3.8 MMU Management

The executive can optionally support the PMMU (M68851 and M68030) to provide memory protection, dynamic task loading, and dynamic memory allocation.

To provide these services, the executive adopts an MMU model which defines the *pagesize*, the structure and depth of the memory map tree, and the degree of control each task has over its own memory map. Different implementations of the RTEID are free to choose different models. However, the model chosen should allow the standard memory management services (regions and partitions) to operate in a consistent and intuitive manner in both an MMU and non-MMU environment.

Logically, the RTEID adopts a sectioned view of the logical address space associated with each task. Memory objects are mapped into a task's logical address space in variable size MMU sections. A single section is contiguous in the logical and possibly the underlying physical address spaces. Thus, the MMU is used to define a set of mappings for each task in the form:

(logical address, length) --> physical address range

Based on this model, the RTEID defines how the memory management services should operate, and defines additional services to manage the MMU directly.

3.8.1 Segments vs. Sections

MMU sections should not be confused with region segments. A segment is a block of memory allocated from a region. It can exist on any CPU in the M68000 family. A section is only meaningful on the M68030 or M68020/M68851 combination, and refers to a contiguous block of memory which is mapped into a task's address space.

3.8.2 Regions

When a task calls *rn_getseg* to obtain a segment from a region, the segment is automatically mapped into the task's logical address space at an executive assigned address. Because *rn_getseg* performs the mapping, the corresponding region is not mapped into the address space of tasks using it. This means that allocated sections are accessible only by the allocating task, and those tasks which explicitly are given access to the segment using the MMU directives. Thus, a segment is fully protected from inadvertent access by other tasks.

3.8.3 Partitions

When a task executes a *pt_create* or *pt_ident* directive, the entire partition is mapped into the task's address space. Thus, tasks which share a partition can share and access any buffers allocated from the partition. However, protection is on the partition level, and individual buffers are not protected.

The directives provided by the MMU manager are:

Directive	Function
mm_l2p	Logical to physical
mm_p2l	Physical to logical
mm_pmap	Map physical
mm_unmap	Unmap logical
mm_pread	Physical read
mm_pwrite	Physical write
mm_ptcreate	Create logical partition

3.8.4 MM_L2P

NAME

mm_l2p -- "Logical to Physical"

SYNOPSIS

```
#include <memory.h>
uint mm_l2p ( tid, laddr, &paddr, &length )

        uint tid;       /* task id as returned by t_create or t_ident */
        char *laddr;    /* logical start address */
        char *paddr;    /* physical start address - returned by this call */ */
        uint length;    /* remaining length in bytes - returned by this call */
```

DESCRIPTION

This directive calculates the physical address within the section associated with the logical address belonging to the task identified by the *tid*.

The physical start address is returned in the *paddr* field. The number of bytes remaining in the section is returned in the *length* field.

RETURN VALUE

If *mm_l2p* was successful, then the physical start address is returned in *paddr*, the number of bytes remaining is returned in *length*, and 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tid*.

Unmapped logical address.

Task not created on local node.

ISR cannot reference remote node.

NOTES

Can be called from within an ISR, except when the task was not created on the local node.

Will not cause a preempt.

3.8.5 MM_P2L

NAME

mm_p2l – "Physical to Logical"

SYNOPSIS

```
#include <memory.h>
uint mm_p2l ( tid, paddr, &laddr, &length )

        uint tid;       /* task id as returned by t_create or t_ident */
        char *paddr;    /* physical start address */
        char *laddr;    /* logical start address - returned by this call */ */
        uint length;    /* remaining length in bytes - returned by this call */
```

DESCRIPTION

This directive returns the logical address within the section associated with the physical address belonging to the task identified by the *tid*. The executive will only return the first valid mapping of the physical address it finds, and the logical address returned may be ambiguous if the task has a many-to-one mapping of the physical address range.

The logical start address is returned in the *laddr* field, and the number of bytes remaining in the section is returned in the *length* field.

RETURN VALUE

If *mm_p2l* was successful, then the logical address is returned in *laddr*, the number of bytes remaining is returned in *length*, and 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tid*.

Unmapped logical address.

Task not created on local node.

NOTES

Not callable from ISR.

Will not cause a preempt.

3.8.6 MM_PMAP

NAME

mm_pmap -- "Map Physical"

SYNOPSIS

```
#include <memory.h>
uint mm_pmap ( tid, laddr, paddr, length, flags )

        uint tid;       /* task id as returned by t_create or t_ident */
        char *laddr;    /* logical start address */
        char *paddr;    /* physical start address */
        uint length;    /* length in bytes */
        uint flags;     /* section attributes */
```

The flags field values are defined as follows:

RDONLY	set	read-only
	clear	read-write

DESCRIPTION

This directive maps physical memory starting at *paddr* for the number of bytes specified in *length,* to a section at the logical start address *laddr* in the address space of the task identified by the *tid.*

The physical start address specified in *paddr* must be on the pagesize boundary. The logical start address specified in *laddr* must be on a section boundary.

If *length* is not a multiple of the pagesize, then more bytes than requested are mapped.

RETURN VALUE

If *mm_pmap* was successful, then 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tid.*

Paddr is not on a pagesize boundary.

Laddr is not on a section boundary.

Length specified is too large.