Cortland Memory Manager

First Draft 4/30/85
Revision 1 6/7/85

James Jatczynski Consulting
10204 Parkwood Drive #6
Cupertino, California 95014
408/446-3405

1. INTRODUCTION

In order coordinate activities within a computer system, it is customary to centralize management of scarce resources such as RAM, timers, disk space, interrupts, etc.

This document is a functional specification of the Memory Manager, one of the ROM based capability modules in Cortland. It is intended to serve as a model for additional capability modules.

2. REFERENCES

[1] *A Framework for Implementing ROM- and RAM-Based Capability Modules in Cortland*, J. Jatczynski, 6/4//85.

[2] *Cortland Capability Module Interface*, J. Jatczynski, 6/4/85.

[3] *SOS Reference Manual—Volumes 1 & 2*, Apple.

[4] *Standard Specification for Microprocessor Operating Systems Interfaces*, Draft, IEEE Task 855, Revision 6.0.

[5] *Inside Macintosh, Chapter 18, Memory Manager*, Apple Computer, Inc., 1985.

3. GOALS

The design tries to meet the following goals:

[1] Conformity to the specifications in References [1] and [2].

[2] Transparency: The Memory Manager makes no assumptions about memory usage, and thus it makes no automatic memory reservations.

[3] Adaptability: The Memory Manager determines what memory exists in the machine and manages all of it.

[4] Appropriate features: The Memory Manager provides features that use knowledge of the 65816 and Cortland memory structure to simplify the application program.

[5] Simplicity: The Memory Manager Provides only the most basic memory bookkeeping functions.

4. FUNCTIONAL SPECIFICATION

4.1 Overview

The Memory Manager (MM) provides a way for system and application programs to keep track of memory usage. The MM implements calls to initialize itself, to allocate, modify, and free contiguous blocks of memory, and to obtain information about the state of the MM. Since Cortland has no memory management hardware, correct memory management depends on cooperation among all memory users.

A major goal of the Memory Manager is simplicity. Thus, in contrast to the Macintosh Memory Manager (Ref. [5]), the Cortland Memory Manager provides only non-relocatable segments. Consequently, it does not support such concepts as *handle*, *purging*, and *locking*, which apply only to relocatable segments. As a result, the system places a larger burden on the user to manage

memory intelligently since it does not provide automatic free space consolidation. Also, in the interest of simplicity, the Memory Manager does not support Mac-like heap zones.

4.2 Operational Parameters

The Memory Manager's permanent capability module number is 2.

The MM allocates RAM in contiguous blocks called **segments**. Each segment consists of an integral number of fixed size **allocation units**, each of which contains 256 bytes aligned on a 256 byte (**page**) boundary. Thus, a segment is a page-aligned block of RAM containing an integral number of pages.

There is no predefined limit on the number of allocated segments or on the amount of memory that can be managed. The MM allocates table space for itself as needed to keep track of allocated segments and the available memory.

4.3 Structure of Calls

MM calls conform to the structure described in References [1] and [2]. All parameters and return values except the error code are passed and returned in the parameter block. The error code is returned in the A register and the C status bit. When a function returns with a non-zero error code, all of the return parameters are undefined. The first four bytes of every parameter block contain the flag and function identifier described in Ref. [2]; this document describes only the parameter bytes starting at offset 4 in the parameter blocks.

4.4 Description of Calls

4.4.1 Boot Initialize

This function initializes the MM at boot time by finding all available memory and initializing the MM's internal tables.

FN = 1.

There are no parameters beyond the function identifier at the beginning of the parameter list. This function must be executed before calls to any other MM functions.

Errors:

$10	Initialization failed

4.4.2 Application Startup

This function performs no operation, but is required by Ref. [1].

FN = 2.

Errors:

Always returns successfully.

4.4.3 Application Shutdown

This function performs no operation, but is required by Ref. [1].

FN = 3.

Errors:

Always returns successfully.

4.4.4 Allocate Segment

This function allocates a segment of specified length and location, a segment of specified length in a specified bank, or a segment of specified length anywhere in RAM. In the last case, the caller indicates whether or not the segment may straddle an interbank boundary. The segment is allocated only if free memory meeting the given constraints is available.

FN = 4.

The parameter block is as follows:

Offset	Name	Type
4	**mode**	value
5-6	**label**	value
7	**bank**	value
8-10	**length**	value/result
11-13	**base**	value/result
14-15	**segnum**	result

Mode describes the nature of the memory allocation. Legal forms are

Bits 76543210	Function
00------	allocate segment of given length and given base address
01L-----	allocate segment of given length completely within a specified bank
10LS----	allocate segment of given length anywhere in free memory
----RRRR	4 low-order bits reserved by Apple—must be zero

Hyphens indicate bits that are not used. L and S have the following significance:

L=0	allocate a segment of the specified size
L=1	allocate the largest available segment that meets the other constraints
S=0	allocated segment must not straddle interbank boundary
S=1	allocated segment may straddle interbank boundary

The bits indicated by RRRR may be used in future versions of the memory manager to specify certain attributes to be attached to the allocated segment.

Label is a value specified by the caller to encode arbitrary information about the segment. The MM

remembers this value but does nothing with it.

Bank is used only in mode 01L-----. It tells the MM the bank in which to allocate the segment. It is ignored in all other modes.

As an input value, **length** gives the length of the segment to be allocated in bytes. The input value is ignored when L=1. In all other cases, the MM rounds this field up to the nearest allocation unit boundary. In all modes, **length** is used as a result field that gives the length of the allocated segment in bytes. The segment will always contain an integral number of allocation units.

As an input value, **base** is used only in mode 00------ to specify the address of the lowest byte in the segment to be allocated. The MM sets enough low order bits to zero so that the segment will be aligned on an allocation unit boundary. In all modes, **base** is used as a result to specify the base address of the allocated segment.

Segnum is a result assigned by the MM to identify the segment in subsequent calls.

The default search order established at initialization is as follows: Search the banks in order from highest bank number toward lowest bank number. Within each bank, search from higher addresses toward lower addresses. Choose the first free block that is big enough to accommodate the request ("first fit") and satisfies the other constraints. When the segment must be in a specific bank, search only that bank from high end toward low end.

Errors:

$10	Initialization failed or was never performed
$11	Invalid **mode** parameter
$12	Memory unavailable
$13	Memory manager ran out of table space

4.4.5 Modify Segment

This function increases or decreases the size of an already allocated segment by adding or removing allocation units at the low end or high end of the segment. A segment can be increased in size only if adjacent space of the specified size is available.

FN = 5.

The parameter block is as follows:

Offset	Name	Type
4-5	**segnum**	value
6	**mode**	value
7-9	**length**	value

Segnum identifies the segment to be modified.

Mode tells how the segment is to be modified:

00------	reduce size by freeing allocation units at lower addresses
01------	reduce size by freeing allocation units at higher addresses
10-S----	increase size by adding allocation units at lower addresses

11-S---- increase size by adding allocation units at higher addresses

S has the following significance:

S=0 the additional storage must not straddle an interbank boundary
S=1 the additional storage may straddle an interbank boundary

Length gives the number of bytes by which the segment is to be reduced or increased in size. The MM rounds **length** upward to the nearest integral allocation unit.

Errors:

$10 Initialization failed or was never performed
$11 Invalid **mode** parameter
$12 Memory unavailable
$13 Memory manager ran out of table space
$14 Invalid **segnum** value

4.4.6 Free Segment

This function returns a currently allocated segment to the pool of available RAM.

FN = 6.

The parameter block is as follows:

Offset	Name	Type
4-5	**segnum**	value

Segnum identifies the segment to be returned to the pool of available memory. After this call, the value of **segnum** no longer refers to the freed segment.

Errors:

$10 Initialization failed or was never performed
$14 Invalid **segnum** value

4.4.7 Get Segment Number

This function returns the segment number of the segment containing a specified memory address.

FN = 7.

The parameter block is as follows:

Offset	Name	Type
4-6	**address**	value
7-8	**segnum**	result

Address is an arbitrary RAM or ROM address.

Segnum identifies the segment (if any) containing the given address..

Errors:

$10	Initialization failed or was never performed
$15	Given address not contained in any segment

4.4.8 Get Segment Information

This function returns information about a specified segment.

FN = 8.

The parameter block is as follows:

Offset	Name	Type
4-5	**segnum**	value
6-7	**label**	result
8-10	**length**	result
11-13	**base**	result

Segnum identifies the segment whose information is to be retrieved.

Label is the label value recorded when the segment was allocated.

Length is the segment length in bytes.

Base is the base address of the segment.

Errors:

$10	Initialization failed or was never performed
$14	Invalid **segnum** value

4.4.9 Get Search Order

This function returns the number of RAM banks and a pointer to a table that lists bank numbers in the order they are searched by the MM. The calling program may reorder the table entries as desired. Subsequent calls to **allocate segment** will use the new search order. The caller should disable interrupts while reordering the table. *This is a dangerous call that should be used only by system software.*

FN = 9.

The parameter block is as follows:

Offset	Name	Type
4-5	**numbanks**	result
6-8	**searchtable**	pointer result

Numbanks gives the number of 64Kby RAM banks in the machine.

Searchtable points to the MM table that lists the search order of the banks. The table contains **numbanks** one-byte entries, each of which gives a bank number. When processing a call to **allocate segment**, the MM searches the banks for free space in the order given in this table. The caller may reorder the table as desired.

Errors:

$10 Initialization failed or was never performed

4.5 Error Summary

Whenever a function returns a non-zero error code, all result parameters in the parameter list are undefined.

$10 Initialization failed or was never performed

If the initialization function fails or is never executed, none of the other functions will work.

$11 Invalid **mode** parameter

The **mode** parameter of either the allocate or modify function is improperly specified.

$12 Memory unavailable

A memory segment cannot be allocated because the specified area of RAM is either non-existent or already allocated, or there is insufficient available memory.

$13 Memory manager ran out of table space

The memory manager cannot perform the requested function because it has run out of internal table space. This will happen only when there is almost no free memory available.

$14 Invalid **segnum** value

The given value of **segnum** does not correspond to any allocated segment.

$15 Given address not contained in any segment

The given value of **address** is not contained in any allocated segment.

4.6 Additional Functionality

Several other useful functions can be performed by using combinations of the existing functions. These are not implemented directly because they are used infrequently.

For example, a program can determine the size of the alloction unit by attempting to allocate a segment of length 1 byte. If the call is successful, the **length** field of the **allocate segment** parameter block will return the number of bytes in the allocation unit. The program should, of course, execute the **free segment** function to return the memory to the available pool.