Identification

The Core Management Module
P. G. Neumann, M. R. Wagner, and R. C. Daley

Purpose

This section describes the operation of the core management module and the calls used in the allocation and deallocation of core memory. During normal Multics operation all requests for core allocation are directed to the core management module by page control (see BG.4). The core management module is the only module which manipulates the core map during normal operation.

Summary

The following list summarizes the various entries in the core management module. The first entry (core_man$init) is called only once during system initialization while the remaining entries are called only by the modules of page control. A detailed description of each entry and its arguments follow the summary.

core_man$init	used for initialization.
core_man$assign	assigns a block of core.
core_man$unassign	unassigns a block of core.
core_man$wire	changes status to wired.
core_man$unwire	changes status to removable.
core_man$get_type	returns current status of a block of core.

The Initialization Entry

In order to initialize certain internal static pointers in the core management module, an initialization entry is provided. This entry is invoked only once during system initialization by the following call.

```
call core_man$init;
```

This call is made by the init_core_control procedure (see BG.5.03) before any other calls are made to the core management module.

The assign Entry

The assign entry is invoked to allocate a single block of core from either the 1024-word or 64-word core pool. This entry is called by interim_1_segfault (see BL.5.03) during system initialization and by page control during normal operation. The assign entry is invoked by the following call.

```
     call core_man$assign (loc, stat, sps, page, sstp, rp1);
```

The arguments used in these calls are described as follows.

loc- is the high order 18 bits of the absolute 24 bit memory location of the assigned block of core. This item is returned to the caller upon completion of the assign call.

stat- indicates the status the assigned block of core is to have after assignment (0=removable, 1=wired, 2=permanent and 3=temporary). The core management module adds 2 to this item to obtain the desired type code as described in BG.5.01.

sps- is a switch indicating if ON (1) that a 64-word block is requested and if OFF (0) that a 1024-word block is requested.

page- is the page number if the requested block is to be used as a page of a pageable segment. This item is saved in the block map entry for the block assigned by this call.

sspt- is a pointer to the SST entry (if any) which defines the segment for which the assigned block of core is to be used as a page. This item is also saved in the block map entry.

rp1- is a switch indicating if ON (1) that removable pages (i.e., assigned to the removal list) may be removed to satisfy the assign call, and if OFF (0) that no pages may be removed during the assign call.

Upon receiving the assign call, core management attempts to assign the first entry in the appropriate (64-word or 1024-word) free list. If this attempt fails or if the amount of free core after assignment falls below a predetermined threshold the rp1 argument is checked.

If rp1 indicates that pages may be removed from core at this time, the removal algorithm (see below) is invoked to free a predetermined number of pages from the appropriate removal list. If at this point the request has been satisfied, core management fills in the block map entry for the assigned block, threads this entry into the appropriate list as indicated by the stat argument, turns the initial usage switch ON (1) in the assigned block map entry and returns the assigned core location to the caller.

If, after the above steps are taken, there is still no available free block with which to satisfy the request, one of the following actions is taken depending on the requested block size. If the requested block size is 1024, core management calls page control to update the status of outstanding page output requests and tries again to assign the 1024-word block. This strategy is continued until a previously requested page removal is completed and a page is unassigned. This looping while waiting for a page to be unassigned is controlled by the number of entries core management attempts to maintain on the free list and should happen rarely under normal operating conditions.

If the requested block size is 64 and the 64-word free list is empty, the core management module reissues the request requesting a 1024-word block and uses the assigned 1024-word block to satisfy the original request.

The Removal Algorithm

Whenever a block of core is assigned to a removable page, the corresponding block map entry is placed at the front of the removal list and the initial usage switch is set indicating that this entry has just been assigned to the removal list. When the removal algorithm is invoked by the assign entry, the algorithm considers the first entry on the removal list. If this entry has recently been assigned (i.e., its initial usage switch is ON) the intial usage switch and the corresponding page use switch in the page table word are set OFF. The entry is then placed at the end of the removal list and will not be considered again until the remainder of the removal list is scanned.

If the first entry in removal list has not recently been assigned (its initial usage switch if OFF) then the corresponding page use switch is tested and reset. If the page has been used since the last time it was considered for removal, the block map entry is then placed at the end of the removal list. If the page has not been used, a call is made to page control to request the removal of the page from core.

The removal algorithm continues to scan through the removal list removing pages and placing entries at the end of the removal list until the required number of pages have been removed. In the worst case (i.e., all pages have been used) the entire removal list is scanned before any pages are removed.

The unassign Entry

The unassign entry is invoked to deallocate a single block of core and place it in the appropriate free list. The entry is called exclusively by page control by means of the following call.

```
call core_man$unassign(loc);
```

In this call loc is again the 18-bit absolute memory location of the block to be unassigned but in this case is specified by the caller. Upon receiving this call, the block map entry corresponding to loc is removed from its current list and placed in the appropriate (1024-word or 64-word) free list.

The wire Entry

The wire entry is invoked to change the status of a currently assigned block of core from removable to wired. This entry is used only by page control by means of the following call.

```
call core_man$wire(loc);
```

In this call loc has the same meaning as in the unassign call. Upon receiving this call, the corresponding block map entry is removed from its current list and placed in the appropriate wired list.

The unwire Entry

The unwire entry is invoked to change the status of a currently assigned block of core from wired to removable. This entry is used only by page control by means of the following call.

```
call core_man$unwire(loc);
```

In this call loc has the same meaning as in the unassign call. Upon receiving this call, the corresponding block map entry is removed from its current list and placed in the appropriate removal list.

The get_type Entry

The get type entry is invoked to obtain the current status of a currently assigned block of core. This entry is used by page control by means of the following call.

```
          call core_man$get_type(loc, stat);
```

In this call, loc has the same meaning as in the unassign call. Upon return from this call, stat will contain the current status of the corresponding core block (0=removable, 1=wired, 2=permanent, 3=temporary).

Declaration of Parameters

The parameters used in the above calls to the core management module are declared below.

```
     dcl  loc fixed bin (18),
          stat fixed bin (2),
          sps fixed bin (1),
          page fixed bin (8),
          sstp ptr,
          rp1 fixed bin (1);
```