCPU-Message without
Replyport (versions less than V12.2 crashed).

The receiving task must NOT access message data, which are not
explicitely located in the message body (for example data which is
referenced by a pointer) unless both tasks care for the cache
consistency. Only the message itself is flushed/invalidated
automatically by the system.

The receiving task may write to the message body of an Inter-CPU
message.

  SEE ALSO

              AllocXMsg
              ,
              FreeXMsg
```

1.10 runppc

```
              powerpc.library/RunPPC                                  powerpc. ↩
                  library/RunPPC

  NAME
RunPPC - runs a PowerPC function (V7)

  CPU
680x0

  SYNOPSIS
status = RunPPC(PPStruct)
d0              a0

LONG RunPPC (struct PowerPC *);

  FUNCTION
Runs a PowerPC function. A mirror PPC process is created. All
registers can be transferred to PPC as well as parameters on
stack. All cache management actions are handled automatically.

All registers are transferred back from PPC after the PPC call
is completed. They are stored in the PowerPC structure.

The register assignment is as follows:

d0      <->     r3              fp0     <->     f1
d1      <->     r4              fp1     <->     f2
d2      <->     r22             fp2     <->     f3
d3      <->     r23             fp3     <->     f4
d4      <->     r24             fp4     <->     f5
```

```
  d5      <->     r25             fp5     <->     f6
  d6      <->     r26             fp6     <->     f7
  d7      <->     r27             fp7     <->     f8
  a0      <->     r5
  a1      <->     r6
  a2      <->     r28
  a3      <->     r29
  a4      <->     r2
  a5      <->     r30
  a6      <->     r31

  Please note, that these registers are NOT transferred directly but
  in the register array mentioned above (PP_REGS).

  Here follows another table from the PPC's point of view:

  PPC-Register:                   Index into the register array:
  ----------------------------------------------------------------
  Base register:
  r2                              12

  Scratch registers:
  r3                              0
  r4                              1
  r5                              8
  r6                              9

  Nonvolatile registers:
  r22                             2
  r23                             3
  r24                             4
  r25                             5
  r26                             6
  r27                             7
  r28                             10
  r29                             11
  r30                             13
  r31                             14

INPUTS
  PPStruct - Pointer to an initialized PowerPC Structure

    PP_CODE       : Pointer to the PowerPC code
    PP_OFFSET     : Not used until V12.2 of powerpc.library. From
        V12.3 on, the PP_OFFSET field is used just like
        at Run68K. If PP_OFFSET is zero, than the code
        pointed to by PP_CODE is executed, if PP_OFFSET
        is not zero, a PPC library function is executed
        with PP_CODE containing the library base and
        PP_OFFSET containing the library vector offset.
    PP_FLAGS      : Flags which can be ore'd together
    - PP[F/B]_ASYNC   : Call PPC function asynchronely (68K process
            returns immediately)
    PP_STACKPTR   : Pointer to the arguments on the stack to be
        transferred. The pointer must point directly
        to the first argument, not to the return address!
        If no arguments on stack should be transferred,
```

```
      set this to zero.
    PP_STACKSIZE  : Size of the stack area to be transferred. If no
      stack parameters should be transferred, set this
      to zero.
    PP_REGS       : Array of longwords where the registers to be
      transferred can be stored (d0-a6). Please see
      above for the exact placement of these registers.
    PP_FREGS      : Array of quadwords (8 bytes) where the FP-registers
      to be transferred can be stored (fp0-fp7). fp0 is
      at offset 0, fp1 at offset 8 etc.

    RESULT
  status - PPERR_SUCCESS if the call was successfully
     PPERR_ASYNCERR if a synchrone PPC call was made after an
      asynchrone PPC call

    NOTES
  Calling a PPC function asynchronely is dangerous. Take care of
  possible cache conflicts. Avoid calling system functions as much
  as possible.

  If an asynchrone PPC call is done, the function WaitForPPC MUST be
  called ONCE after the call was done. No other PPC call is allowed
  for this 68K process after an asynchrone PPC call and before a call
  to WaitForPPC.

  If an asynchronely called PPC function performs a 68K call, the call
  is only performed when WaitForPPC is called by the 68K process. Note
  that the PPC mirror process is still connected to the calling 68K
  process.

  DON'T pass arguments on stack when calling a PPC function asynchronely.
  The stack is most likely to be trashed before it is copied to the PPC
  stack.

  Assembler programmers should use the macros RUNPOWERPC and
  RUNPOWERPC_XL located in the include file 'powerpc/powerpc.i'

    SEE ALSO

                WaitForPPC
                , powerpc/powerpc.i, powerpc/powerpc.h
```

1.11 sprintf68k

```
powerpc.library/SPrintF68K                          powerpc.library/SPrintF68K

    NAME
  SPrintF68K - prints a formatted string to the serial port (V7)

    CPU
  680x0

    SYNOPSIS
```

```
SPrintF68K (Formatstring, values )
      a0            a1

void SPrintF68K (STRPTR, APTR);

  FUNCTION
Prints a formatted string to the serial port using the AMIGA-OS
functions 'exec/RawPutChar' and 'exec/RawDoFmt'. Can be used to add
debugging features and to improve the maintenance of software.

  INPUTS
Formatstring - A C style string with % commands to indicate where
         parameters have to be inserted (see 'exec/RawDoFmt'
         for a detailed description of these commands).
values - A pointer to an array of parameters to be inserted into
   specified places in the string.

  SEE ALSO
exec/RawDoFmt
```

1.12 waitforppc

```
              powerpc.library/WaitForPPC                              powerpc. ↩
                  library/WaitForPPC

  NAME
WaitForPPC - waits for the completion of an asynchrone PPC call (V7)

  CPU
680x0

  SYNOPSIS
status = WaitForPPC(PPStruct)
d0                  a0

LONG WaitForPPC (struct PowerPC *);

  FUNCTION
After an asynchrone PPC call was done (see RunPPC) this function must
be called to wait for the completion of the PowerPC function. All
registers transferred to the PowerPC with RunPPC are returned into the
PowerPC Structure.

  INPUTS
PPStruct - Pointer to a PowerPC Structure (see RunPPC for a description
     of the elements). The structure has not to be initialized.
     The structure must be transferred to hold the returned
     registers by the PPC function.

  RESULT
status - PPERR_SUCCESS if the call was successfully
   PPERR_WAITERR if WaitForPPC is called after a synchrone PPC
           call.

  NOTES
```

```
Assembler programmers should use the macros WAITFORPPC and
WAITFORPPC_XL located in the include file 'powerpc/powerpc.i'

  SEE ALSO

              RunPPC
              , powerpc/powerpc.i, powerpc/powerpc.h
```

1.13 addheadppc

```
              powerpc.library/AddHeadPPC                              powerpc. ↩
                  library/AddHeadPPC

  NAME
AddHeadPPC - insert a node at the head of a list (V8)

  CPU
PowerPC

  SYNOPSIS
AddHeadPPC(_PowerPCBase, list, node)
     r3            r4    r5

void AddHeadPPC(struct Library *, struct List*, struct Node*);

  FUNCTION
Insert a node to the head of a standard exec list. This is the
mirror function of exec/AddHead.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
list - a pointer to the target list
node - the node to insert

  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

Assembler programmers may use the macro _ADDHEAD located in
'powerpc/listsPPC.i'

This function is safe to call from exception handlers.

  SEE ALSO

              InsertPPC
              ,
              AddTailPPC
              ,
              RemovePPC
              ,
              RemHeadPPC
              ,
              RemTailPPC
```

,

EnqueuePPC
,
FindNamePPC
, powerpc/listsPPC.i

1.14 addportppc

powerpc.library/AddPortPPC powerpc. ↩
library/AddPortPPC

```
  NAME
AddPortPPC - adds a public PPC message port to the system (V11)

  CPU
PowerPC

  SYNOPSIS
AddPortPPC(_PowerPCBase, MsgPortPPC)
     r3            r4

void AddPortPPC(struct Library *, struct MsgPortPPC *);

  FUNCTION
This function adds a PPC message port to a public list of ports.
The message port should be named and the priority field should
be set. A public message port can be found by name using
'FindPortPPC'. This is the mirror function of exec/AddPort.

  INPUTS
_PowerPCBase - base of powerpc.library
MsgPortPPC - pointer to a PPC message port. It's safe to pass a
       NULL parameter.

  SEE ALSO
```

CreateMsgPortPPC
,
DeleteMsgPortPPC
,
FindPortPPC
,
RemPortPPC
powerpc/portsPPC.i, powerpc/portsPPC.h

1.15 addsemaphoreppc

powerpc.library/AddSemaphorePPC powerpc.library/ ↩
AddSemaphorePPC

```
  NAME
AddSemaphorePPC - initializes a global signal semaphore (V8)

  CPU
PowerPC

  SYNOPSIS
status = AddSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
r3                       r3            r4

LONG AddSemaphorePPC(struct Library *, struct SignalSemaphorePPC *);

  FUNCTION
Initializes a signal semaphore and adds it to the public semaphore
list. This is the mirror function of exec/AddSemaphore.

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure
         (a semaphore name should be specified)

  RESULT
status - status value:
    SSPPC_SUCCESS: function was successful
    SSPPC_NOMEM: function failed due to lack of memory

  SEE ALSO

              InitSemaphorePPC
              ,
              FreeSemaphorePPC
              ,
              ObtainSemaphorePPC
              ,

              AttemptSemaphorePPC
              ,
              ReleaseSemaphorePPC
              ,
              RemSemaphorePPC
              ,

              FindSemaphorePPC
              , powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h
```

1.16 addtailppc

```
              powerpc.library/AddTailPPC                          powerpc. ↩
                  library/AddTailPPC

  NAME
AddTailPPC - insert a node at the tail of a list (V8)
```

```
  CPU
PowerPC

  SYNOPSIS
AddTailPPC(_PowerPCBase, list, node)
     r3            r4    r5

void AddTailPPC(struct Library *, struct List*, struct Node*);

  FUNCTION
Insert a node to the tail of a standard exec list. This is the
mirror function of exec/AddTail.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
list - a pointer to the target list
node - the node to insert

  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

Assembler programmers may use the macro _ADDTAIL located in
'powerpc/listsPPC.i'

This function is safe to call from exception handlers.

  SEE ALSO

             InsertPPC
             ,
             AddHeadPPC
             ,
             RemovePPC
             ,
             RemHeadPPC
             ,
             RemTailPPC
             ,

             EnqueuePPC
             ,
             FindNamePPC
             , powerpc/listsPPC.i
```

1.17 addtimeppc

```
             powerpc.library/AddTimePPC                          powerpc. ↩
                library/AddTimePPC

  NAME
AddTimePPC - adds one time request to another (V7)

  CPU
```

```
PowerPC

  SYNOPSIS
AddTimePPC(_PowerPCBase, Dest, Source)
     r3            r4    r5

void AddTimePPC(struct Library *, struct timeval *, struct timeval *);

  FUNCTION
This routine adds one timeval structure to another. The results are
stored in the destination (Dest + Source -> Dest)

This is the mirror function of timer/AddTime.

  INPUTS
_PowerPCBase - base of powerpc.library
Dest - pointer to a timeval structure
Source - pointer to a timeval structure

  NOTES
This function is safe to call from exception handlers

  SEE ALSO

              GetSysTimePPC
              ,
              SubTimePPC
              ,
              CmpTimePPC
```

1.18 allocsignalppc

```
              powerpc.library/AllocSignalPPC                  powerpc.library/ ↩
                 AllocSignalPPC

  NAME
AllocSignalPPC - allocate a signal (V8)

  CPU
PowerPC

  SYNOPSIS
signalnum = AllocSignalPPC(_PowerPCBase, signalNum)
r3                         r3            r4

LONG AllocSignalPPC(struct Library *, LONG);

  FUNCTION
Allocate a signal bit from the current task's pool. Either a
particular bit or the next free bit may be allocated. This
is the mirror function of exec/AllocSignal.

  INPUTS
_PowerPCBase - base of powerpc.library
```

```
signalNum - the desired signal bit number (0..31) or -1
      if the next free bit should be allocated

  RESULT
signalnum - the signal bit allocated or -1 for failure.

  NOTES
IMPORTANT: The signal bit numbers are returned in the
68K notation! For example, if the number 27 is returned,
the waiting mask must be $08000000.

All signal allocations are kept coherent on both CPU's.
A signal allocated on the 68K side is not free anymore
for the mirror PPC task and vice versa. The PPC also
can wait for signals or send signals allocated on the
68K side and vice versa (V11).

  SEE ALSO

              FreeSignalPPC
              ,
              SetSignalPPC
              ,
              SignalPPC
              ,
              WaitPPC
```

1.19 allocvecppc

```
              powerpc.library/AllocVecPPC                          powerpc.library ↩
                 /AllocVecPPC

  NAME
AllocVecPPC - allocates memory for PPC with MMU support (V7)

  CPU
PowerPC

  SYNOPSIS
memblock = AllocVecPPC(_PowerPCBase, memsize, attributes, alignment)
r3                     r3            r4       r5          r6

void *AllocVecPPC(struct Library *, ULONG, ULONG, ULONG);

  FUNCTION
This function allocates memory which is correctly aligned for the use
by PowerPC applications. It's the mirror function of exec/AllocVec
but offers some additional features.
AllocVecPPC supports user defined alignment and allocation of memory-
blocks with a desired cache mode (MMU support only for V9+)
Since V12, it's possible to allocate memory, which is protected
against other tasks (either full- or write-protected).

  INPUTS
```

```
_PowerPCBase - base of powerpc.library
memsize - the amount of memory to be allocated.
attributes - the requirements as explained in exec/AllocMem
       This function offers some additional attributes:

    MEMF_WRITETHROUGH: maps the allocated memory as writethrough
    MEMF_COPYBACK:     maps the allocated memory as copyback
    MEMF_CACHEON:      maps the allocated memory as cachable
    MEMF_CACHEOFF:     maps the allocated memory as noncachable
    MEMF_GUARDED:      maps the allocated memory as guarded
    MEMF_NOTGUARDED:   maps the allocated memory as not guarded
    MEMF_BAT:          puts the allocated memory block into a
           BAT register
    MEMF_PROTECT:      the memory block should be full-protected
           against other tasks (no user-accesses of
           other tasks allowed).
    MEMF_WRITEPROTECT: the memory block should be write-protected
           against other tasks (no user-write-acesses
           of other tasks allowed).

alignment - the desired alignment of the memory block. The system
      may round this value up to a minimal alignment. It's
      safe to pass 0 as alignment.

  RESULT
memblock - The address of the allocated memory. If the
     memory couldn't be allocated 0 is returned.

  NOTES
The amount of memory effectively allocated is usually bigger
than the given memsize. It's not a good idea to call this
function many times to allocate very small pieces of memory.

If some of the additional memflags are specified, the alignment
and the size is internally rounded up to meet the requirements
of the MMU.

The additional MMU memflags are intended for highly optimizing
code and shoud not be used by standard applications.

The memflag MEMF_BAT can improve the performance of the memory
accesses heavily (especially on CPU's with software tablesearch).
But DON'T use this flag unless you really need the speed. Note:
The required free memory has to be much bigger than the size of
the memory to be allocated, because there are severe alignment
restrictions when using BAT registers.

The memflag MEMF_BAT has no effect if the task runs with BAT MMU
Setup.

Note that no other tasks should access memory which was allocated
using special MMU memflags, because the other task can probably
run under a different MMU setup which can cause cache problems,
if the other one writes to the same memory in copyback mode
while this task accesses the data in noncachable mode, for
example.
```

```
The MMU support ist not implemented in powerpc.library <= V9.
The memory protection support ist not implemented in
powerpc.library <= V11.

  BUGS
Before V14, allocations > 512KB could fail.

  SEE ALSO

              FreeVecPPC
              ,
              FreeAllMem
              , powerpc/memoryPPC.i, powerpc/memoryPPC.h
```

1.20 allocxmsgppc

```
              powerpc.library/AllocXMsgPPC                        powerpc.library/ ↩
                 AllocXMsgPPC

  NAME
AllocXMsgPPC - allocates a message for Inter-CPU communication (V12)

  CPU
PowerPC

  SYNOPSIS
message = AllocXMsgPPC(_PowerPCBase, bodysize, replyport)
r3                     r3            r4        r5

struct Message *AllocXMsgPPC(struct Library *, ULONG, struct MsgPort *);

  FUNCTION
This function allocates memory for a message which can be used
for Inter-CPU communication. Some fields of the message are
initialized.
After this function was called, the message body must be
created before sending this message.

  INPUTS
_PowerPCBase - base of powerpc.library
bodysize - the size of the message body (max. 65535-MN_SIZE)
replyport - the reply port

  RESULT
message - The address of an initialized message (except for
    the message body, which must be initialized by the
    programmer).

  NOTES
Calling this function is the only way allowed to create a
message which can be sent to a 68K task.

A message allocated with 'AllocXMsgPPC' should be freed using
'FreeXMsgPPC' if it is not used anymore. Since V14, it is
```

```
allowed to free the messsage using 'FreeXMsg' on the
68K side (which is internally done using a cross call).

An Inter-CPU message must be sent with 'PutXMsgPPC' to a 68K
task.

It is possible not to specify a replyport (simply set
replyport to NULL).

If you want to be compatible to earlier versions, you
shouldn't free InterCPU messages by the foreign task.

  SEE ALSO

              FreeXMsgPPC
              ,
              PutXMsgPPC
```

1.21 attemptsemaphoreppc

```
              powerpc.library/AttemptSemaphorePPC        powerpc.library/ ↩
                  AttemptSemaphorePPC

  NAME
AttemptSemaphorePPC - try to obtain without blocking (V8)

  CPU
PowerPC

  SYNOPSIS
status = AttemptSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
           r3            r4

LONG AttemptSemaphorePPC(struct Library *,
        struct SignalSemaphorePPC *);

  FUNCTION
Tries to get exclusive access to a signal semaphore. If the semaphore
is locked by another task, this function returns with an appropriate
status value. This is the mirror function of exec/AttemptSemaphore

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure

  RESULT
status - status value:
    ATTEMPT_SUCCESS: operation successful
    ATTEMPT_FAILURE: semaphore couldn't be locked

  NOTES
This call is guaranteed to preserve all GPR (except r0 and r3)
and the CTR.
```

```
    SEE ALSO

                InitSemaphorePPC
                ,
                FreeSemaphorePPC
                ,
                ObtainSemaphorePPC

                ReleaseSemaphorePPC
                ,
                AddSemaphorePPC
                ,
                RemSemaphorePPC

                FindSemaphorePPC
                , powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h
```

1.22 changemmu

```
powerpc.library/ChangeMMU                              powerpc.library/ChangeMMU

    NAME
  ChangeMMU - changes the MMU setup of the current task (V10)

    CPU
  PowerPC

    SYNOPSIS
  ChangeMMU(_PowerPCBase, MMUMode)
      r3            r4

  void ChangeMMU(struct Library *, ULONG);

    FUNCTION
  Changes the MMU setup of the currently running task. A task is able
  to run with two different MMU setups:

  - paged MMU setup: The standard method, where almost every memory
    access is controlled by the page table
  - BAT MMU setup: Almost all the memory is controlled by the 4 BAT
    registers.

    INPUTS
  _PowerPCBase - base of powerpc.library
  MMUMode - CHMMU_STANDARD: change MMU setup to standard
      CHMMU_BAT     : change MMU setup to BAT setup

    NOTES
  This function should usually NOT be called. It is intended for
  highly optimizing code and should only be used, if enough MMU
  knowledge is present.

  The state of the current task can be changed from the shell by
  using the tool 'changemmu' (and this is the better way how to
```

```
change the setup rather than calling the library function)

  SEE ALSO
powerpc/tasksppc.i, powerpc/tasksPPC.h
```

1.23 clearexcmmu

```
              powerpc.library/ClearExcMMU                          powerpc.library ↩
                  /ClearExcMMU

  NAME
ClearExcMMU - removes the temp. MMU setup installed by SetExcMMU (V10)

  CPU
PowerPC

  SYNOPSIS
ClearExcMMU(_PowerPCBase)
      r3

void ClearExcMMU(struct Library *);

  FUNCTION
This function is for exception handlers only. It removes the temporary
BAT based MMU setup, which was installed using SetExcMMU. The old MMU
state is restored.

  INPUTS
_PowerPCBase - base of powerpc.library

  NOTES
This function must not be called from anywhere else than from
an exception handler.

  SEE ALSO

              SetExcMMU
```

1.24 cmptimeppc

```
              powerpc.library/CmpTimePPC                            powerpc. ↩
                  library/CmpTimePPC

  NAME
CmpTimePPC - compares two timeval structures (V7)

  CPU
PowerPC

  SYNOPSIS
result = CmpTimePPC(_PowerPCBase, Dest, Source)
```

```
  r3                  r3            r4    r5

  LONG CmpTimePPC(struct Library *, struct timeval *, struct timeval *);

    FUNCTION
  This routine compares two timeval structures.

  This is the mirror function of timer/CmpTime.

    INPUTS
  _PowerPCBase - base of powerpc.library
  Dest - pointer to a timeval structure
  Source - pointer to a timeval structure

    RESULT
  0  - if both timeval structures are equal
  -1 - if Dest is greater than Source
  1  - if Dest is less than Source


    NOTES
  This function is safe to call from exception handlers

    SEE ALSO

                GetSysTimePPC
                ,
                AddTimePPC
                ,
                SubTimePPC
```

1.25 copymemppc

```
powerpc.library/CopyMemPPC                          powerpc.library/CopyMemPPC

    NAME
  CopyMemPPC - copies memory the fastest way possible (V12)

    CPU
  PowerPC

    SYNOPSIS
  CopyMemPPC(_PowerPCBase, source, dest, size)
       r3            r4      r5    r6

  void CopyMemPPC(struct Library *, void *, void *, ULONG);

    FUNCTION
  This function copies a source memory area to a destination memory
  area. No overlapping is supported. CopyMemPPC tries to copy with
  the highest bandwidth possible.

    INPUTS
  _PowerPCBase - base of powerpc.library
```

```
source - address of the source memory area
dest - address of the destination memory area
size - size of the memory area to be copied

  NOTES
The highest performance can be achieved if both memory areas have
a minimal alignment of 8.
```

1.26 createmsgportppc

```
               powerpc.library/CreateMsgPortPPC              powerpc.library/ ↩
                  CreateMsgPortPPC

  NAME
CreateMsgPortPPC - creates a new PPC message port (V11)

  CPU
PowerPC

  SYNOPSIS
MsgPortPPC = CreateMsgPortPPC(_PowerPCBase)
r3                            r3

struct MsgPortPPC *CreateMsgPortPPC(struct Library *);

  FUNCTION
This function creates a new PowerPC message port. This is the only
way allowed to create a PPC message port. It is the mirror function
of exec/CreateMsgPort.

  INPUTS
_PowerPCBase - base of powerpc.library

  RESULT
MsgPortPPC - pointer to a MsgPortPPC structure or NULL for failure

  NOTES
A PowerPC message port should be deleted using 'DeleteMsgPortPPC'
if it is not used anymore.

It's forbidden to access PPC message ports by the standard
exec message handling routines.

  SEE ALSO

              DeleteMsgPortPPC
              ,
              FindPortPPC
              ,
              AddPortPPC
              ,
              RemPortPPC
                powerpc/portsPPC.i, powerpc/portsPPC.h
```

1.27 createtaskppc

```
               powerpc.library/CreateTaskPPC                    powerpc.library/ ↩
                  CreateTaskPPC

  NAME
CreateTaskPPC - creates a new PPC task (V8)

  CPU
PowerPC

  SYNOPSIS
TaskPPC = CreateTaskPPC(_PowerPCBase, TagItems)
r3                      r3            r4

struct TaskPPC *CreateTaskPPC(struct Library *, struct TagItem *);

  FUNCTION
This function creates a new PPC task under control of the tags passed.
All memory (inclusive stack) is allocated automatically.

PPC tasks are similar to exec tasks (the first element of the TaskPPC
structure is an exec task structure). The scheduling of these tasks
works similar to exec, so a running task blocks all tasks with lower
priority.

  INPUTS
_PowerPCBase - base of powerpc.library
TagItems - pointer to a tagitem array. The following tags are
     supported:

    TASKATTR_CODE:      pointer to the entry point of the new task
      (MUST be specified)
    TASKATTR_EXITCODE:  pointer to the exit routine of the new task
    TASKATTR_NAME:      task name (MUST be specified)
    TASKATTR_PRI:       task priority (-128 ... 127). Default = 0.
    TASKATTR_STACKSIZE: the desired stack size. If this tag is omitted
      the default stack size will be 16K.
    TASKATTR_R2:        smalldata base of the PPC program
    TASKATTR_R3:
    ...
    TASKATTR_R10:       parameters to be passed to the new task
      in the specified registers
    TASKATTR_MOTHERPRI: the priority is taken from the currently
      running task (TASKATTR_PRI is ignored) (V9)
    TASKATTR_BAT:       lets the task run under BAT MMU setup by
      default (V10)

  RESULT
TaskPPC - pointer to a TaskPPC structure or NULL for failure

  NOTES
If a 68K application only wants to call a PPC function, it's
better to use Run68K instead of creating a new PPC task.

While a PPC task created by Run68K is always connected to
```

```
the calling 68K process, a task created by CreateTaskPPC is
completely independent. If such a task peforms 68K calls, a
new mirror process on the 68K side is created.

If an alternative exit code is specified (TASKATTR_EXITCODE)
the value passed in TASKATTR_R2 remains intact in this exit
code.

  SEE ALSO

              DeleteTaskPPC
              ,
              FindTaskPPC
              ,
              FindTaskByID
              ,
powerpc/tasksPPC.i, powerpc/tasksPPC.h
```

1.28 deletemsgportppc

```
              powerpc.library/DeleteMsgPortPPC              powerpc.library/ ↩
                 DeleteMsgPortPPC

  NAME
DeleteMsgPortPPC - deletes a PPC message port (V11)

  CPU
PowerPC

  SYNOPSIS
DeleteMsgPortPPC(_PowerPCBase, MsgPortPPC)
     r3            r4

void DeleteMsgPortPPC(struct Library *, struct MsgPortPPC *);

  FUNCTION
This function deletes a PowerPC message port created using
'CreateMsgPortPPC'. It is the mirror function of exec/DeleteMsgPort.

  INPUTS
_PowerPCBase - base of powerpc.library
MsgPortPPC - Pointer to the message port to delete. It's safe to
       pass NULL as parameter

  NOTES
Calling 'DeleteMsgPortPPC' is the ONLY way allowed to delete a PPC
message port.

  SEE ALSO

              CreateMsgPortPPC
              ,
              FindPortPPC
              ,
```

```
              AddPortPPC
              ,
              RemPortPPC
                powerpc/portsPPC.i, powerpc/portsPPC.h
```

1.29 deletetaskppc

```
              powerpc.library/DeleteTaskPPC                    powerpc.library/ ↩
                 DeleteTaskPPC

  NAME
DeleteTaskPPC - deletes a PPC task (V8)

  CPU
PowerPC

  SYNOPSIS
DeleteTaskPPC(_PowerPCBase, PPCTask)
        r3            r4

void DeleteTaskPPC(struct Library *, struct TaskPPC *);

  FUNCTION
Deletes a PPC task created by CreateTaskPPC.

  INPUTS
_PowerPCBase - base of powerpc.library
TaskPPC - PPC task to remove or NULL for self removal

  NOTES
It's not encouraged to delete other tasks. This function
should only be called with a NULL parameter to remove the
calling task itself.

The system may also remove an existing 68K mirror process
connected to the calling PPC task.

  SEE ALSO

              CreateTaskPPC
              ,
              FindTaskPPC
              ,
              FindTaskByID
              ,
powerpc/tasksPPC.i, powerpc/tasksPPC.h
```

1.30 endsnooptask

```
              powerpc.library/EndSnoopTask                          powerpc.library/ ↩
                  EndSnoopTask

  NAME
EndSnoopTask - stops monitoring a PPC task (V13)

  CPU
PowerPC

  SYNOPSIS
EndSnoopTask (_PowerPCBase, SnoopID)
        r3            r4

void EndSnoopTask (struct Library *, ULONG);

  FUNCTION
This function removes a callback job, which was installed using
powerpc/SnoopTask.

  INPUTS
_PowerPCBase - base of powerpc.library
SnoopID - The value returned by SnoopTask. It's safe to pass
    NULL as parameter (is handled as no-op)

  SEE ALSO

              SnoopTask
```

1.31 enqueueppc

```
              powerpc.library/EnqueuePPC                              powerpc. ↩
                  library/EnqueuePPC

  NAME
EnqueuePPC - inserts a node into a list sorted by priority (V8)

  CPU
PowerPC

  SYNOPSIS
EnqueuePPC(_PowerPCBase, list, node)
     r3            r4    r5

void EnqueuePPC(struct Library *, struct List*, struct Node*);

  FUNCTION
Inserts a node to a standard exec list based on the node priority.
In this way a list can be kept sorted by priority all the time.
New nodes will be inserted in front of the first node with a
lower priority. This is the mirror function of exec/Enqueue.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
```

```
list - a pointer to the target list
node - the node to enqueue

  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

This function is safe to call from exception handlers.

  SEE ALSO

              InsertPPC
              ,
              AddTailPPC
              ,
              AddHeadPPC
              ,
              RemovePPC
              ,
              RemHeadPPC
              ,

              RemTailPPC
              ,
              FindNamePPC
              , powerpc/listsPPC.i
```

1.32 findnameppc

```
              powerpc.library/FindNamePPC                          powerpc.library ↩
                  /FindNamePPC

  NAME
FindNamePPC - finds a node with given name (V8)

  CPU
PowerPC

  SYNOPSIS
node = FindNamePPC(_PowerPCBase, start, name)
r3                 r3            r4     r5

struct Node *FindNamePPC(struct Library *, struct List*, STRPTR);

  FUNCTION
Searches a list for a node with the given name. If multiple nodes
with same names should be found, this function can be called with
a node starting point.

  INPUTS
_PowerPCBase - base of powerpc.library
list - a list header or a node to start the searche (if node,
       this one is skipped)
name - the name of the node
```

```
  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

This function is safe to call from exception handlers.

  SEE ALSO

              InsertPPC
              ,
              AddTailPPC
              ,
              AddHeadPPC
              ,
              RemovePPC
              ,
              RemHeadPPC
              ,

              RemTailPPC
              ,
              EnqueuePPC
              , powerpc/listsPPC.i
```

1.33 findportppc

```
              powerpc.library/FindPortPPC                          powerpc.library ↩
                  /FindPortPPC

  NAME
FindPortPPC - finds a public PPC message port by name (V11)

  CPU
PowerPC

  SYNOPSIS
MsgPortPPC = FindPortPPC(_PowerPCBase, name)
r3                       r3            r4

struct MsgPortPPC* FindPortPPC(struct Library *, STRPTR);

  FUNCTION
This function will search the global list of PPC message ports
for a port with the given name. No arbritation is needed. This
is the mirror function of exec/FindPort.

  INPUTS
_PowerPCBase - base of powerpc.library
name - name of the PPC message port to search

  RESULT
MsgPortPPC - pointer to a PPC message port or NULL if it was not
       found.
```

```
  SEE ALSO

              CreateMsgPortPPC
              ,
              DeleteMsgPortPPC
              ,
              AddPortPPC
              ,
              RemPortPPC
                powerpc/portsPPC.i, powerpc/portsPPC.h
```

1.34 findsemaphoreppc

```
              powerpc.library/FindSemaphorePPC              powerpc.library/ ↩
                 FindSemaphorePPC

  NAME
FindSemaphorePPC - finds a public semaphore (V8)

  CPU
PowerPC

  SYNOPSIS
SignalsemaphorePPC = FindSemaphorePPC(_PowerPCBase, SemaphoreName)
r3                                    r3            r4

struct signalsemaphorePPC *FindSemaphorePPC(struct Library *, STRPTR);

  FUNCTION
Finds a public semaphore added to the system semaphore list by
AddSemaphore. This is the mirror function to exec/FindSemaphore.

  INPUTS
_PowerPCBase - base of powerpc.library
SemaphoreName - name of the semaphore to find

  RESULT
SignalsemaphorePPC - signal semaphore requested or 0 if it was not
         found

  SEE ALSO

              InitSemaphorePPC
              ,
              FreeSemaphorePPC
              ,
              ObtainSemaphorePPC
              ,

              AttemptSemaphorePPC
              ,
              ReleaseSemaphorePPC
              ,
```

AddSemaphorePPC
,

RemSemaphorePPC
, powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h

1.35 findtagitemppc

powerpc.library/FindTagItemPPC powerpc.library/ ↩
FindTagItemPPC

```
  NAME
FindTagItemPPC - scan a tag list for a specific tag (V8)

  CPU
PowerPC

  SYNOPSIS
tag = FindTagItemPPC(_PowerPCBase, tagValue, tagList)
r3                   r3            r4        r5

struct TagItem *FindTagItemPPC(struct Library *, ULONG,
             struct TagItem *);

  FUNCTION
Scans a tag list and returns a pointer to the first item with
ti_Tag matching the 'tagValue' parameter. This is the mirror
function of utility/FindTagItem.

  INPUTS
_PowerPCBase - base of powerpc.library
tagValue - tag value to search for
tagList - tag item list to search (may be NULL)

  RESULT
tag - a pointer to the item with ti_Tag matching 'tagValue'
      or NULL if no match was found.

  NOTES
This function is safe to call from exception handlers.

  SEE ALSO
```

GetTagDataPPC
,
NextTagItemPPC

1.36 findtaskbyid

```
              powerpc.library/FindTaskByID                        powerpc.library/ ↩
                 FindTaskByID

  NAME
FindTaskByID - evaluates the task address for a given task ID (V14)

  CPU
PowerPC

  SYNOPSIS
TaskPPC = FindTaskByID(_PowerPCBase, taskID)
r3                     r3            r4

struct TaskPPC *FindTaskByID(struct Library *, ULONG);

  FUNCTION
Evaluates the task address for a given task ID.

  INPUTS
_PowerPCBase - base of powerpc.library
taskID - the task's ID number

  RESULT
TaskPPC - Pointer to the PPCTask structure

  SEE ALSO

              CreateTaskPPC
              ,
              DeleteTaskPPC
              ,
              FindTaskPPC
              ,
powerpc/tasksPPC.i, powerpc/tasksPPC.h
```

1.37 findtaskppc

```
              powerpc.library/FindTaskPPC                      powerpc.library/ ↩
                 FindTaskPPC

  NAME
FindTaskPPC - finds a task by name (or find oneself) (V8)

  CPU
PowerPC

  SYNOPSIS
TaskPPC = FindTaskPPC(_PowerPCBase, Name)
r3                    r3            r4

struct TaskPPC *FindTaskPPC(struct Library *, STRPTR);

  FUNCTION
```

```
Tries to find a task with the given name (or the current task if
NULL is specified). This is the mirror function of exec/FindTask.

  INPUTS
_PowerPCBase - base of powerpc.library
Name - name of the task to find or NULL for the current task

  RESULT
TaskPPC - pointer to the TaskPPC structure or NULL if the task
    was not found

  NOTES
Be cautious that a task may be removed at any time, so the
pointer returned may not be valid anymore when used.

It's allowed to call FindTaskPPC with a NULL parameter from
an exception handler. In this case the interrupted task is
returned.

  SEE ALSO

             CreateTaskPPC
             ,
             FindTaskByID
             ,
             DeleteTaskPPC
             , powerpc/tasksPPC.i,
powerpc/tasksPPC.h
```

1.38 freeallmem

```
             powerpc.library/FreeAllMem                          powerpc. ↩
                 library/FreeAllMem

  NAME
FreeAllMem - frees all memory allocated by the calling task (V11)

  CPU
PowerPC

  SYNOPSIS
FreeAllMem(_PowerPCBase)
     r3

void FreeVecPPC(struct Library *);

  FUNCTION
Frees all memory which was allocated by the calling task. This is an
easy way to free the memory rather than calling FreeVecPPC for every
allocation made.

  INPUTS
_PowerPCBase - base of powerpc.library
```

```
  SEE ALSO

              AllocVecPPC
              ,
              FreeVecPPC
```

1.39 freesemaphoreppc

```
              powerpc.library/FreeSemaphorePPC              powerpc.library/ ↩
                 FreeSemaphorePPC

  NAME
FreeSemaphorePPC - frees a signal semaphore (V8)

  CPU
PowerPC

  SYNOPSIS
FreeSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
     r3            r4

void FreeSemaphorePPC(struct Library *, struct SignalSemaphorePPC *);

  FUNCTION
Frees a signal semaphore initialized by InitSemaphorePPC. There
is no similar function in exec.library!

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure

  SEE ALSO

              InitSemaphorePPC
              ,
              ObtainSemaphorePPC
              , AttemptSemaphorePPC,

              ReleaseSemaphorePPC
              ,
              AddSemaphorePPC
              ,
              RemSemaphorePPC
              ,

              FindSemaphorePPC
              , powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h
```

1.40 freesignalppc

```
              powerpc.library/FreeSignalPPC                    powerpc.library/ ↩
                 FreeSignalPPC

  NAME
FreeSignalPPC - frees a signal (V8)

  CPU
PowerPC

  SYNOPSIS
FreeSignalPPC(_PowerPCBase, signalNum)
        r3            r4

void FreeSignalPPC(struct Library *, LONG);

  FUNCTION
Frees a signal bit allocated by AllocSignalPPC. This is the
mirror function of exec/FreeSignal.

  INPUTS
_PowerPCBase - base of powerpc.library
signalNum - the signal bit number to free (0..31). It's
      safe to pass -1 as input parameter.

  SEE ALSO

              AllocSignalPPC
              ,
              SetSignalPPC
              ,
              SignalPPC
              ,
              WaitPPC
```

1.41 freevecppc

```
              powerpc.library/FreeVecPPC                       powerpc. ↩
                 library/FreeVecPPC

  NAME
FreeVecPPC - frees memory allocated by AllocVecPPC (V7)

  CPU
PowerPC

  SYNOPSIS
status = FreeVecPPC(_PowerPCBase, memblock)
r3                  r3            r4

LONG FreeVecPPC(struct Library *, void *);

  FUNCTION
Frees memory allocated by AllocVecPPC. This is the mirror function
```

```
of exec/FreeVec.

  INPUTS
_PowerPCBase - base of powerpc.library
memblock - address of memory to be freed. It's safe to pass NULL
     as input parameter.

  RESULT
status - a status value:
    MEMERR_SUCCESS - operation was successful

  NOTES
It is absolutely required that no part of the freed memory is
located in the 68K data cache! This is only important if your
application is working asynchronously and shares data between
independent 68K and PPC tasks. Otherwise (for standard synchron
applications) it is guaranteed that the 68K data cache doesn't
contain parts of the freed memory.

The same rules also apply to the reverse case (free memory
using exec/FreeVec).

  SEE ALSO

              AllocVecPPC
              ,
              FreeAllMem
```

1.42 freexmsgppc

```
              powerpc.library/FreeXMsgPPC                          powerpc.library ↩
                 /FreeXMsgPPC

  NAME
FreeXMsgPPC - frees a message allocated with 'AllocXMsgPPC' (V12)

  CPU
PowerPC

  SYNOPSIS
FreeXMsgPPC(_PowerPCBase, message)
       r3           r4

void FreeXMsgPPC(struct Library *, struct Message *);

  FUNCTION
This function frees a memory allocated using 'AllocXMsgPPC'.

  INPUTS
_PowerPCBase - base of powerpc.library
message - a message allocated by 'AllocXMsgPPC'.

  NOTES
There were some restrictions in earlier versions using
```

```
  FreeXMsgPPC. Since V14, a XMessage created by AllocXMsgPPC can be
  freed either by another PPC task (if the message isn't used
  anymore) or by a 68K task using FreeXMsg.

    SEE ALSO

                AllocXMsgPPC
                ,
                PutXMsgPPC
```

1.43 gethalinfo

```
powerpc.library/GetHALInfo                          powerpc.library/GetHALInfo

    NAME
  GetHALInfo - evaluates some HAL related information (V14)

    CPU
  PowerPC

    SYNOPSIS
  GetHALInfo(_PowerPCBase, HALInfoTagList)
       r3            r4

  void GetHALInfo(struct Library *, struct TagItem *);

    FUNCTION
  This function is able to evaluate some information related to
  the WarpUp-HAL' status. The values which should be evaluated are
  specified using the appropriate tags (defined in powerpc/powerpc.i)
  and the value is returned in the appropriate ti_Data field.

    INPUTS
  _PowerPCBase - base of powerpc.library
  HALInfoTagList - pointer to a tagitem array. The following tags are
       supported:
      HINFO_ALEXC_HIGH:   returns the high-longword of a 64-bit-value
        showing the number of emulated alignment
        exceptions since the PowerPC was reset.
      HINFO_ALEXC_LOW:    returns the low-longword of a 64-bit-value
        showing the number of emulated alignment
        exceptions since the PowerPC was reset.

    NOTES
  If the number of emulated alignment exceptions should be evaluated
  for specific tasks, you should use the tc_Switch/tc_Launch fields
  of the WarpOS task structure to keep track of task switches.

    SEE ALSO
  powerpc/powerpc.i, powerpc/powerpc.h
```

1.44 getinfo

```
powerpc.library/GetInfo                                powerpc.library/GetInfo

    NAME
  GetInfo - evaluates many CPU related information (V10)

    CPU
  PowerPC

    SYNOPSIS
  GetInfo(_PowerPCBase, PPCInfoTagList)
    r3            r4

  void GetInfo(struct Library *, struct TagItem *);

    FUNCTION
  This function is able to evaluate many CPU related information
  such as CPU type, cache states and more. The values which should
  be evaluated are specified using the appropriate tags (defined
  in powerpc/powerpc.i) and the value is returned in the appropriate
  ti_Data field.

    INPUTS
  _PowerPCBase - base of powerpc.library
  PPCInfoTagList - pointer to a tagitem array. The following tags are
       supported:
      PPCINFO_CPU:        evaluates the PowerPC CPU type (see powerpc/
        powerpc.i for a description of the possible
        return values)
      PPCINFO_PVR:        returns the PVR register which also contains
        the revision number of the CPU besides the
        CPU type.
      PPCINFO_ICACHE:     returns the state of the instruction cache
        See powerpc/powerpc.i for a description of
        the possible values.
      PPCINFO_DCACHE:     returns the state of the data cache. The
        values possible are the same as for PPCINFO_
        ICACHE.
      PPCINFO_PAGETABLE:  returns the location of the standard page table
      PPCINFO_TABLESIZE:  returns the size of the page table (in KBytes)
      PPCINFO_BUSCLOCK:   returns the bus clock value (in Hz)
      PPCINFO_CPUCLOCK:   returns the CPU clock value (in Hz)

    NOTES
  The CPU clock cannot be evaluated on PowerPC systems without the
  extension E (for example PPC603,PPC604) because the supervisor
  register HID1 is missing. In this case, 0 is returned.

  Usually the PPC-CPU's available for AMIGA will have this register
  (for example: PPC603E, PPC604E)

    SEE ALSO
  powerpc/powerpc.i, powerpc/powerpc.h
```

1.45 getmsgppc

```
               powerpc.library/GetMsgPPC                                powerpc. ↩
                   library/GetMsgPPC

  NAME
GetMsgPPC - get next message from a message port (V11)

  CPU
PowerPC

  SYNOPSIS
message = GetMsgPPC(_PowerPCBase, MsgPortPPC)
r3                  r3            r4

struct Message *GetMsgPPC(struct Library *, struct MsgPortPPC *);

  FUNCTION
This function receives a message from a given message port. This
is the mirror function of exec/GetMsg.

  INPUTS
_PowerPCBase - base of powerpc.library
port - a pointer to a message port

  RESULTS
message - a pointer to the first available message or NULL if none
    is available.

  SEE ALSO

               WaitPortPPC
               ,
               PutMsgPPC
               ,
               ReplyMsgPPC
               , powerpc/portsPPC.i,
powerpc/portsPPC.h
```

1.46 getsystimeppc

```
               powerpc.library/GetSysTimePPC                    powerpc.library/ ↩
                   GetSysTimePPC

  NAME
GetSysTimePPC - get the current (relative) time (V7)

  CPU
PowerPC

  SYNOPSIS
GetSysTimePPC(_PowerPCBase, Dest )
        r3      r4
```

```
void GetSysTimePPC(struct Library *, struct timeval * );

  FUNCTION
Returns the current system time. This time is NOT absolute, there
is no relation between the real time and the time returned by
GetSysTimePPC. This function can be used for measurement of
time spans.

This is the mirror function of timer/GetSysTime.

  INPUTS
_PowerPCBase - base of powerpc.library
Dest - pointer to a timeval structure (where the time is stored)

  NOTES
This function has different behaviour on powerpc.library V7 and
powerpc.library V8+.

V7: The time is evaluated using the timer.device (via a cross
call) because it isn't possible to evaluate the busclock frequency
of the PPC with V7 (and ppc.library below) until now.

V8+: The time is evaluated completely native (and fast) using the
internal timers and the busclock frequency evaluated by WarpOS.

This function is safe to call from exception handlers ONLY in
powerpc.library V8 and higher !!

  SEE ALSO

              AddTimePPC
              ,
              SubTimePPC
              ,
              CmpTimePPC
```

1.47 gettagdatappc

```
              powerpc.library/GetTagDataPPC                    powerpc.library/ ↩
                 GetTagDataPPC

  NAME
GetTagDataPPC - obtain the data corresponding to a tag (V8)

  CPU
PowerPC

  SYNOPSIS
value = GetTagDataPPC(_PowerPCBase, tagValue, defaultVal, tagList)
r3                    r3            r4        r5          r6

ULONG *GetTagDataPPC(struct Library *, ULONG, ULONG, struct TagItem* );
```

```
  FUNCTION
Searches a tag list for a matching tag and returns the corresponding
ti_Data value for the TagItem found. If no match is found,
this function returns the value passed in as 'defaultVal'. This
is the mirror function of utility/GetTagData.

  INPUTS
_PowerPCBase - base of powerpc.library
tagValue - tag value to search for
defaultVal - value to be returned if tagValue is not found
tagList - tag item list to search (may be NULL)

  RESULT
value - the ti_Data value for the first matching TagItem, or
  'defaultVal' if a ti_Tag matching Tag is not found.

  NOTES
This function is safe to call from exception handlers.

  SEE ALSO

              FindTagItemPPC
              ,
              NextTagItemPPC
```

1.48 initsemaphoreppc

```
              powerpc.library/InitSemaphorePPC              powerpc.library/ ↩
                  InitSemaphorePPC

  NAME
InitSemaphorePPC - initializes a signal semaphore (V8)

  CPU
PowerPC

  SYNOPSIS
status = InitSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
r3                        r3            r4

LONG InitSemaphorePPC(struct Library *, struct SignalSemaphorePPC *);

  FUNCTION
Initializes a signal semaphore. This is the mirror function of
exec/InitSemaphore with some changes.

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure
         (all fields to zero)

  RESULT
status - status value:
    SSPPC_SUCCESS: function was successful
```

```
    SSPPC_NOMEM: function failed due to lack of memory

  NOTES
In opposite to exec/InitSemaphore a signal semaphore for PPC
has to be freed with FreeSemaphorePPC, because InitSemaphorePPC
allocates memory which should be freed after use.

  SEE ALSO

             FreeSemaphorePPC
             ,
             ObtainSemaphorePPC
             ,
             AttemptSemaphorePPC
             ,

             ReleaseSemaphorePPC
             ,
             AddSemaphorePPC
             ,
             RemSemaphorePPC
             ,

             FindSemaphorePPC
             , powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h
```

1.49 insertppc

```
             powerpc.library/InsertPPC                                  powerpc. ↩
                 library/InsertPPC

  NAME
InsertPPC - insert a node into a list (V8)

  CPU
PowerPC

  SYNOPSIS
Insert(_PowerPCBase, list, node, nodepredecessor)
       r3            r4    r5    r6

void InsertPPC(struct Library *, struct List*, struct Node*,
         struct Node*);

  FUNCTION
Insert a node into an standard exec list. This is the mirror
function of exec/Insert.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
list - a pointer to the target list
node - the node to insert
nodepredecessor - the node after which to insert. If 0 is passed,
      the node is inserted at the head of the list.
```

```
  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

This function is safe to call from exception handlers.

  SEE ALSO

              AddHeadPPC
              ,
              AddTailPPC
              ,
              RemovePPC
              ,
              RemHeadPPC
              ,
              RemTailPPC
              ,

              EnqueuePPC
              ,
              FindNamePPC
              , powerpc/listsPPC.i
```

1.50 locktasklist

```
              powerpc.library/LockTaskList                        powerpc.library/ ↩
                  LockTaskList

  NAME
LockTaskList - locks a list of all tasks (V10)

  CPU
PowerPC

  SYNOPSIS
TaskPtr = LockTaskList(_PowerPCBase)
r3                     r3

struct TaskPtr* LockTaskList(struct Library *);

  FUNCTION
This function locks a list of all PPC tasks currently available.
The main purpose of this function is to allow examining the
PPCtask structures without to worry about protecting the access
by semaphores (this is done internally).

The usual method of accessing the task information is the
following:

- Lock the task list using 'LockTaskList'
- Get the pointer to the first task (by reading out the entry
  TASKPTR_TASK of the returned TaskPtr structure)
```

```
- Scan trough the list and read out all infomation you need until
  you find the end of the list
- Unlock the list using 'UnLockTaskList'

  INPUTS
_PowerPCBase - base of powerpc.library

  RESULTS
TaskPtr - Ptr to the first node of a list of TaskPtr structures
    (see powerpc/tasksppc.i)

  NOTES
The WarpOS multitasking is NOT halted between LockTaskList and
UnLockTaskList.

No new tasks are created and no resources of removed tasks are
freed between LockTaskList and UnLockTaskList. So don't lock the
list for a too long time.

  SEE ALSO

             UnLockTaskList
             , powerpc/tasksppc.i, powerpc/tasksppc.h
```

1.51 modifyfpexc

```
             powerpc.library/ModifyFPExc                          powerpc.library ↩
                /ModifyFPExc

  NAME
ModifyFPExc - enables/disables specific floating point exceptions (V9)

  CPU
PowerPC

  SYNOPSIS
ModifyFPExc(_PowerPCBase, FPflags)
      r3            r4

void ModifyFPExc(struct Library *, ULONG);

  FUNCTION
This function allows to enable/disable particular floating
point exceptions. Multiple exceptions can be affected
simultaneously.

  INPUTS
_PowerPCBase - base of powerpc.library
FPflags - action to be performed:
    FPF_EN_OVERFLOW:    Enables FP overflow exception
    FPF_EN_UNDERFLOW:   Enables FP underflow exception
    FPF_EN_ZERODIVIDE:  Enables FP zero divide exception
    FPF_EN_INEXACT:     Enables FP inexact operation exception
    FPF_EN_INVALID:     Enables FP invalid operation exception
```

```
    FPF_DIS_OVERFLOW:   Disables FP overflow exception
    FPF_DIS_UNDERFLOW:  Disables FP underflow exception
    FPF_DIS_ZERODIVIDE: Disables FP zero divide exception
    FPF_DIS_INEXACT:    Disables FP inexact operation exception
    FPF_DIS_INVALID:    Disables FP invalid operation exception

  NOTES
Floating point exceptions must be enabled globally using 'SetHardware'
otherwise this function doesn't have any effect.

  SEE ALSO

              SetHardware
              , powerpc/powerpc.i, powerpc/powerpc.h
```

1.52 nexttagitemppc

```
              powerpc.library/NextTagItemPPC                  powerpc.library/ ↩
                 NextTagItemPPC

  NAME
NextTagItemPPC - iterate through a tag list (V8)

  CPU
PowerPC

  SYNOPSIS
tag = NextTagItemPPC(_PowerPCBase, tagItemPtr)
r3                   r3            r4

struct TagItem *NextTagItemPPC(struct Library *, struct TagItem **);

  FUNCTION
Iterates through a tag list, skipping and chaining as dictated by
system tags. Each call returns either the next item to be examined
or NULL when the end of the list has benn reached. This is the
mirror function of utility/NextTagItem.

  INPUTS
_PowerPCBase - base of powerpc.library
tagItemPtr - double-indirect reference to a TagItem structure.
       The pointer will be changed to keep track of the
       iteration.

  RESULT
tag - each TagItem in the array that should be processed is
      returned in turn with succesive calls.

  NOTES
This function is safe to call from exception handlers.

  SEE ALSO

              FindTagItemPPC
```

,
GetTagDataPPC

1.53 obtainsemaphoreppc

powerpc.library/ObtainSemaphorePPC powerpc.library/ ↩
ObtainSemaphorePPC

```
  NAME
ObtainSemaphorePPC - gain exclusive access to a semaphore (V8)

  CPU
PowerPC

  SYNOPSIS
ObtainSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
       r3            r4

void ObtainSemaphorePPC(struct Library *, struct SignalSemaphorePPC *);

  FUNCTION
Tries to get exclusive access to a signal semaphore. If the semaphore
is occupied, the task adds itself into a waiting queue. This is the
mirror function of exec/ObtainSemaphore.

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure

  NOTES
This call is guaranteed to preserve all GPR (except r0) and the CTR.

  SEE ALSO
```

InitSemaphorePPC
,
FreeSemaphorePPC
,
AttemptSemaphorePPC
,

ReleaseSemaphorePPC
,
AddSemaphorePPC
,
RemSemaphorePPC

FindSemaphorePPC
, powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h

1.54 putmsgppc

```
               powerpc.library/PutMsgPPC                                powerpc. ↩
                   library/PutMsgPPC

  NAME
PutMsgPPC - put a message to a message port (V11)

  CPU
PowerPC

  SYNOPSIS
PutMsgPPC(_PowerPCBase, MsgPortPPC, message)
    r3            r4          r5

void PutMsgPPC(struct Library *, struct MsgPortPPC *, struct Messge *);

  FUNCTION
This function attaces a message to the end of the given port. In
opposition to exec/PutMsg, only ports with PA_SIGNAL are supported.
This is the mirror function of exec/PutMsg.

  INPUTS
_PowerPCBase - base of powerpc.library
port - a pointer to a message port
message - a pointer to the message to be sent

  NOTES
This function is safe to call from exception handlers.

  SEE ALSO

              WaitPortPPC
              ,
              GetMsgPPC
              ,
              ReplyMsgPPC
              , powerpc/portsPPC.i,
powerpc/portsPPC.h
```

1.55 putxmsgppc

```
               powerpc.library/PutXMsgPPC                               powerpc. ↩
                   library/PutXMsgPPC

  NAME
PutXMsgPPC - sends an Inter-CPU message to a 68K task (V12)

  CPU
PowerPC

  SYNOPSIS
PutXMsgPPC(_PowerPCBase, MsgPort, message)
```

```
     r3            r4       r5

void PutXMsgPPC(struct Library *, struct MsgPort *, struct Message *);

  FUNCTION
This function sends an Inter-CPU message allocated by 'AllocXMsgPPC'
to a 68K task.

  INPUTS
_PowerPCBase - base of powerpc.library
MsgPort - an exec message port
message - a message allocated by 'AllocXMsgPPC'.

  NOTES
Inter-CPU must NOT be used for internal communication. They can
only be used for communication between tasks on different processors.

Inter-CPU messages get a different node type, if they are
sent. If you want to filter out Reply-Messages from standard
or Inter-CPU messages, compare the LN_TYPE field to NT_REPLYMSG.
Replied Inter-CPU messages still get the same node type
(NT_REPLYMSG). Any assumptions about the value of the new
node type are ILLEGAL!!

As soon as an Inter-CPU message is sent, the PPC looses ownership
over the message. No access to the message is allowed until the
reply has been arrived. If no replyport was specified, it's
allowed to free the message, after it was read from the other side.

Inter-CPU messages can be reused if they have been replied.

Inter-CPU messages are read and replied using the standard
message handling mechanisms (exec/WaitPort,exec/GetMsg,exec/ReplyMsg
for 68K, powerpc/WaitPortPPC, powerpc/GetMsgPPC, powerpc/ReplyMsgPPC
for PPC).

The receiving task must NOT access message data, which are not
explicitely located in the message body (for example data which is
referenced by a pointer) unless both tasks care for the cache
consistency. Only the message itself is flushed/invalidated
automatically by the system.

The receiving task may write to the message body of an Inter-CPU
message.

  SEE ALSO

              AllocXMsgPPC
              ,
              FreeXMsgPPC
```

1.56 releasesemaphoreppc

```
              powerpc.library/ReleaseSemaphorePPC        powerpc.library/ ↩
                 ReleaseSemaphorePPC

  NAME
ReleaseSemaphorePPC - make signal semaphore available to others (V8)

  CPU
PowerPC

  SYNOPSIS
ReleaseSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
        r3            r4

void ReleaseSemaphorePPC(struct Library *,
       struct SignalSemaphorePPC *);

  FUNCTION
Releases a semaphore locked by either ObtainSemaphorePPC or
AttemptSemaphorePPC. If other tasks are waiting, the foremost
in the waiting queue is waked up.

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure

  NOTES
This call is guaranteed to preserve all GPR (except r0) and the CTR.

If the semaphore is in an illegal state after calling
ReleaseSemaphorePPC, a system message will appear and the task is
put into waiting state.

  SEE ALSO

              InitSemaphorePPC
              ,
              FreeSemaphorePPC
              ,
              ObtainSemaphorePPC
              ,

              AttemptSemaphorePPC
              ,
              AddSemaphorePPC
              ,
              RemSemaphorePPC

              FindSemaphorePPC
              , powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h
```

1.57 remexchandler

```
              powerpc.library/RemExcHandler                  powerpc.library/ ↩
                 RemExcHandler
```

```
  NAME
RemExcHandler - removes a custom exception handler (V9)

  CPU
PowerPC

  SYNOPSIS
RemExcHandler(_PowerPCBase, XLock)
        r3            r4

void RemExcHandler(struct Library *, void *);

  FUNCTION
Removes an exception handler inserted by the function SetExcHandler.

  INPUTS
_PowerPCBase - base of powerpc.library
XLock - The lock value returned by SetExcHandler. It's safe to pass
  NULL as parameter

  SEE ALSO

             RemExcHandler
             , powerpc/powerpc.i, powerpc/powerpc.h
```

1.58 remheadppc

```
             powerpc.library/RemHeadPPC                              powerpc. ↩
                 library/RemHeadPPC

  NAME
RemHeadPPC - removes the head node from a list (V8)

  CPU
PowerPC

  SYNOPSIS
node = RemHeadPPC(_PowerPCBase, list)
r3                r3            r4

struct Node *RemHeadPPC(struct Library *, struct List*);

  FUNCTION
Removes the head node of a list. This is the mirror function of
exec/RemHead.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
list - the target list from which the head node should be removed

  RESULT
node - the node removed or 0 if the list was empty
```

```
  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

Assembler programmers may use the macro _REMHEAD located in
'powerpc/listsPPC.i'

This function is safe to call from exception handlers.

  SEE ALSO

              InsertPPC
              ,
              AddTailPPC
              ,
              AddHeadPPC
              ,
              RemovePPC
              ,
              RemTailPPC
              ,

              EnqueuePPC
              ,
              FindNamePPC
              , powerpc/listsPPC.i
```

1.59 removeppc

```
              powerpc.library/RemovePPC                              powerpc. ↩
                  library/RemovePPC

  NAME
RemovePPC - removes a node from a list (V8)

  CPU
PowerPC

  SYNOPSIS
RemovePPC(_PowerPCBase, node)
     r3           r4

void RemovePPC(struct Library *, struct Node*);

  FUNCTION
Removes a node from whatever list it is in. This is the mirror
function of exec/Remove.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
node - the node to remove

  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
```

```
is not passed in r3.

Assembler programmers may use the macro _REMOVE located in
'powerpc/listsPPC.i'

This function is safe to call from exception handlers.

  SEE ALSO

              InsertPPC
              ,
              AddTailPPC
              ,
              AddHeadPPC
              ,
              RemHeadPPC
              ,
              RemTailPPC
              ,

              EnqueuePPC
              ,
              FindNamePPC
              , powerpc/listsPPC.i
```

1.60 remportppc

```
              powerpc.library/RemPortPPC                              powerpc. ↩
                  library/RemPortPPC

  NAME
RemPortPPC - removes a public PPC message port from the system (V11)

  CPU
PowerPC

  SYNOPSIS
RemPortPPC(_PowerPCBase, MsgPortPPC)
     r3            r4

void RemPortPPC(struct Library *, struct MsgPortPPC *);

  FUNCTION
This function removes a public message port from the global list
of message ports. It is the mirror function of exec/RemPort.

  INPUTS
_PowerPCBase - base of powerpc.library
MsgPortPPC - pointer to a PPC message port. It's safe to pass a NULL
       parameter.

  SEE ALSO

              CreateMsgPortPPC
```

```
              ,
              DeleteMsgPortPPC
              ,
              FindPortPPC
              ,
              AddPortPPC
                powerpc/portsPPC.i, powerpc/portsPPC.h
```

1.61 remtailppc

```
              powerpc.library/RemTailPPC                              powerpc. ↩
                  library/RemTailPPC

  NAME
RemTailPPC - removes the tail node from a list (V8)

  CPU
PowerPC

  SYNOPSIS
node = RemTailPPC(_PowerPCBase, list)
r3                r3            r4

struct Node *RemTailPPC(struct Library *, struct List*);

  FUNCTION
Removes the tail node of a list. This is the mirror function of
exec/RemTail.

  INPUTS
_PowerPCBase - base of powerpc.library (can be omitted)
list - the target list from which the tail node should be removed

  RESULT
node - the node removed or 0 if the list was empty

  NOTES
This function is guaranteed to work correctly, if the PowerPCBase
is not passed in r3.

Assembler programmers may use the macro _REMTAIL located in
'powerpc/listsPPC.i'

This function is safe to call from exception handlers.

  SEE ALSO

              InsertPPC
              ,
              AddTailPPC
              ,
              AddHeadPPC
              ,
              RemovePPC
```

,
RemHeadPPC
,

EnqueuePPC
,
FindNamePPC
, powerpc/listsPPC.i

1.62 remsemaphoreppc

powerpc.library/RemSemaphorePPC powerpc.library/ ↩
RemSemaphorePPC

```
  NAME
RemSemaphorePPC - removes a global signal semaphore (V8)

  CPU
PowerPC

  SYNOPSIS
RemSemaphorePPC(_PowerPCBase, SignalSemaphorePPC)
    r3            r4

void RemSemaphorePPC(struct Library *, struct SignalSemaphorePPC *);

  FUNCTION
Removes a global signal semaphore created by AddSemaphorePPC. This
is the mirror function of exec/RemSemaphore.

  INPUTS
_PowerPCBase - base of powerpc.library
SignalSemaphorePPC - pointer to a signalsemaphorePPC structure

  SEE ALSO
```

InitSemaphorePPC
,
FreeSemaphorePPC
,
ObtainSemaphorePPC
,

AttemptSemaphorePPC
,
ReleaseSemaphorePPC
,
AddSemaphorePPC
,

FindSemaphorePPC
, powerpc/semaphoresPPC.i, powerpc/semaphoresPPC.h

1.63 replymsgppc

```
               powerpc.library/ReplyMsgPPC                          powerpc.library ↩
                   /ReplyMsgPPC

  NAME
ReplyMsgPPC - put a message to its reply port (V11)

  CPU
PowerPC

  SYNOPSIS
ReplyMsgPPC(_PowerPCBase, message)
      r3            r4

void ReplyMsgPPC(struct Library *, struct Messge *);

  FUNCTION
This function sends a message to its reply port, if one is present.
This is the mirror function of exec/ReplyMsg.

  INPUTS
_PowerPCBase - base of powerpc.library
message - a pointer to the message to be replied

  NOTES
This function is safe to call from exception handlers.

  SEE ALSO

               WaitPortPPC
               ,
               PutMsgPPC
               ,
               GetMsgPPC
               , powerpc/portsPPC.i,
powerpc/portsPPC.h
```

1.64 run68k

```
               powerpc.library/Run68K                                 powerpc. ↩
                   library/Run68K

  NAME
Run68K - runs a 680x0 function resp. AMIGA-OS library function (V7)

  CPU
PowerPC

  SYNOPSIS
status = Run68K(_PowerPCBase, PPStruct )
r3              r3           r4
```

```
LONG RunPPC (struct Library *, struct PowerPC *);

  FUNCTION
Runs a 680x0 function or an AMIGA-OS library function. All registers
can be transferred to 68K as well as parameters on stack. All cache
management actions are handled automatically.

All registers are transferred back from 68K after the 68K call
is completed. They are stored in the PowerPC structure.

See RunPPC for the register assignment.

  INPUTS
_PowerPCBase - base of powerpc.library
PPStruct - Pointer to an initialized PowerPC Structure

  PP_CODE       : Pointer to the 680x0 code resp. pointer to the
      library base (if PP_OFFSET is not zero).
  PP_OFFSET     : Library offset or 0 if no library function is
      called.
  PP_FLAGS      : Flags which can be ore'd together
  - PP[F/P]_ASYNC   : call 68K function asynchronely (PPC process
          returns immediately)
  PP_STACKPTR   : Pointer to the arguments on the stack to be
      transferred. The pointer must point to the top
      of the calling process' stackframe. The stack area
      to be transferred is located at offset 24 from this
      position. If no arguments on stack should be
      transferred, set this to zero.
  PP_STACKSIZE  : Size of the stack area to be transferred. If no
      stack parameters should be transferred, set this
      to zero.
  PP_REGS       : Array of longwords where the registers to be
      transferred can be stored (d0-a6)
  PP_FREGS      : Array of quadwords (8 bytes) where the FP-registers
      to be transferred can be stored (fp0-fp7)

  RESULT
status - PPERR_SUCCESS if the call was successfully
   PPERR_ASYNCERR if a synchrone 68K call was made after an
      asynchrone 68K call

  NOTES
Calling a 68K function asynchronely is dangerous. Take care of
possible cache conflicts. Avoid calling system functions as much
as possible.

If an asynchrone 68K call is done, the function WaitFor68K MUST be
called ONCE after the call was done. No other 68K call is allowed
for this PPC process after an asynchrone 68K call and before a call
to WaitFor68K.

If an asynchronely called 68K function performs a PPC call, the call
is only performed when WaitFor68K is called by the PPC process. Note
that the 68K mirror process is still connected to the calling PPC
process.
```

```
  DON'T pass arguments on stack when calling a 68K function asynchronely.
  The stack is most likely to be trashed before it is copied to the 68K
  stack.

  Assembler programmers should use the macros RUN68K and
  RUN68K_XL located in the include file 'powerpc/powerpc.i'

    SEE ALSO

                WaitFor68K
                , powerpc/powerpc.i, powerpc/powerpc.h
```

1.65 setcache

```
powerpc.library/SetCache                                powerpc.library/SetCache

    NAME
  SetCache - cache manipulation function (V8)

    CPU
  PowerPC

    SYNOPSIS
  SetCache(_PowerPCBase, cacheflags, start, length)
     r3            r4          r5     r6

  void SetCache(struct Library *, ULONG, void *, ULONG);

    FUNCTION
  This function offers many possibilities to affect the caches of
  the PPC. It performs the action defined by the cache flags. Only
  one action can be performed at the same time.

    INPUTS
  _PowerPCBase - base of powerpc.library
  cacheflags - action to be performed:
      CACHE_DCACHEOFF:    Data cache is disabled. The Cache is flushed
        automatically.
      CACHE_DCACHEON:     Data cache is enabled.
      CACHE_DCACHELOCK:   Data cache is locked (is ignored if either
        'start' or 'length' is zero).
      CACHE_DCACHEUNLOCK: Data cache is unlocked.
      CACHE_DCACHEFLUSH:  Data cache is flushed.
      CACHE_ICACHEOFF:    Instruction cache is disabled.
      CACHE_ICACHEON:     Instruction cache is enabled.
      CACHE_ICACHELOCK:   Instruction cache is locked.
      CACHE_ICACHEUNLOCK: Instruction cache is unlocked.
      CACHE_ICACHEINV:    Instruction cache is invalidated.

  start - pointer to the start address of the area to be affected.
    The following cacheflags support an area specification:
    CACHE_DCACHELOCK, CACHE_DCACHEFLUSH, CACHE_ICACHEINV

    if 'start' is 0 the whole address space is affected
```

```
length - length of the area to be affected (see above for the
   cache flags which support area specification).
   if 'length' is 0 the whole address space is affected

  NOTES
Invalidating the whole instruction cache is much more efficient
than flushing only a part of it.

Flushing the whole data cache is less efficient than flushing
a specific area, if this area is not too large.

The cacheflag DCACHELOCK requires 'start' and 'length' to be not
zero. The area specified is then copied into the data cache and
the data cache is locked afterwards.

The caches should not be switched on/off resp. locked/unlocked
without GOOD justification. Global manipulations of the cache
should be avoided. Better affect the cache locally by using
AllocVecPPC.

This function is safe to call from exception handlers

  SEE ALSO
powerpc/powerpc.i, powerpc/powerpc.h
```

1.66 setexchandler

```
              powerpc.library/SetExcHandler                    powerpc.library/ ↩
                  SetExcHandler

  NAME
SetExcHandler - insert a custom exception handler (V9)

  CPU
PowerPC

  SYNOPSIS
XLock = SetExcHandler(_PowerPCBase, ExcTags)
r3                    r3            r4

void *SetExcHandler(struct Library *, struct TagItem *);

  FUNCTION
This function allows applications to insert custom exception
handlers. Those handlers can be global or task dependant,
priorities are also supported. Multiple exceptions can be
selected for one exception handler.

The exception handlers are executed in supervisor mode and
have access to all supervisor registers.

The handlers are called in one of two possible ways (dependant
of the tag EXC_FLAGS):
```

```
1) EXC_FLAGS has the flag EXCF_SMALLCONTEXT set:


   Status = CustomHandler(SmallContext)
   r3                     r3

   ULONG CustomHandler(struct XCONTEXT*)

   Inputs:
  SmallContext - a pointer to a XCONTEXT structure (see
           powerpc/powerpc.i)

   Result:
  Status - a return value which decides, whether the
     exception should be leaved immediately or
     if following exception handlers should be
     executed, too (see powerpc/powerpc.i)


  In this first method, the handler gets all registers directly,
  except for r3, which is passed in the XCONTEXT structure. The
  exception ID, which gives information about the kind of
  exception occurred, is passed also in XCONTEXT structure.

  All registers which are modified by the exception handler are
  also modified for the interrupted task. If r3 should be modified
  for the interrupted task, the appropriate field of the XCONTEXT
  structure has to be modified.

  Some of the interrupted task's registers are passed in special
  supervisor registers. If they should be changed, the appropriate
  supervisor registers have to be changed:

  SPRG1 -       The interrupted task's Link Register
  SPRG2 -       The interrupted task's Stackpointer (r1)
  SPRG3 -       The interrupted task's Smalldata Base (r2)

  The exception stack, which is passed in r1, is allocated from the
  user stack of the interrupted task.


2) EXC_FLAGS has the flag EXCF_LARGECONTEXT set:


   Status = CustomHandler(LargeContext)
   r3                     r3

   ULONG CustomHandler(struct EXCCONTEXT*)

   Inputs:
  LargeContext - a pointer to a EXCCONTEXT structure (see
           powerpc/powerpc.i)

   Result:
  Status - a return value which decides, whether the
     exception should be leaved immediately or
     if following exception handlers should be
```

```
      executed, too (see powerpc/powerpc.i)

   In this second method, the handler gets all registers in the
   EXCCONTEXT structure. If the handler wishes to change some
   of the register contents it must change the appropriate fields
   in the EXCCONTEXT structure which are copied back to the
   registers after the custom handler has completed. If no
   field is provided for a certain register, it has to be modified
   directly.

   The exception stack, which is passed in r1, is allocated from the
   user stack of the interrupted task.

   INPUTS
_PowerPCBase - base of powerpc.library
ExcTags - pointer to a tagitem array. The following tags are
     supported:

     EXCATTR_CODE:       pointer to the exception handler code
       (required)
     EXCATTR_DATA:       a user data which is passed in r2 to the
       custom exception handler. This is usually
       a base register which provides access to
       all global data required.
     EXCATTR_TASK:       specifies the task which is allowed to take
       the exception handler. If this tag is 0 or
       omitted, the current task is taken instead.
       This tag has no effect if the exception
       handler is specified as global (see below
       at EXCATTR_FLAGS)
     EXCATTR_EXCID:      Defines which exceptions should call this
       exception handler. See powerpc/powerpc.i
       for a description of all supported exceptions.
       Multiple exceptions can be selected.
     EXCATTR_FLAGS:
   EXCF_GLOBAL:       Marks the exception handler as global. It
          is then called for every exception occurred.
   EXCF_LOCAL:        Marks the exception handler as local. It is
          then only called, if the interrupted task
          matches the task specified in EXCATTR_TASK.
   EXCF_SMALLCONTEXT: The exception handler is called with a
          XCONTEXT structure as parameter (see above
          for a description of this mode).
   EXCF_LARGECONTEXT: The exception handler is called with a
          EXCCONTEXT structure as parameter (see above
          for a description of this mode).

       One flag of EXCF_GLOBAL and EXCF_LOCAL and one
       flag of EXCF_SMALLCONTEXT and EXCF_LARGECONTEXT
       must be specified, otherwise this function
       fails.

     EXCATTR_NAME:       An identification name for this handler
     EXCATTR_PRI:        The priority of this exception handler

   RESULT
XLock - A lock to be passed to RemExcHandler or 0 if something
```

failed.

NOTES

Exception handlers should generally take care that they don't destroy any registers of the interrupted task except if it is desired. Special care must be taken if the exception handler is called with the small context structure (take care of r0!). All registers, inclusive CR, CTR, LR and others must be restored if they should not be changed.

IMPORTANT: The exception handler is called with MMU switched off! The whole address space is accessed in cachable copyback mode, so no access to critical locations such as custom chip space must be performed. If such locations should be accessed, a temporary MMU setup must be done using the library functions 'SetExcMMU' and 'ClearExcMMU' (V10)

Note that changes to the MSR of the interrupted task must be done by writing to SRR1 (i.e. setting the trace bit).

Note that exception handlers should generally not call system functions with some exceptions (for example 'SignalPPC' is often useful to call from exception handlers). System functions must not be called unless it's allowed explicitely by the documentation of each function.

Note that the content of the program counter (SRR0) differs depending on the exception type. Sometimes it contains the address of the excepting instruction and sometimes the address of the next instruction to complete. Exception handlers must take care about this and should set the program counter appropriately.
Here follows a table of all supported exceptions and their behaviour:

```
Machine check (EXCF_MCHECK):         PC -> maybe next instruction
Data access (EXCF_DACCESS):          PC -> excepting instruction
Instruction access (EXCF_IACCESS):   PC -> next instruction
Alignment (EXCF_ALIGN):              PC -> excepting instruction
Program (EXCF_PROGRAM):              PC -> excepting instruction
FP unavailable (EXCF_FPUN):          PC -> excepting instruction
Trace (EXCF_TRACE):                  PC -> next instruction
Performance Monitor (EXCF_PERFMON): unknown
Instruction breakpoint (EXCF_IABR): PC -> excepting instruction
```

Exception handlers should not waste stack space. The system allocates an extra space for this purpose but it may not be sufficient if very stack-intensive routines are called as exception handlers.

If every exception handler returns the state EXCRETURN_NORMAL then the standard WarpOS exception handler is executed (except for Trace- and PerformanceMonitor-Exceptions).

If exception handlers are written to emulate commands causing an exception they should return EXCRETURN_ABORT as return value to suppress following exception handlers which might output some alert messages. The priority should be probably high enough to ensure that no unwanted reactions occur.

```
The WarpOS debugging system is disabled during exception
processing.

It's completely ILLEGAL to exit an exception handler by an RFI
instruction!! It's also illegal to trash SPRG0!

  SEE ALSO

              RemExcHandler
              , powerpc/powerpc.i, powerpc/powerpc.h
```

1.67 setexcmmu

```
              powerpc.library/SetExcMMU                                powerpc. ↩
                 library/SetExcMMU

  NAME
SetExcMMU - installs a BAT based MMU setup for exception handlers (V10)

  CPU
PowerPC

  SYNOPSIS
SetExcMMU(_PowerPCBase)
    r3

void SetExcMMU(struct Library *);

  FUNCTION
This function is for exception handlers only. It installs a new
temporary MMU setup which allows exception handlers to access critical
address space, such as custom chip space. Exception handlers are
normally run with MMU switched off to avoid problems on systems
without hardware tablesearch.

The new MMU setup is based on the BAT registers.

The function 'ClearExcMMU' restores the old MMU state and should
be called at the end of the exception handler.


  INPUTS
_PowerPCBase - base of powerpc.library

  NOTES
This function must not be called from anywhere else than from
an exception handler.

  SEE ALSO

              ClearExcMMU
```

1.68 sethardware

```
              powerpc.library/SetHardware                          powerpc.library ↩
                  /SetHardware

  NAME
SetHardware - hardware manipulation function (V9)

  CPU
PowerPC

  SYNOPSIS
Status = SetHardware(_PowerPCBase, hardwareflags, parameter)
         r3            r4             r5

ULONG SetHardware(struct Library *, ULONG, void *);

  FUNCTION
This function offers some functions to access the PPC hardware.

  INPUTS
_PowerPCBase - base of powerpc.library
hardwareflags - action to be performed:
    HW_TRACEON:         Enables trace mode for the current task
    HW_TRACEOFF:        Disables trace mode for the current task
    HW_BRANCHTRACEON:   Enables branch trace mode for the current task
    HW_BRANCHTRACEOFF:  Disables branch trace mode for the current task
    HW_FPEXCON:         Enables the floating point exceptions for the
      current task
    HW_FPEXCOFF:        Disables the floating point exceptions for the
      current task
    HW_SETIBREAK:       Sets the global instruction breakpoint
    HW_CLEARIBREAK:     Clears the global instruction breakpoint
    HW_SETDBREAK:       Sets the global data breakpoint
    HW_CLEARDBREAK:     Clears the global data breakpoint

parameter - additional parameter only used if a breakpoint should be
      set. Then it contains the breakpoint address.

  RESULT
Status - HW_AVAILABLE: The requested feature is available on this CPU
   HW_NOTAVAILABLE: The requested feature is not available on
        this CPU

  NOTES
Floating point exceptions are only enabled globally with HW_FPEXCON.
It's necessary to call 'ModifyFPExc' to enable the desired floating
point exceptions.

Floating point exceptions should not be enabled by standard
applications. They are intended to use for debugging purposes.

The data breakpoint feature is not available on PPC603[E].

  SEE ALSO
```

```
               ModifyFPExc
               , powerpc/powerpc.i, powerpc/powerpc.h
```

1.69 setnicevalue

```
               powerpc.library/SetNiceValue                    powerpc.library/ ↩
                  SetNiceValue

  NAME
SetNiceValue - sets the NICE value of a task (V14)

  CPU
PowerPC

  SYNOPSIS
OldNice = SetNiceValue(_PowerPCBase, TaskPPC, Nice)
r3                     r3            r4       r5

LONG* SetNiceValue(struct Library *, struct TaskPPC *, LONG);

  FUNCTION
This function can be used to set the NICE value of a task. The
NICE value is a kind of priority, which replaces the old priority
in LN_PRI. NICE values were introduced with the dynamic scheduler
which works very differently than the old scheduler.

If a task gets a high NICE value, it means that the task is nice
to other tasks and won't request much CPU time. If a task gets
a low NICE value, the opposite occurs. Using NICE values it is
possible to affect execution speed of tasks.

  INPUTS
_PowerPCBase - base of powerpc.library
TaskPPC - pointer to the task which should be affected
Nice - the NICE value (-20 ... 20)

  RESULT
The old NICE value

  SEE ALSO

               SetTaskPriPPC
               , powerpc/tasksPPC.i, powerpc/tasksPPC.h
```

1.70 setreplyportppc

```
               powerpc.library/SetReplyPortPPC                 powerpc.library/ ↩
                  SetReplyPortPPC

  NAME
```

```
  SetReplyPortPPC - exchanges the reply port of a message (V13)

    CPU
  PowerPC

    SYNOPSIS
  OldPort = SetReplyPortPPC(_PowerPCBase, Message, MsgPortPPC)
  r3                        r3            r4       r5

  struct MsgPortPPC* SetReplyPortPPC(struct Library *,
         struct Message *, struct MsgPortPPC *);

    FUNCTION
  This function exchanges the message port of a message. It
  can be used for internal PPC messages and for InterCPU
  messages.

    INPUTS
  _PowerPCBase - base of powerpc.library
  Message - a pointer to the message affected
  MsgPortPPC - a pointer to a PPC message port

    RESULT
  The old reply port, which was installed in the message.
  Can be NULL, of course.

    NOTES
  It's possible to specify NULL as MsgPort to remove the
  reply port.

  Don't poke into the message structure at MN_REPLYPORT,
  please use this function here.

    SEE ALSO

                WaitPortPPC
                ,
                GetMsgPPC
                ,
                ReplyMsgPPC
                ,
                PutMsgPPC
                , powerpc/portsPPC.i,
  powerpc/portsPPC.h
```

1.71 setscheduling

```
powerpc.library/SetScheduling                    powerpc.library/SetScheduling

    NAME
  SetScheduling - affects scheduling behaviour (V14)

    CPU
  PowerPC
```

```
  SYNOPSIS
SetScheduling(_PowerPCBase, SchedTagList)
        r3            r4

void SetScheduling(struct Library *, struct TagItem *);

  FUNCTION
This function is used to set some scheduling parameters.

  INPUTS
_PowerPCBase - base of powerpc.library
SchedTagList - pointer to a tagitem array. The following tags are
         supported:

    SCHED_REACTION:     Sets the reaction time of low-activity tasks.
      This value can be in range of (1..20). The
      higher the value the more CPU time a
      low-activity task gets (but which causes
      longer blocking of busy tasks). Default is
      currently 6.

  SEE ALSO
powerpc/powerpc.i, powerpc/powerpc.h
```

1.72 setsignalppc

```
              powerpc.library/SetSignalPPC                      powerpc.library/ ↩
                 SetSignalPPC

  NAME
SetSignalPPC - define the state of this task's signals (V8)

  CPU
PowerPC

  SYNOPSIS
oldSignals = SetSignalPPC(_PowerPCBase, newSignals, signalMask)
r3                        r3            r4          r5

ULONG SetSignalPPC(struct Library *, ULONG, ULONG);

  FUNCTION
This function can query or modify the state of the current
signals. This function is the mirror function of exec/SetSignal.

  INPUTS
_PowerPCBase - base of powerpc.library
newSignals - the new values for the signals
signalMask - the set of signals to be affected.

  RESULT
oldSignals - the prior values for all signals

  NOTES
```

```
It’s possible to check for the system signals (i.e. CTRL_C).

Before V11, this only worked if the task was stated using
RunPPC, this is not the case anymore.

  SEE ALSO

              AllocSignalPPC
              ,
              FreeSignalPPC
              ,
              SignalPPC
              ,
              WaitPPC
```

1.73 settaskprippc

```
              powerpc.library/SetTaskPriPPC                    powerpc.library/ ↩
                  SetTaskPriPPC

  NAME
SetTaskPriPPC - get and set the priority of a task (V8)

  CPU
PowerPC

  SYNOPSIS
oldpriority = SetTaskPriPPC(_PowerPCBase, taskPPC, priority)
r3                          r3            r4       r5

LONG SetTaskPriPPC(struct Library *, struct TaskPPC *, LONG);

  FUNCTION
This function changes the priority of a task regardless of its
state. The old priority of the task is returned. A reschedule
may be performed. This is the mirror function of exec/SetTaskPri.

Important: These task priorities are completely useless from V14
on because of the new dynamic scheduler which uses NICE values
instead of fixed priorities. Use ’SetNiceValue’ to give tasks
more or less CPU time.

  INPUTS
_PowerPCBase - base of powerpc.library
taskPPC - task to be affected
priority - the new priority for the task

  RESULT
old priority - the tasks previous priority

  SEE ALSO

              SetNiceValue
```

1.74 signal68k

```
powerpc.library/Signal68K                              powerpc.library/Signal68K

    NAME
  SignalPPC - signal a 68K task (V8)

    CPU
  PowerPC

    SYNOPSIS
  Signal68K(_PowerPCBase, task, signals)
      r3            r4       r5

  void Signal68K(struct Library *, struct Task*, ULONG);

    FUNCTION
  This function signals a 68K task with the given signals. If
  the signalled task is sleeping, it's woken up and a reschedule
  may occur.

    INPUTS
  _PowerPCBase - base of powerpc.library
  task - the 68K task to be signalled
  signals - the signals to be sent
```

1.75 signalppc

```
               powerpc.library/SignalPPC                              powerpc. ↩
                   library/SignalPPC

    NAME
  SignalPPC - signal a task (V8)

    CPU
  PowerPC

    SYNOPSIS
  SignalPPC(_PowerPCBase, taskPPC, signals)
      r3            r4       r5

  void SignalPPC(struct Library *, struct TaskPPC*, ULONG);

    FUNCTION
  This function signals a task with the given signals. If the
  signalled task is sleeping, it's woken up and a reschedule
  may occur. This is the mirror function of exec/Signal.

    INPUTS
  _PowerPCBase - base of powerpc.library
```

```
taskPPC - the task to be signalled
signals - the signals to be sent

  NOTES
This function is safe to call from exception handlers.

Since V11 it's possible to signal a 68K task directly
with 'SignalPPC' and the 68K task structure in r4.

Before V11, 'SignalPPC' only worked, if the PPC task
had a mirror 68K task. Since V11, every PPC task has
a mirror task, so 'SignalPPC' can always be used.

Furthermore, all signals, which are sent to a task currently
waiting for its mirror task to complete, are transferred
automatically to the mirror task. In fact, the whole signal
system is shared and can be viewed as one single 'virtual
signaling system'. It really doesn't matter, on which CPU
a program is running, the signals are always transferred to
the currently active part.

  SEE ALSO

             AllocSignalPPC
             ,
             FreeSignalPPC
             ,
             SetSignalPPC
             ,
             WaitPPC
```

1.76 snooptask

```
             powerpc.library/SnoopTask                                  powerpc. ↩
                library/SnoopTask

  NAME
SnoopTask - monitors beginning or end of a PPC task (V13)

  CPU
PowerPC

  SYNOPSIS
SnoopID = SnoopTask (_PowerPCBase, SnoopTags)
r3                  r3             r4

ULONG SnoopTask (struct Library *, struct TagItem *);

  FUNCTION
This function allows to intall a callback job, which is executed
when a new PPC task is started or when a PPC task is removed.
This is useful for debuggers which want to be kept informed
about new tasks installed into the system and about tasks removed
from the system.
```

```
The callback function has two different formats (prototypes),
dependant on the type of callback:

1. callback function for monitoring the beginning of a PPC task
   (SNOOP_TYPE = TASK_START)

   SYNOPSIS:

   CallbackFunction (PPCTask, EntryCode, CreatorTask, CreatorCPU)
         r3       r4         r5           r6

   void CallbackFunction (struct TaskPPC *, APTR, struct Task *, ULONG);

   INPUTS:
   PPCTask - pointer to the new PPC task which is created
   EntryCode - pointer to the start code, which will be executed
         by the new task
   CreatorTask - pointer to the task structure of the task, which
     created the new PPC task. If the new task is created
     due to a call of RunPPC from 68K side, then the
     CreatorTask points to the 68K-Task-Structure. If
     the PPC task was created by directly calling
     'CreateTaskPPC', CreatorTask points to this PPC task.
   CreatorCPU - One of two possible values:
    CREATOR_PPC : The new PPC task was created using
            'CreateTaskPPC' by a PPC task.
    CREATOR_68K : The new PPC task was created using
            'RunPPC' by a 68K task.


2. callback function for monitoring the end of a PPC task
   (SNOOP_TYPE = TASK_EXIT)

   SYNOPSIS:

   CallbackFunction (PPCTask)
         r3

   void CallbackFunction (struct TaskPPC *);

   INPUTS:
   PPCTask - pointer to the PPC task which is removed. NEVER use
       FindTaskPPC(NULL), because it's possible that a PPC
       task is removed by another PPC task!

  INPUTS
_PowerPCBase - base of powerpc.library
TagItems - pointer to a tagitem array. The following tags are
     supported:

    SNOOP_CODE:         pointer to the callback function which should
      be called, if a new PPC task is created or
      if a PPC task is removed.
    SNOOP_DATA:         custom data which passed in register r2.
      this will usually be the smalldata base of
      the task which wants to snoop other tasks
```

```
       to gain access to more data space.
     SNOOP_TYPE:         two possible values:
       SNOOP_START : The callback function is called
               when a new PPC task is created
       SNOOP_EXIT  : The callback function is called
               when a PPC task is removed
   RESULT
 SnoopID - An ID, which must be passed to EndSnoopTask, as soon as the
     snooping action should be terminated. NULL, if an error
     occurs.

   SEE ALSO

               EndSnoopTask
```

1.77 sprintf

```
powerpc.library/SPrintF                                powerpc.library/SPrintF

   NAME
 SPrintF - prints a formatted string to the serial port (V7)

   CPU
 PowerPC

   SYNOPSIS
 SPrintF (_PowerPCBase, Formatstring, values )
    r3             r4             r5

 void SPrintF (struct Library *, STRPTR, APTR);

   FUNCTION
 Prints a formatted string to the serial port using the AMIGA-OS
 functions 'exec/RawPutChar' and 'exec/RawDoFmt'. Can be used to add
 debugging features and to improve the maintenance of software.

   INPUTS
 _PowerPCBase - base of powerpc.library
 Formatstring - A C style string with % commands to indicate where
          parameters have to be inserted (see 'exec/RawDoFmt'
          for a detailed description of these commands).
 values - A pointer to an array of parameters to be inserted into
    specified places in the string.

   SEE ALSO
 exec/RawDoFmt
```

1.78 subtimeppc

```
               powerpc.library/SubTimePPC                          powerpc. ↩
                  library/SubTimePPC
```

```
  NAME
SubTimePPC - subtracts one time request from another (V7)

  CPU
PowerPC

  SYNOPSIS
SubTimePPC(_PowerPCBase, Dest, Source)
     r3            r4    r5

void SubTimePPC(struct Library *, struct timeval *, struct timeval *);

  FUNCTION
This routine subtracts one timeval structure from another. The results
are stored in the destination (Dest - Source -> Dest)

This is the mirror function of timer/SubTime.

  INPUTS
_PowerPCBase - base of powerpc.library
Dest - pointer to a timeval structure
Source - pointer to a timeval structure

  NOTES
This function is safe to call from exception handlers

  SEE ALSO

              GetSysTimePPC
              ,
              AddTimePPC
              ,
              CmpTimePPC
```

1.79 super

```
              powerpc.library/Super                                    powerpc. ↩
                  library/Super

  NAME
Super - switch to supervisor mode (V9)

  CPU
PowerPC

  SYNOPSIS
SuperKey = Super(_PowerPCBase)
r3               r3

ULONG Super(struct Library *);

  FUNCTION
This function changes the current task to supervisor mode.
```

```
  INPUTS
_PowerPCBase - base of powerpc.library

  RESULT
SuperKey - A key value which must be passed to 'User' to switch back
     to user mode

  NOTES
Applications should generally not enter supervisor mode. Check first
if there exists a library function which gives you access to the
supervisor resources required.

  SEE ALSO

             User
```

1.80 unlocktasklist

```
             powerpc.library/UnLockTaskList                powerpc.library/ ↩
                 UnLockTaskList

  NAME
UnLockTaskList - unlocks a list locked by LockTaskList (V10)

  CPU
PowerPC

  SYNOPSIS
UnLockTaskList(_PowerPCBase)
         r3

void UnLockTaskList(struct Library *);

  FUNCTION
Unlocks the task list which was locked by LockTaskList

  INPUTS
_PowerPCBase - base of powerpc.library

  SEE ALSO

             LockTaskList
             , powerpc/tasksppc.i, powerpc/tasksppc.h
```

1.81 user

```
             powerpc.library/User                                    powerpc. ↩
                 library/User
```

```
  NAME
User - switch to user mode (V9)

  CPU
PowerPC

  SYNOPSIS
User(_PowerPCBase, SuperKey)
     r3            r4

void User(struct Library *, ULONG);

  FUNCTION
This function changes the current task to user mode.

  INPUTS
_PowerPCBase - base of powerpc.library
SuperKey - The return value of the matching call of 'Super'

  SEE ALSO

              Super
```

1.82 waitfor68k

```
              powerpc.library/WaitFor68K                              powerpc. ↩
                  library/WaitFor68K

  NAME
WaitFor68K - waits for the completion of an asynchrone 68K call (V7)

  CPU
PowerPC

  SYNOPSIS
status = WaitFor68K(_PowerPCBase, PPStruct )
r3                  r3             r4

LONG WaitFor68K (struct Library *, struct PowerPC *);

  FUNCTION
After an asynchrone 68K call was done (see Run68K) this function must
be called to wait for the completion of the 68K function. All
registers transferred to the PowerPC with Run68K are returned into the
PowerPC Structure.

  INPUTS
_PowerPCBase - base of powerpc.library
PPStruct - Pointer to a PowerPC Structure (see Run68K for a description
     of the elements). The structure has not to be initialized.
     The structure must be transferred to hold the returned
     registers by the 68K function.

  RESULT
```

```
status - PPERR_SUCCESS if the call was successfully
   PPERR_WAITERR if WaitFor68K is called after a synchrone 68K
           call.

  NOTES
Assembler programmers should use the macros WAITFOR68K and
WAITFOR68K_XL located in the include file 'powerpc/powerpc.i'

  SEE ALSO

              Run68K
              , powerpc/powerpc.i, powerpc/powerpc.h
```

1.83 waitportppc

```
              powerpc.library/WaitPortPPC                          powerpc.library ↩
                 /WaitPortPPC

  NAME
WaitPortPPC - wait for a given port to be non-empty (V11)

  CPU
PowerPC

  SYNOPSIS
message = WaitPortPPC(_PowerPCBase, MsgPortPPC)
r3                    r3            r4

struct Message *WaitPortPPC(struct Library *, struct MsgPortPPC *);

  FUNCTION
This function waits until the given port becomes non-empty. The
first message in the port is returned without removing it from
the port. This is the mirror function of exec/WaitPort.

  INPUTS
_PowerPCBase - base of powerpc.library
port - a pointer to a message port

  RESULTS
message - a pointer to the first available message

  SEE ALSO

              PutMsgPPC
              ,
              GetMsgPPC
              ,
              ReplyMsgPPC
              , powerpc/portsPPC.i,
powerpc/portsPPC.h
```

1.84 waitppc

```
               powerpc.library/WaitPPC                                  powerpc. ↩
                  library/WaitPPC

  NAME
WaitPPC - wait for one or more signals (V8)

  CPU
PowerPC

  SYNOPSIS
signals = WaitPPC(_PowerPCBase, signalSet)
r3                r3            r4

ULONG WaitPPC(struct Library *, ULONG);

  FUNCTION
This function attempts to wait for the given signals. If at
least one of these signal is already set, the task returns
immediately, otherwise it changes to waiting state. This is
the mirror function of exec/Wait.

  INPUTS
_PowerPCBase - base of powerpc.library
signalSet - the set of signals for which to wait

  RESULTS
signals - the signals which were received

  NOTES
Since V11 it's possible to wait for signals which might
be sent by 68K tasks (and maybe only to the mirror 68K
task of this PPC task here). Calling exec/Signal with a
PPC task structure as first argument will work, too.
See the desription of 'SignalPPC' for more information
about the shared signaling system.

  SEE ALSO

               AllocSignalPPC
               ,
               FreeSignalPPC
               ,
               SetSignalPPC
               ,
               SignalPPC
```

1.85 waittime

```
               powerpc.library/WaitTime                                 powerpc. ↩
                  library/WaitTime
```

```
  NAME
WaitTime - wait for a given time or for given signals (V10)

  CPU
PowerPC

  SYNOPSIS
signals = WaitTime(_PowerPCBase, signalSet, time)
r3                 r3            r4         r5

ULONG WaitTime(struct Library *, ULONG, ULONG);

  FUNCTION
This function attempts to wait for a specific time of for
given signals. This function acts just like 'WaitPPC' with
the difference that the task returns when the time specified
is over.

  INPUTS
_PowerPCBase - base of powerpc.library
signalSet - the set of signals for which to wait (can be 0
      if the task should only wait for a given time)
time - the time in microseconds to wait

  RESULTS
signals - the signals which were received (if this value is 0,
    then the time is up).

  NOTES
The time which explicitly passes between calling 'WaitTime' and
returning from it can vary dependant of current system state. If
many tasks are active, the time can be delayed. Furthermore the
overhead of this function must be taken in account.

  SEE ALSO

              WaitPPC
```