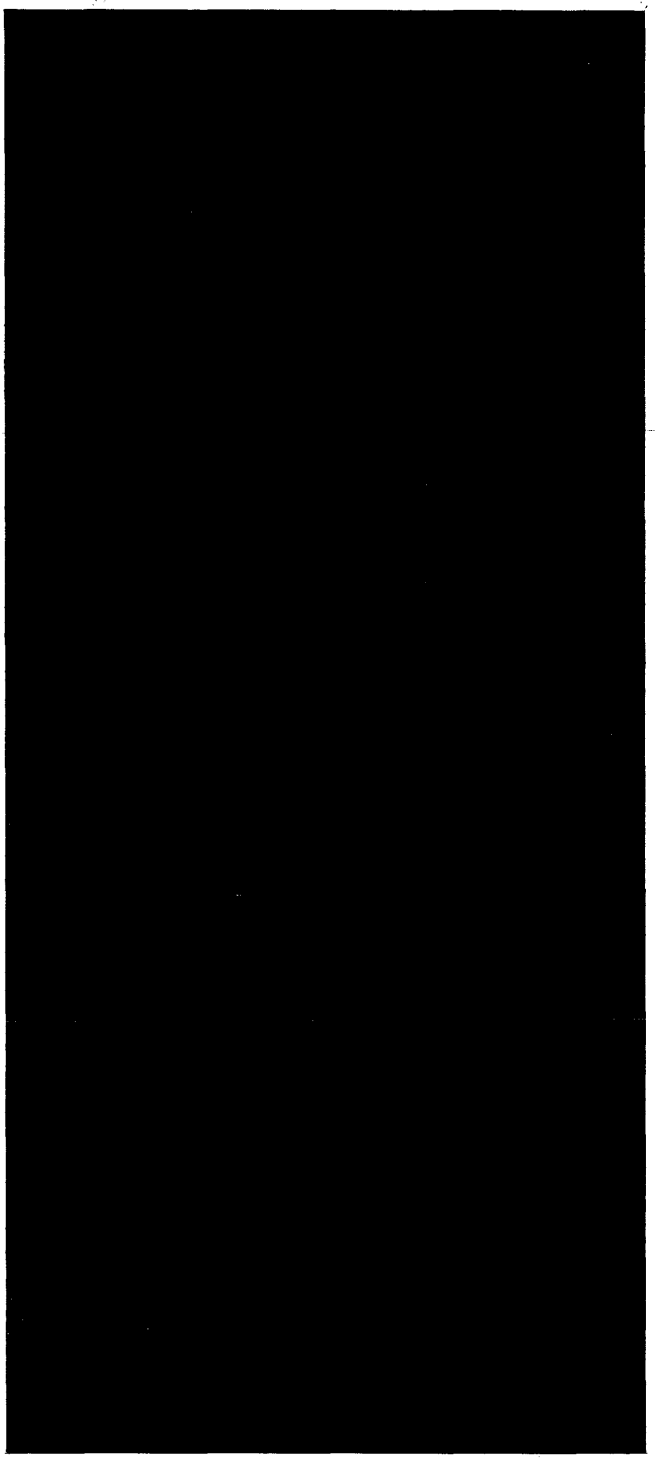
Honeywell

MULTICS RECONFIGURATION PROGRAM LOGIC MANUAL

SERIES 60 (LEVEL 68)

SOFTWARE



RESTRICTED DISTRIBUTION

Honeywell

MULTICS RECONFIGURATION PROGRAM LOGIC MANUAL

SERIES 60 (LEVEL 68)

RESTRICTED DISTRIBUTION

SUBJECT:

Dynamic Reconfiguration Software for the Major Hardware Modules (Processor, System Controller, and Bulk Store).

SPECIAL INSTRUCTIONS:

This Program Logic Manual (PLM) describes certain internal modules constituting the Multics System. It is intended as a reference for only those who are thoroughly familiar with the implementation details of the Multics operating system; interfaces described herein should not be used by application programmers or subsystem writers; such programmers and writers are concerned with the external interfaces only. The external interfaces are described in the Multics Programmers' Manual, Commands and Active Functions (Order No. AG92), Subroutines (Order No. AG93), and Subsystem Writers' Guide (Order No. AK92).

As Multics evolves, Honeywell will add, delete, and modify module descriptions in subsequent PLM updates. Honeywell does not ensure that the internal functions and internal module interfaces will remain compatible with previous versions.

This PLM is one of a set, which when complete, will supersede the System Programmers' Supplement to the Multics Programmers' Manual (Order No. AK96).

DATE:

April 1977

ORDER NUMBER:

AN71, Rev. 1

PREFACE

Multics Program Logic Manuals (PLMs) are intended for use by Multics system maintenance personnel, development personnel, and others who are thoroughly familiar with Multics internal system operation. They are not intended for application programmers or subsystem writers.

The PLMs contain descriptions of modules that serve as internal interfaces and perform special system functions. These documents do not describe external interfaces, which are used by application and system programmers.

Since internal interfaces are added, deleted, and modified as design improvements are introduced, Honeywell does not ensure that the internal functions and internal module interfaces will remain compatible with previous versions. To help maintain accurate PLM documentation, Honeywell publishes a special status bulletin containing a list of the PLMs currently available and identifying updates to existing PLMs. This status bulletin is distributed automatically to all holders of the System Programmers' Supplement to the Multics Programmers' Manual (Order No. AK96) and to others on request. To get on the mailing list for this status bulletin, write to:

Large Systems Sales Support
Multics Project Office
Honeywell Information Systems Inc.
Post Office Box 6000 (MS A-85)
Phoenix, Arizona 85005

 File No.: 2L13

AN71

CONTENTS

		Page
Section I	Introduction	1-1
Section II	Terminology	2-1
	system controller	2-2
	memory controller	2-2
	memory	2-2
	controller	2-2
	processor	2-2
	system controller port	2-2
	active module port	2-2
	port enable register	2-2
	interrupt register	2-3
	interrupt cell	2-3
	interrupt mask	2-3
	interrupt mechanism	2-4
	bootload controller	2-4
	system interrupt	2-4
	bootload processor	2-4
	BOS processor	2-4
	interrupt processor	2-5
	processor tag	2-5
	internal interlace	2-5
	external interlace	2-5
	main memory frame	2-6
	main memory map	2-6
	core map	2-6
	main memory used list	2-6
	used list	2-6
	page	2-6
	record	2-6
	abs_usable	2-6
	abs_wired	2-7
	paging device map	2-7
	pdmap	2-7
	read/write sequence	2-7
	rws	2-7
Section III	Data Structures	3-1
	Processor and System Controller Reconfiguration Structures	3-1
	Processor Reconfiguration Structures	3-4

		Page
	System Controller Addressing Segment	3-8
	Main Memory and Paging Device Maps	3-9
Section IV	Data Base Initialization	4-1
	SCS Initialization	4-1
	SCAS Initialization	4-2
	SST Initialization	4-2
	Other Data Base Initialization	4-3
Section V	Hardcore Reconfiguration Entries	5-1
	Reconfiguration Entries	5-1
	hphcs_$add_cpu	5-1
	hphcs_$del_cpu	5-2
	hphcs_$add_mem	5-2
	hphcs_$del_mem	5-3
	hphcs_$add_main	5-3
	hphcs_$del_main	5-4
	hphcs_$reconfig_info	5-4
	hphcs_$rc_force_unlock	5-5
	Error Codes	5-5
Section VI	Processor Reconfiguration	6-1
	Idle Processes	6-1
	Adding a Processor	6-2
	Removing a Processor	6-4
Section VII	Memory Reconfiguration	7-1
	Adding Main Memory	7-1
	Adding a System Controller	7-2
	Removing a System Controller	7-2
	Removing Main Memory	7-3
	Automatic Memory Removal	7-4
Section VIII	Bulk Store Reconfiguration	8-1
	Bulk Store Initialization	8-1
	Bulk Store Reconfiguration Entries	8-2
	hphcs_$delete_pd_records	8-2
	hphcs_$add_pd_records	8-2
	Adding Bulk Store Records	8-3
	Removing Bulk Store Records	8-4
	Automatic Paging Device Record Removal	8-5
Section IX	The Command Interface	9-1

SECTION I

INTRODUCTION

This document describes the implementation and design of the Multics dynamic reconfiguration software for the major hardware modules of the system. This document is limited to processor, system controller and bulk store memory reconfiguration although there are many more hardware and software switchable modules in the system.

Dynamic reconfiguring, on a per-module basis, is done only in response to explicit operator request. The facility of the system that automatically deconfigures selected subregions of main memory or bulk store when hardware problems arise uses the same basic mechanism as module deconfiguration where appropriate. There is currently no way the system will automatically deconfigure a faulty processor. The software to automatically deconfigure main memory is incomplete. The software to automatically deconfigure a faulty record of the bulk store is operational.

Two types of system controller can be used for Multics operation: the 6000 system controller (MC6000) and the four megaword system controller unit (SCU003). These system controllers can be intermixed in any way in a Multics configuration. They are hereafter referred to as the 6000 SC and the 4MW SCU, respectively. Basic differences between the 6000 SC and the 4MW SCU are described in the next section.

SECTION II

TERMINOLOGY

Terms and phrases frequently used in discussions of dynamic reconfiguration are defined on the following pages in logical order. They are listed below in alphabetical order for convenience.

BOS processor	2-4
abs_usable	2-6
abs_wired	2-7
active module port	2-2
bootload controller	2-4
bootload processor	2-4
controller	2-2
core map	2-6
external interlace	2-5
internal interlace	2-5
interrupt cell	2-3
interrupt mask	2-3
interrupt mechanism	2-3
interrupt processor	2-4
interrupt register	2-3
main memory frame	2-6
main memory map	2-6
main memory used list	2-6
memory	2-2
memory controller	2-2
page	2-6
paging device map	2-7
pdmap	2-7
port enable register	2-2
processor	2-2
processor tag	2-5
read/write sequence	2-7
record	2-6
rws	2-7
system controller	2-2
system controller port	2-2
system interrupt	2-4
used list	2-6

system controller
: A passive hardware module that interfaces active modules to the main memory of the configuration. The system controller manages system interrupts, passes connect signals from one active module to another, contains the system calendar clock, and provides main memory functions to its active users. A system controller may be either an 6000 SC or a 4MW SCU.

memory controller
: See system controller above.

memory
: See system controller above.

controller
: See system controller above.

processor
: One of the three types of active modules. (The other two are the IOM and bulk store controller.) The processor is the major processing unit (CPU).

system controller port
: A point on a system controller for connection to an active module. There are eight ports on a system controller. Each system controller contains hardware to enable or disable requests over each of its ports. Only active modules connected to enabled system controller ports can interact with that system controller.

active module port
: A point on an active module for connection to a system controller. Each active module has eight ports controlled by port logic. The port logic maps an absolute address generated by an active module into a port number and an address within the memory associated with a system controller.

port enable register
: An eight-bit mask register associated with each system controller that contains one bit for each controller port. If a bit is on, the active module on the corresponding controller port can use the system controller. If it is off, the active module will receive a fault condition if it attempts to access the system controller.

On a 6000 SC, port enable register bits can be forced on or off by eight configuration panel switches. These switches have three positions: ENABLE, DISABLE, and PROG CONTROL. If a switch is in PROG CONTROL position, the corresponding port enable register bit can be turned on or off by system software. For normal Multics operation, all eight switches are usually set to the PROG CONTROL position.

The 4MW SCU configuration panel contains eight two-position switches. These switches are read into the port enable register only when the system controller is initialized. At all other times, port enable register bits must be set by system software.

interrupt register
A 32-bit register associated with each system controller. Active modules can instruct the system controller to set any of these bits. When one or more bits in the interrupt register are set, the system controller will attempt to notify one or more active modules that an execute interrupt is present (XIP). This is described in more detail below.

interrupt cell
A single bit of the interrupt register. Interrupt cells are numbered from 0 to 31.

interrupt mask
Interrupt masks are used to allow or prevent the receipt of XIP signals by processors. (See interrupt mechanism.)

Each 6000 SC contains four 32-bit interrupt mask registers. Each 4MW SCU contains only two. Each interrupt mask register is assigned to a particular controller port through the use of the execute interrupt mask assignment (EIMA) switches. On the 4MW SCU, the EIMA switches are read into internal mask assignment registers when the system controller is initialized. Interrupt mask register assignments can be changed by system software. On the 6000 SC, no software changes can be made to the mask assignments.

A processor can read or set an interrupt mask register assigned to its system controller port through the use of the RMCM and SMCM instructions. Processors can also read and set interrupt mask registers assigned to other ports through the use of the RSCR and SSCR instructions. On the 6000 SC, a processor may change other interrupt mask registers only if it has a mask register assigned to its own controller port.

interrupt mechanism

When one or more bits are set in a system controller's interrupt register, the controller examines all interrupt mask registers for matching bits. (When a bit is on in an interrupt mask register, the active module to which the mask is assigned is said to have the corresponding interrupt "unmasked".) The system controller will send an XIP signal to all active modules with unmasked interrupts set in the interrupt register.

Active modules respond to the XIP signal by interrogating the system controller about the interrupt. The system controller will find the highest priority (lowest cell number) unmasked interrupt set and return the interrupt cell number to the requesting active module. The system controller will clear the interrupt cell at that time. If more than one active module responds to an XIP signal, only one will receive information pertaining to a particular interrupt cell. If a set interrupt is not unmasked by any active module, the interrupt will be retained until some active module unmasks that interrupt.

bootload controller

The system controller containing low-order main memory in a system configuration. All interrupts sent by active modules are sent via the interrupt register in the bootload controller. No other system controllers convey interrupts to active modules. The bootload system controller cannot be removed while the system is running since it contains fault and interrupt vectors, IOM, bulk store, and DataNet 6600 FNP mailboxes, and unpaged segments.

system interrupt

An interrupt required by the system in order to carry out its orderly functions of communicating between I/O devices and Multics processes. (In addition, there is one interrupt which is sent by a processor to start up a new processor.) All I/O interrupts are set in the bootload system controller. I/O interrupts will be sent to processors selected by the EIMA switches on the bootload system controller.

bootload processor (or BOS processor)

The central processor used to initialize Multics and to shutdown and return to BOS at the end of Multics operation. It is also the processor used to enter BOS after a system crash. The bootload processor may be dynamically deconfigured, at which time another processor will be made the new bootload processor.

interrupt processor
Processors that can receive interrupts (i.e. that have assigned interrupt masks in the bootload system controller). When a 6000 SC is used as the bootload controller, four interrupt masks are available; hence, all four processors in such a Multics configuration are interrupt processors. When a 4MW SCU is the bootload controller, only two processors can receive interrupts. System software will reassign the interrupt mask register of an interrupt processor which is being dynamically removed to a processor which is not an interrupt processor.

processor tag
A processor identification corresponding directly to the processor number. Processors have two switches on their configuration panels which allow the setting of a two-bit processor number. This number can be read by the RSW instruction. Each processor in a Multics configuration must have a different processor number. The processor number corresponds to the processor tag on the configuration card for that processor.

Processor Number	Processor Tag
00	A
01	B
10	C
11	D

A maximum of four processors can be configured to a Multics system.

internal interlace
A system controller feature which allows interleaving of double-words between the low-order and high-order store units connected to the controller. Except for timing changes, internal interlace is invisible to all active modules. Only system controllers with low-order and high-order store units of the same size can be internally interlaced.

external interlace
The port logic of each active module allows the main memory of two system controllers on even/odd active module port pairs to be interlaced. Interlacing may be done at either two words at a time or four words at a time. Only system controllers containing stores of the same size can be externally interlaced. Four word external interlace may be combined with internal interlace to provide a four-way interlace mechanism. Of necessity, all active modules must have their interlace switches set in the same positions.

main memory frame
: A contiguous region of main memory that is one page in length and starts on a page boundary. All of main memory is thus divided into fixed length regions the size of a page. Some main memory frames contain pages which are permanently wired. These frames can never contain paged data. Other main memory frames contain data or code which is "temp wired" (i.e., is temporarily forced to remain in main memory). A temp wired main memory frame may later be freed up and reused for some other page. The term "wired" applies to anything which must remain in main memory for some time for some reason. The terms "latched", "locked" and "core resident" are also used in the literature for what is here called wired.

main memory map (or core map)
: An array of entries for all main memory frames that can ever be configured into the system. The main memory map is indexed by absolute main memory frame numbers. A main memory map entry (often called a core map entry or CME) describes which page, if any, is currently occupying the associated main memory frame.

main memory used list (or used list)
: A threaded list of main memory map entries for the main memory frames in the paging pool.

page
: A 1024-word extent of data beginning at a 1024-word boundary of a segment. Pages belong to segments; they can reside in main memory frames, secondary storage records, or both.

record
: A contiguous region of a secondary storage device that begins on a page boundary and is one page long. Satisfying a page fault, for example, consists in moving the data of a page from a given record of secondary storage to a given frame of main memory and performing the necessary connections.

abs_usable
: That attribute of a main memory frame which permits the main memory to be used for I/O. This concept is needed by several hardcore I/O procedures since they must set up DCW lists which have absolute addresses in them. The main memory frames of the bootload controller can not be dynamically deconfigured (for several unrelated reasons), and therefore, all main memory frames of the bootload controller which are part of the paging pool are marked as abs_usable. In addition, main memory frames of other system controllers will also be so marked if there are not enough abs_usable frames in the bootload controller.

abs_wired A frame of main memory that contains a page that is wired down because it may contain locations that are absolutely addressed. Such a page cannot be moved, either to make room for another abs_wired page or to deconfigure the controller. Any controller that contains one abs_wired page can not be dynamically deconfigured until that page is no longer required to be abs_wired.

paging device map (or pdmap)
A map, analogous to the main memory map, used as part of the bulk store management algorithms. The paging device map is ordered according to time of recent reference and hence is the key to the bulk store replacement algorithm.

read/write sequence (or rws)
The mechanism used to move a modified page from the bulk store to secondary storage. This mechanism consists of finding a frame of main memory, reading in the page from the bulk store, and then writing the page out to secondary storage.

SECTION III

DATA STRUCTURES

The several key data structures used by the reconfiguration software are kept in the segments SCS and SST. These are initialized as described in Section IV and modified as described in Sections VI, VII and VIII.

PROCESSOR AND SYSTEM CONTROLLER RECONFIGURATION STRUCTURES

The following declarations of data structures describe structures that are used both during processor and system controller reconfiguration:

```
declare 1 scs$controller_data (0: 7) aligned ext,
          2 size fixed bin(17) unaligned,
          2 base fixed bin(17) unaligned,
          2 eima_data (4) unaligned,
            3 mask_available bit(1) unaligned,
            3 mask_assigned bit(1) unaligned,
            3 mbz bit(3) unaligned,
            3 mask_assignment fixed bin(3) unaligned,
          2 info aligned,
            3 online bit(1) unaligned,
            3 offline bit(1) unaligned,
            3 store_a_online bit(1) unaligned,
            3 store_a1_online bit(1) unaligned,
            3 store_b_online bit(1) unaligned,
            3 store_b1_online bit(1) unaligned,
            3 store_b_is_lower bit(1) unaligned,
            3 ext_interlaced bit(1) unaligned,
            3 int_interlaced bit(1) unaligned,
            3 four_word bit(1) unaligned,
            3 cyclic_priority (7) bit(1) unaligned,
            3 type bit(4) unaligned,
            3 abs_wired bit(1) unaligned,
            3 program bit(1) unaligned,
            3 pad bit(13) unaligned,
          2 lower_store_size fixed bin(17) uanligned,
          2 upper_store_size fixed bin(17) unaligned;
declare scs$reconfig_lock bit(36) aligned ext;
```

```
declare scs$reconfig_locker_id char(32) aligned ext;
declare scs$interrupt_controller fixed bin(3) ext;
declare scs$port_addressing_word (0: 7) bit(3) aligned ext;
```

The variables declared above have the following meanings:

1. controller_data
   is an array, indexed by system controller tag, containing information for each system controller.

2. controller_data.size
   is the size, in 1024-word frames, of the main memory contained in each system controller.

3. controller_data.base
   is the base address, modulo 1024 words, of the main memory contained in each system controller.

4. controller_data.eima_data
   is an array containing the execute interrupt mask assignment (EIMA) switch settings for each system controller.

5. eima_data.mask_available
   is set to "1"b if the corresponding interrupt mask is available on a system controller.

6. eima_data.mask_assigned
   is set to "1"b if the corresponding interrupt mask is assigned to a processor port.

7. eima_data.mask_assignment
   is the system controller port to which the interrupt mask is assigned.

8. controller_data.online
   if equal to "1"b, indicates that the corresponding system controller is online and in use.

9. controller_data.offline
   if equal to "1"b, indicates that the corresponding system controller is offline, but could be dynamically added at a later time.

10. controller_data.store_a_online
    is equal to "1"b if store "A" of the corresponding system controller is online and in use.

11. controller_data.store_a1_online
    is equal to "1"b if store "A1" of the corresponding system controller is online and in use.

12. controller_data.store_b_online
    is equal to "1"b if store "B" of the corresponding system controller is online and in use.

13. controller_data.store_b1_online
    is equal to "1"b if store "B1" of the corresponding system controller is online and in use.

14. controller_data.store_b_is_lower
    is equal to "1"b if store "B" (and store "B1", if present) is the lower order store for a given system controller.

15. controller_data.ext_interlaced
    is set to "1"b if the corresponding system controller is interlaced with a system controller on an adjacent active module port.

16. controller_data.int_interlaced
    is set to "1"b if the two stores of the corresponding system controller are internally interlaced.

17. controller_data.four_word
    is set to "1"b if two adjacent system controllers are interlaced every four words. It is set to "0"b if they are interlaced every two words. If controller_data.ext_interlaced is equal to "0"b, this bit is meaningless.

18. controller_data.cyclic_priority
    is an array of bits giving the cyclic port priority ("anti-hogging") switch settings for each system controller.

19. controller_data.type
    is a code giving the type of system controller. If it is greater than or equal to "0010"b, the controller is a 4MW SCU. Otherwise, it is a 6000 SC.

20. controller_data.abs_wired
    is set to "1"b if abs wired pages are contained in the main memory associated with the corresponding system controller.

21. controller_data.program
    is "1"b if the controller is a 4MW SCU and is in programmable mode. Multics requires this bit to be on.

22. controller_data.lower_store_size
    is the size, in 1024-word frames, of the lower of the two stores connected to the corresponding system controller.

23. controller_data.upper_store_size
    is the size, in 1024-word frames, of the upper of the two stores connected to the corresponding system controller.

24. reconfig_lock
    is the lock used to prevent simultaneous attempts by several processes to perform dynamic reconfiguration.

25. reconfig_locker_id
    is the process group ID of the process which has set the reconfig_lock.

26. interrupt_controller
    is the tag of the bootload system controller. All interrupts go through this system controller.

27. port_addressing_word
    is an indirect word needed to access a given processor port (and thus a given system controller) by certain processor instructions such as RMCM, SMCM, and RCCL.

PROCESSOR RECONFIGURATION STRUCTURES

The following declarations of data structures describe structures that are primarily used during processor reconfiguration:

```
declare 1 scs$processor_data (0: 3) ext aligned,
          2 online bit(1) unaligned,
          2 offline bit(1) unaligned,
          2 pad1 bit(2) unaligned,
          2 delete_cpu bit(1) unaligned,
          2 interrupt_cpu bit(1) unaligned,
          2 halted_cpu bit(1) unaligned,
          2 pad2 bit(27) unaligned,
          2 controller_port fixed bin(3) unaligned;
declare scs$processor_start_int_no fixed bin(5) ext;
declare scs$processor_start_pattern bit(36) aligned;
declare scs$processor_start_mask bit(72) aligned ext;
declare scs$set_mask (0: 3) bit(36) aligned ext;
declare scs$read_mask (0: 3) bit(36) aligned ext;
declare scs$mask_ptr (0: 3) ptr unaligned ext;
declare scs$nprocessors fixed bin ext;
declare scs$bos_processor_tag fixed bin(2) ext;
declare scs$processor bit(4) aligned ext;
declare scs$processor_switch_template (4) bit(36) aligned ext;
declare scs$processor_switch_compare (4) bit(36) aligned ext;
declare scs$processor_switch_mask (4) bit(36) aligned ext;
```

The variables declared above have the following meanings:

1. processor_data
   is an array, indexed by processor tag, of information for each possible processor that can be configured in a Multics system.

2. processor_data.online
   if equal to "1"b, indicates that the corresponding processor is online and running.

3. processor_data.offline
   if equal to "1"b, indicates that the corresponding processor is offline, but could be dynamically added at a later time.

4. processor_data.delete_cpu
   is set to "1"b by the processor reconfiguration software when it is desired to dynamically remove the corresponding processor.

5. processor_data.interrupt_cpu
   is set to "1"b if the corresponding processor has an interrupt mask assigned to it in the bootload system controller.

6. processor_data.halted_cpu
   is set to "1"b after the corresponding processor has been successfully dynamically removed. This bit is also set to "1"b for all processors at the beginning of Multics system initialization.

7. processor_data.controller_port
   is the system controller port to which the corresponding processor is connected. Note that the port number occupies bits 33 through 35 of the word containing processor_data. This enables the entire word to be used as a Connect Operand Word (COW) when sending connects to a particular processor.

8. processor_start_int_no
   is the interrupt cell number used to start up a processor that is being dynamically added. This interrupt cell number is assigned by the system initialization software.

9. processor_start_pattern
   is the bit pattern used to set the processor start interrupt with a SMIC instruction.

10. processor_start_mask
    is the system controller interrupt mask used to allow a processor to take the processor start interrupt but to mask all other interrupts.

11. set_mask
    is an array of instructions used to set interrupt masks for corresponding processors. If a processor has an assigned mask, the corresponding element of set_mask will contain an SMCM instruction. Otherwise, it will contain an STAQ instruction into a software simulated mask register.

12. read_mask
    is an array of instructions used to read interrupt masks for corresponding processors. If a processor has an assigned mask, the corresponding element of read_mask will contain an RMCM instruction. Otherwise, it will contain an LDAQ instruction from a software simulated mask register.

13. mask_ptr
    is an array of packed pointers used to set and read interrupt masks for corresponding processors. If a processor has an assigned mask, the corresponding element of mask_ptr will point to scs$port_addressing_word for the bootload system controller. Otherwise, it will point to the simulated mask register located at prds$simulated_mask.

14. nprocessors
    is the number of processors currently online and running.

15. bos_processor_tag
    is the processor tag of the processor that was the bootload processor when Multics was bootloaded. If the original bootload processor has been dynamically removed, bos_processor_tag will be set to the tag of a processor which will be used to return to BOS when Multics is shut down.

16. processor
    contains one bit for each processor. The bit corresponding to a processor will be equal to "1"b if that processor is online and running.

17. processor_switch_template
    is an array containing template values for the processor switches read by the RSW 1 through RSW 4 instructions. This array is used to verify the configuration switch settings of a processor when attempting to dynamically add that processor.

18. processor_switch_compare
    is an array containing discrepancies between the expected and actual data read by the RSW 1 through RSW 4 instructions.

19. processor_switch_mask
    is an array containing masks for the processor switches read by the RSW 1 through RSW 4 instructions. The processor_switch_compare data is produced by exclusive ORing the processor_switch_template data with the actual configuration switch settings and ANDing the result with the processor_switch_mask data.

In addition to the structures described above, there are several structures contained in the procedure "init_processor" which are set, examined, or used during processor reconfiguration. Note that init_processor is an impure procedure.

```
declare init_processor$wait_flag fixed bin(35) ext;
declare init_processor$new_dbr bit(72) aligned ext;
declare init_processor$first_tra bit(36) aligned ext;
declare init_processor$trouble_tra bit(36) aligned ext;
declare init_processor$startup_tra bit(36) aligned ext;
declare init_processor$lockup_tra bit(36) aligned ext;
declare init_processor$onc_tra bit(36) aligned ext;
declare init_processor$controller_data (0:7) bit(1)
        unaligned ext;
```

The variables declared above are used in the following ways:

1. wait_flag
   is a cell that is used to contain an error code if trouble is experienced in starting up a new processor. If the processor is started successfully, wait_flag will contain zero.

2. new_dbr
   is the descriptor segment base register value for a new processor. It is filled in by reconfiguration software before starting up a new processor. It is loaded by a new processor just before that processor enters appending mode.

3. first_tra
   is an inhibited TRA instruction to be placed in the interrupt vector entry for the processor start interrupt. This instruction is executed when a new processor takes a processor start interrupt.

4. trouble_tra
   is an inhibited TRA instruction to be placed in the fault vector for the trouble fault. This instruction is executed if a new processor takes an unexpected trouble fault.

5. startup_tra
   is an inhibited TRA instruction to be placed in the fault vector for the startup fault. This instruction is executed if a new processor takes an unexpected startup fault.

6. lockup_tra
   is an inhibited TRA instruction to be placed in the fault vector for the lockup fault. This instruction is executed if a new processor takes an unexpected lockup fault.

7. onc_tra
   is an inhibited TRA instruction to be placed in the fault vector for the op-not-complete fault. This instruction is executed if a new processor takes an unexpected op-not-complete fault.

8. controller_data
   is an array of bits, one for each active module port. If a system controller is currently configured and in use, the bit corresponding to its active module port is turned on. Otherwise, it is turned off. This array is used by a new processor to test for the presence of each configured system controller.

SYSTEM CONTROLLER ADDRESSING SEGMENT

The system controller addressing segment (SCAS) is a specialized data base that is used to read and set certain registers in system controllers and their associated store units. The SCAS is essentially a segment composed of a page in each store unit of each configured system controller. It may be up to 32 pages in length. The actual content of the pages is not of importance and in general changes as pages are moved in and out of the particular regions contained in the SCAS.

The SCAS is used to generate the correct final (absolute) address needed by the RSCR and SSCR instructions. (These instructions operate on the system controller or store unit that contains the final absolute address generated by the address preparation logic of the processor.)

Page 0 of the SCAS is located in the first main memory frame of the lower store unit connected to the system controller on port 0 of active modules; page 1 is located in the system controller on port 1; and so forth. Pages 8 through 15 are located in the first memory frame of the upper store units connected to the system controllers on ports 0 through 7. Pages 16 through 24 of the SCAS are used to reference auxiliary lower store units, if configured; pages 25 through 31 are used to reference auxiliary upper store units. Note that there may be "holes" in the SCAS due to certain system controllers or store units not being configured.

MAIN MEMORY AND PAGING DEVICE MAPS

The main memory map consists of entries threaded into a circular list. Entries that correspond to unused frames of main memory are not threaded into the list. The paging device map is analogous to the main memory map. Both are described in detail in Storage System, Order No. AN61.

SECTION IV

DATA BASE INITIALIZATION

This section describes the initialization of the data bases used by the reconfiguration software. Some of these data bases will not be changed after the bootload, others will be changed all the time, and still others will be changed only when reconfiguration is explicitly requested. (System Initialization, Order No. AN70, should be consulted for a much more thorough discussion of system initialization than can be provided here.)

SCS INITIALIZATION

The system communication segment (SCS) described in Section III is initialized primarily by the programs scs_init, scs_and_clock_init, init_scu, scr_util, and scas_init.

The contents of scs$bos_processor_tag and scs$interrupt_controller are set at the very beginning of system initialization. At this time, the clock reading mechanism is initialized. This mechanism consists of a pointer at sys_info$clock_ pointing to scs$port_addressing_word (scs$interrupt_controller). The high-order three bits of this word contain the port number of the bootload system controller. Clock initialization must be performed early in initialization since the clock reading facility is needed by the Multics error message facility.

Elements of scs$controller_data are filled in in stages as various programs learn more about the configuration. The processor switches are read to determine the base and size of each system controller. An RSCR-CFG instruction is then issued to the controller. This CFG data is read into the appropriate element of scs$cfg_data and is interpreted and placed in the appropriate members of the scs$controller_data structure.

The scs$processor_data structure is initialized to mark all processors (including the BOS processor) as offline and halted. The controller port number is filled in from the configuration deck. The interrupt_cpu bit is set on if an interrupt mask on the bootload system controller has been assigned to that processor.

As scs$processor_data is initialized, the interrupt mask pointers and masking instructions for each processor at scs$mask_ptr, scs$set_mask, and scs$read_mask are filled in. At this time, system controller interrupt mask assignments are checked to make sure that they are correct.

As the interrupt handling mechanism is initialized, an unused interrupt cell is selected by the system initialization software and assigned as the interrupt to be used for starting a new processor. This interrupt cell number is saved in scs$processor_start_int_no. A SMIC pattern to generate this interrupt is placed in scs$processor_start_pattern, and an interrupt mask setting to allow the interrupt is saved in scs$processor_start_mask.

SCAS INITIALIZATION

The program "init_scu" is called during initialization and during system controller reconfiguration. It is responsible for filling in the SCAS page table for a system controller. Based on the processor switch settings, init_scu determines the base address of the memory in a controller and sets a page of SCAS to point to that address. init_scu then calls out to read configuration information pertaining to the number and size of the store units connected to the system controller. This data is used by init_scu to set up to three additional pages at the base of each additional store unit. These pages are needed to issue RSCR instructions directed to a particular store unit rather than to the system controller. This function is used primarily by error analysis and logging programs.

SST INITIALIZATION

Before the paging mechanism can be enabled, the main memory map in the SST must be initialized. Since the main memory map cannot be grown, it is required that any system controllers that will ever be configured to the system for a given bootload be specified in the configuration deck and correctly assigned in the configuration switches of the bootload processor. Main memory frames in online system controllers are threaded into the used list. Main memory frames for system controllers not yet placed online are threaded into no list. When a system controller and its main memory are dynamically added, the main memory frames for that controller can then be threaded into the main memory used list. The abs_usable bits for each main memory frame in the bootload system controller are turned on in the main memory map. This action will prevent the removal of any main memory frames contained in the bootload system controller.

The bulk store (paging device) map is also contained in the SST. It is initialized as described on the "page" configuration card.

OTHER DATA BASE INITIALIZATION

The initialization of the PRDS, done mainly by prds_init, tc_data, tc_init and start_cpu, is straightforward and simple. The primary interaction between the traffic controller and reconfiguration consists in the creation, running and deletion of the idle processes.

SECTION V

HARDCORE RECONFIGURATION ENTRIES

Processor, system controller, and main memory reconfiguration is split into two main parts: a user ring command interface and hardcore ring procedures. The user ring command interface, contained in the procedure reconfigure, is responsible for validating reconfiguration command arguments, passing them to the hardcore ring procedures, and analyzing returned error information.

The hardcore portion of processor, system controller, and main memory reconfiguration is located in the procedure reconfig and the many procedures called by it. reconfig is called through the highly privileged hardcore gate "hphcs_".

RECONFIGURATION ENTRIES

Name: hphcs_$add_cpu

This entry is called to add a processor to the system.

Usage

```
declare hphcs_$add_cpu entry (fixed bin(3), (4) bit(36)
                  aligned, fixed bin(35));
call hphcs_$add_cpu (tag, switches, code);
```

1. tag — is the processor tag or processor number of the processor to be added. (Input)

2. switches — are the processor switches which are in error if an attempt was made to add an improperly configured processor. (Output)

3. code — is a reconfiguration error code. The following values are possible:
   1 = no response from processor.
   2 = processor configuration switches set improperly.
   3 = trouble fault attempting to start processor.

4 = startup fault attempting to start processor.
5 = lockup fault attempting to start processor.
6 = processor not in Multics mode.
7 = PTW associative memory and/or SDW associative memory not enabled on processor.
8 = some system controller could not be accessed by processor. (Output)

Name: hphcs_$del_cpu

This entry is called to remove a running processor.

Usage

```
declare hphcs_$del_cpu entry (fixed bin(3), fixed bin(35));
call hphcs_$del_cpu entry (tag, code);
```

1. tag — is as described above. (Input)

2. code — is an error code. The following values are possible:
   1 = processor did not stop.
   2 = only one remaining processor configured. (Output)

Name: hphcs_$add_mem

This entry is called to add a system controller and its associated main memory. If the system controller to be added is interlaced with a controller on an adjacent active module port, both system controllers are added.

Usage

```
declare hphcs_$add_mem entry (fixed bin(3), bit(1) aligned,
                fixed bin(3), fixed bin(35));
call hphcs_$add_mem (tag, interlace, error_tag, code);
```

1. tag — is the tag of the system controller to be added. (Input)

2. interlace — is set to "1"b if the system controller to be added is interlaced with a system controller on an adjacent active module port. In this case, both system controllers are added. (Output)

3. error_tag — is the tag of a processor which has the system controller to be added incorrectly configured. (Output)

4. code — is an error code. The following values are possible:
   1 = actual memory size is smaller than the size found on the configuration card for the system controller.
   2 = two interrupt masks are assigned to one processor on the system controller.
   3 = no mask is assigned to a processor on the system controller.
   4 = a mask is assigned to a system controller port which is not connected to a processor.
   5 = some active module has incorrect configuration switch settings for the system controller.
   6 = some active module is not enabled by the system controller.
   7 = 4MW SCU is not in PROGRAM mode. (Output)

Name: hphcs_$del_mem

This entry is called to remove a system controller and its associated main memory. If the system controller to be removed is interlaced with a controller on an adjacent active module port, both system controllers are removed.

Usage

```
declare hphcs_$del_mem entry (fixed bin(3), bit(1) aligned,
                  fixed bin(35));
call hphcs_$del_mem (tag, interlace, code);
```

1. tag — is as described above. (Input)

2. interlace — is as described above. (Input)

3. code — is an error code which can take on the following value:
   1 = system controller contains abs wired pages in its memory and cannot be removed. (Output)

Name: hphcs_$add_main

The entry is called to add a region of main memory for use by Multics pages.

Usage

```
declare hphcs_$add_main entry (fixed bin(18), fixed bin(18),
                  fixed bin(35));
call hphcs_$add_main (first_frame, n_frames, code);
```

1. first_frame — is the number of the first 1024-word main memory frame to be added. (Input)

2. n_frames is the number of main memory frames to be added. (Input)

3. code is an error code. (Output)

Name: hphcs_$del_main

This entry is called to remove a region of main memory from use by Multics pages.

Usage

```
declare hphcs_$del_main entry (fixed bin(18), fixed bin(18),
                fixed bin(35));
call hphcs_$del_main (first_frame, n_frames, code);
```

1. first_frame is as described above. (Input)

2. n_frames is as described above. (Input)

3. code is an error code. The following values are possible:
   1 = not enough main memory would be left if this request were honored.
   2 = region to be removed contains abs wired pages. (Output)

Name: hphcs_$reconfig_info

This entry returns the information found in scs$controller_data and scs$processor_data. It locks the reconfiguration data base and leaves it locked. If the data base was previously locked, it returns the process group ID of the process which set the lock.

Usage

```
declare hphcs_$reconfig_info entry (ptr, fixed bin(35));
declare 1 rci based (rci_ptr) aligned,
          2 locker_group_id char (32),
          2 controller_data (0: 7) like scs$controller_data,
          2 processor_data (0: 7)  like  scs$processor_data;

call hphcs_$reconfig_info (rci_ptr, code);
```

1. rci_ptr is a pointer to the reconfiguration info structure described above. (Input)

2. code is an error code. (Output)

Name: hphcs_$rc_force_unlock

This entry is called only when the reconfiguration lock has been left locked by a system error or by a call to hphcs_$reconfig_info. It forcibly clears the reconfiguration lock.

Usage

```
declare hphcs_$rc_force_unlock entry;
call hphcs_$rc_force_unlock;
```

ERROR CODES

There are several general error codes which may be returned by any of the reconfiguration entries. These are summarized below:

11 = reconfiguration data base is locked.

12 = device to be added is already online.

13 = device to be added is not in the system configuration.

14 = device to be removed is not online.

15 = requested region of memory is not in the Multics configuration.

SECTION VI

PROCESSOR RECONFIGURATION

This section describes the workings of the processor-adding and processor-deleting functions. Before this can be fully described, however, the mechanism of idle processes must be briefly explained.

IDLE PROCESSES

There is one idle process for each processor on the system. In general, the idle process for a processor is run whenever that processor cannot find another process to run, either because no other process wants service or because all processes that want service are either running on other processors or are waiting for some system event such as a page fault to be satisfied. A processor will never run another processor's idle process.

An idle process is a limited Multics process. It has its own descriptor segment, its own APT entry, but no process stack. The idle process for a processor must be created before that processor is added to the system. (This is not quite true for the bootload CPU which must somehow be bootstrapped into the normal state. See System Initialization, Order No. AN70, for a complete description of this bootstrap mechanism.) Similarly, each processor on the system must have a processor data segment (prds) before it can be run.

An APT entry for each configurable processor (i.e. each processor found in the configuration deck) is reserved during system initialization. When a processor is in use, its idle process APT entry is threaded into a list of idle APT entries. The idle process descriptor segments are apportioned from the single unpaged segment "idle_dsegs" during system initialization. The process data segments (pds) are apportioned from "idle_pdses" in a similar manner. A processor data segment (prds) is created when a processor is added and destroyed when a processor is removed.

ADDING A PROCESSOR

The program start_cpu is called to add a processor to the system. start_cpu is responsible for creating and initializing the idle process for a processor, managing the assignment of system controller interrupt masks, and starting up a processor. First, start_cpu creates the prds for the processor to be added and fills in certain variables in the idle process APT entry. The APT entry is threaded into the idle list at this time, but it is set to a state that will prevent attempts to use the processor.

First, start_cpu ensures that the processor to be started has an interrupt mask assigned to it. If the bootload system controller is a 6000 SC, one interrupt mask must have been assigned to each configurable processor before the system was booted. If the bootload system controller is a 4MW SCU (which has only two interrupt masks) and there are more than two processors in the configuration, the new processor may not have an interrupt mask assigned to it. In this case, another processor must be "persuaded" to give up its interrupt mask and assign it to the new processor. (An SSCR-CFG instruction is issued to the bootload system controller to effect this change.) The mask is cleared and the system controller port to the processor is enabled.

Now, the new processor is capable of being interrupted and can be started up. The contents of init_processor$new_dbr are set to the descriptor segment base register value for the new processor's idle process descriptor segment. The contents of init_processor$wait_flag are set to a value which indicates that the processor to be started has not yet responded to its interrupt. The interrupt vector is patched to direct the processor start interrupt to init_processor$first_steps, and the processor start interrupt cell is set via a SMIC instruction. The interrupt mask for the new processor is set to allow only the processor start interrupt. The new processor should immediately respond to that interrupt. (Note that a connect fault could be used to start a new processor, but it is not used for many reasons. One of these is that the interrupt vector location cannot be moved by changing processor switches.)

After setting the interrupt cell, start_cpu loops for several milliseconds until init_processor$wait_flag changes. If the new processor started up successfully, the value of wait_flag will be zero. If it failed to respond to the interrupt, start_cpu will time out with the no response error code already in init_processor$wait_flag. If another error condition was detected, wait_flag will contain an error code indicating why the processor could not be added. This error code is returned to the caller of start_cpu.

The program init_processor (see System Initialization, Order No. AN70) is invoked to start the idle processes of all processors on the system, including the bootload processor during system initialization. This program consists of two distinct sections: the initialization code to start a processor and the idle process loop for all processors. The initialization code, in turn, consists of two parts. The first part is entered when the processor start interrupt is received. It runs in absolute mode and checks that all processor switches are set correctly. If the first part of processor initialization is successfully completed, the processor's DBR is loaded and an indirect transfer is executed to place the processor in appending mode and enter the second part of processor initialization.

The second part of the initialization code further fills in the idle process APT entry so that the processor can now be assigned to do useful work. It then issues a connect to itself to preempt the idle process and look for useful work for the processor. The wait_flag is cleared, indicating to start_cpu that the processor was successfully started. (The second part of processor initialization is called directly by start_cpu when initializing the idle process of the bootload processor.)

The idle loop is essentially an uninhibited DIS instruction with a transfer back to the DIS. A processor which is idle will always be at the DIS instruction. If work exists for the processor, the idle process will be preempted by sending a connect to the processor.

Many mechanisms are included in start_cpu and init_processor to allow recovery from operator and hardware errors when attempting to add a processor. Unexpected startup faults, trouble faults, and lockup faults (which sometimes occur for unexplained reasons when adding a processor) are directed to a special place in init_processor during the time that a processor is being added. When init_processor catches such an unexpected fault, it sets a special error code in wait_flag. An SCU instruction is not placed in the fault vector for these faults since a processor may experience difficulties in executing an SCU at this time.

All processor configuration switches are checked for correctness by init_processor. If one or more switches are incorrect, an appropriate error code is set, and information indicating which switches are in error is returned by start_cpu. If the processor is inadvertently left in GCOS or ABS mode, init_processor will detect the error and an appropriate error code will be returned. If the processor is left in STEP, it will not respond to the processor start interrupt. After several milliseconds, start_cpu will time out and return the no response error code initially placed in init_processor$wait_flag. The init_processor program also checks to make sure that the new processor can access each configured system controller. It does this by issuing a Read Calendar Clock (RCCL) instruction to each configured controller. If, for some reason, a controller cannot

be accessed, init_processor intercepts the resultant op-not-complete fault and returns an appropriate error code. The PTW and the SDW associative memories must both be enabled in a processor to be added; init_processor checks to make sure that this is so.

One error condition especially difficult to detect is the incorrect setting of the memory assignment switches on a processor. Such an error may cause the processor to believe that its fault and interrupt vectors are located in a system controller other than the bootload system controller. Recovery from such an error is accomplished by replacing the contents of the first two memory frames of each system controller with a special fault and interrupt vector image contained in the program fv_iv_template. Upon intercepting any fault or interrupt, fv_iv_template will direct the processor to read its switches and store them in a reserved place. It will then stop the processor at an inhibited DIS. If start_cpu times out, it will search the reserved place in each copy of fv_iv_template to see if processor switch data has been stored there. If data has been stored, start_cpu will return the error code indicating that a configuration switch error has occurred along with data indicating which switches are in error.

NOTE: If more than one system controller is incorrectly assigned as the bootload system controller on the configuration panel of a processor to be added, the recovery method described above will probably fail. This is one of the few error conditions that cannot be handled by the reconfiguration software.

Since no operator intervention or interaction is normally required to add a processor to the system, it is possible to bootload a system with several processors in the configuration. Additional processors will automatically be started at the completion of system initialization.

REMOVING A PROCESSOR

The program stop_cpu is called to remove a processor from the system. It first checks to see if the processor being removed is the BOS (or bootload) processor. If it is, a new BOS processor is assigned. If the bootload system controller is a 4MW SCU, the processor relinquishes its interrupt mask. If any other processors are currently running without an assigned interrupt mask, the freed interrupt mask is given to one such processor. Now the processor is ready to be stopped. The delete_cpu bit in scs$processor_data for the processor is set and a preempt connect is sent to the processor.

When the dying processor receives the preemption, it enters special code in the traffic controller. The idle process APT entry is updated in order to prevent further use of the processor. The halted_cpu bit is turned on in scs$processor_data for the processor, and the processor is stopped at an inhibited DIS instruction.

When stop_cpu detects the halted_cpu bit, it proceeds with destroying the processor's prds and removing its idle APT entry from the thread of idle APT entries.

During system shutdown, stop_cpu is called automatically to remove all processors other than the BOS processor. It is therefore not necessary to manually remove processors before shutting down Multics.

SECTION VII

MEMORY RECONFIGURATION

This section describes the mechanisms used to dynamically reconfigure main memory (core or MOS memory). Two subsections describe system controller reconfiguration and another two describe main memory frame reconfiguration within a controller.

ADDING MAIN MEMORY

At system initialization time, the data bases scs$controller_data and the main memory map in the SST are initialized. These are initialized from the configuration deck (and active register values); since the main memory map cannot easily be grown, it is required that any system controllers that will ever be configured to the system for a given bootload must be specified in the configuration deck. This is done by using an ON or OFF field of the MEM configuration cards. All system controllers actually configured and to be used at bootload time are indicated as being ON. Other system controllers are OFF.

When the main memory map is initially set up, only map entries for main memory frames which are in configured system controllers are threaded into the used list. Map entries for main memory frames in system controllers that are not yet configured are left alone and threaded into no list. To add a main memory frame to the system, all that need be done is to thread the map entry for the frame into the main memory used list. This is exactly what is done after an addmain request is given.

A frame of main memory is added by calling freecore, a primitive in Multics page control. Before threading the map entry for a frame into the used list, freecore touches all words in the frame. It then notes if any parity errors occurred. A main memory frame containing one or more words with parity errors will not be added to the used list.

ADDING A SYSTEM CONTROLLER

To add a system controller to the system, reconfig checks consistency of all arguments and calls add_scu to actually add the system controller. The program add_scu fills in the PTW for the page of the SCAS used to reference the system controller. It then forces the executing process to run on all processors and attempt to reference the system controller. If the controller is not properly configured at a processor configuration panel, or if a processor is not enabled at the system controller, a fault will occur. Any fault will be caught by add_scu and reflected back to the caller as an error. Note that no mechanism currently exists to check the configuration switches on the IOM and the bulk store.

If the system controller can be successfully accessed by all processors, add_scu returns to reconfig which can then make calls to freecore to thread the main memory map entries for each main memory frame in the controller into the used list. If an attempt is made to add a system controller which is interlaced with a another controller on an adjacent active module port, both controllers will be added automatically. Two calls are made to add_scu before the main memory shared by the two controllers is added.

REMOVING A SYSTEM CONTROLLER

Two major problems are involved in removing a system controller from the system. First, a mechanism must be provided to remove all references to any pages in the system controller by processors. This mechanism is described in the following subsection.

The second major problem in removing main memory, that of preventing other active modules from referencing the memory is solved before it even becomes a problem. This is accomplished via the abs_wiring technique, which requires that any pages which are referenceable via nonprocessor active modules (e.g., the IOM) cannot reside in a deconfigurable system controller. In order to do this, certain system controllers are set up as "abs_usable" and hence nondeconfigurable. For most configurations, the bootload system controller alone is abs_usable, but the system dynamically chooses other controllers as necessary if there is not enough abs_wireable memory in the bootload system controller.

Therefore, any program that uses a page for I/O (that is not permanently wired) must call a special program to have the page wired down. This program is "pc_contig". (See Storage System, Order No. AN61, for a complete description of this mechanism.) Thus, the dynamic reconfiguration software need not be concerned about pages wired for I/O activity.

REMOVING MAIN MEMORY

The problem in removing main memory mentioned above, that of preventing processors from referencing the memory, is not hard to solve since all processor references to the memory are made indirectly through PTWs over which the system software has control. (It is not possible to remove memory which contains permanently wired code or data.) It is thus necessary only to remove access in PTWs or to copy pages into main memory which is not being removed. This in fact is just what is done. There are three cases to consider:

1. Main memory frames that contain wired pages.

2. Main memory frames that contain pages that are not wired but are modified.

3. Other main memory frames.

Pages which are temp-wired must remain in main memory but need not remain in the same frame of main memory. Such pages are copied from the region of memory being removed to a region of memory remaining. After the copy is complete, the PTW is changed to point to the new copy, and all processors are forced to clear their associative memories so that they will refetch the PTW with the new address (making all subsequent references to the new copy). If the page is modified while the copy is being performed, all processors are stopped (forced to loop lock via a connect fault), and the copy is made while no other processor can modify the data. The entire mechanism to move a wired page is implemented in the subroutine evict_page.

Pages that are modified are simply written out to secondary storage and evicted when the I/O completes. This process continues until a page does not get modified while the I/O is going on, in which case the frame of memory can be claimed.

Pages that are neither wired nor modified are evicted immediately (the PTW is set to fault) unless is I/O is in progress, in which case the I/O is waited for and the page is evicted on the next pass.

The program responsible for performing all this work is pc_abs$remove_core. It loops through all frames to be removed until a pass is completed that leaves no frames unclaimed. Note that a request to remove a frame that is already removed is not considered to be an error. Such a condition might well occur when an operator makes a request to remove an entire controller after having removed several main memory frames in that controller.

AUTOMATIC MEMORY REMOVAL

(To be supplied.)

SECTION VIII

BULK STORE RECONFIGURATION

The bulk store reconfiguration mechanism provides a means for the dynamic removal and addition of selected records of the bulk store (paging device). The software is set up so that if all records on a given bulk store are removed from active use by the system, the controller can safely be deconfigured for offline test or for use by another local system. No switches need be changed during this reconfiguration mechanism. (It is, of course, necessary that any parts of the bulk store which are to be used by a system be configured to that system; the bulk store reconfiguration software assumes that necessary configuration switches have been set.)

BULK STORE INITIALIZATION

During system initialization, the "page" configuration card is read to determine which paging device records are initially to be used by the system. The label of the root physical volume (RPV) is inspected by system initialization to see if the system had previously crashed without successfully flushing the contents of the paging device. If this is the case, the system is said to have an unflushed paging device, and use of the paging device by the system (other than by the physical volume salvager) is inhibited until all relevant records have been flushed. Refer to System Initialization, Order No. AN70 and Storage System, Order No. AN61 for greater detail on this operation.

If the previous bootload had shut down or flushed the paging device successfully, the pages specified on the "page" configuration card are threaded into the paging device used list and marked as free. All of these records will remain in this used list until reconfiguration time, unless removed for any of the reasons listed below:

1. The record is momentarily removed as part of a rethreading operation.

2. The record is removed because a read/write sequence is in progress for the given page.

3. The record is automatically removed because of a fatal I/O error.

The paging device map (like the main memory map) can be searched either by following the used list thread or by indexing into the map with a given record number. The latter method is used for bulk store reconfiguration under operator control.

BULK STORE RECONFIGURATION ENTRIES

The main supervisor program that controls bulk store record reconfiguration is delete_pd_records. This program, callable through the hphcs_ gate, has two entries related to bulk store reconfiguration.

Name: hphcs_$delete_pd_records

Usage

```
declare hphcs_$delete_pd_records entry (fixed bin, fixed
                bin, fixed bin(35));
call hphcs_$delete_pd_records (first, count, code);
```

1. first — is the record number of the first of count contiguous records to be removed from active use by the system. (Input)

2. count — is the number of records being deconfigured. (Input)

3. code — indicates, if nonzero, that the input parameters were inconsistent with the current configuration. (Output)

Name: hphcs_$add_pd_records

Usage

```
declare hphcs_$add_pd_records entry (fixed bin, fixed bin,
                fixed bin(35));
call hphcs_$add_pd_records (first, count, code);
```

1. - 3. are analogous to above.

A request to delete a record that is already deconfigured is not considered fatal. In fact, it is convenient to be able to delete an entire core storage module (CSM) after several records within it have been automatically deleted by page control. The paging device map always resides on the first few records of the paging device region which is potentially usable for a given

bootload. It is again nonfatal to request that these records be deleted. However, they will not be deleted because the current implementation does not provide for moving the paging device map copied onto the paging device. In particular, if the first CSM is to be deleted (for offline work) the entire bulk store must be disabled.

ADDING BULK STORE RECORDS

To add a region of the bulk store to the current configuration, the operator must specify which regions of the bulk store should be added. As mentioned earlier, all configuration switches must have previously been set correctly before the bulk store add request is given; this includes the various switches on the bulk store controller as well as all the port enable switches on all system controllers. (Recall that normal operation is to have the port enable switches under program control.) Since the bulk store controller is not the target of the operator requested reconfiguration, the port enable registers in the system controllers are not changed even if the entire set of bulk store records are deconfigured.

The actual mechanism to add bulk store records to the system is quite similar to the main memory addition mechanism. It is necessary that all of the paging device map that will ever be needed for a bootload be allocated at system initialization time. Those records of the bulk store that are not initially part of the system do not have their PDMEs threaded into the paging device used list. The "addpage" request issued by the operator merely threads the PDMEs for records being added into the paging device used list and updates the two system-wide variables in the SST, pd_free and pd_using. The variable pd_free reflects the number of records actively configured and free for use. The variable pd_using indicates the number of records actively configured. Both of the variables are updated by the internal procedure set_pd_free_and_using (under control of the global paging lock) in the main bulk store reconfiguration program delete_pd_records. Note that when pd_using reaches zero, (i.e., there are no records actively being used), the automatic update of the paging device map is disabled, making it possible to physically deconfigure the bulk store controller. The variable pd_using also controls whether or not page control will attempt to allocate pages on the bulk store.

The internal procedure build_page_card of delete_pd_records updates the page configuration card (if possible) to reflect the current bulk store configuration for both adding and deleting bulk store records.

REMOVING BULK STORE RECORDS

Removing bulk store records is analogous to removing main memory frames. The entire mechanism is controlled by the program delete_pd_records in the hardcore ring. This program first checks its parameters for consistency and then loops through the specified region of the paging device map (indexing by paging device record number), cleaning out pages as it goes. The entire process is under control of the global paging lock; since the various control bits (such as the modified bit) of the PDME are simulated and under control of the same lock, these bits will not change as long as the lock remains set.

There are five cases of interest. These are:

1. The record is not used.

2. The page for a given record is in main memory.

3. The page is not in main memory but has been modified since last written to secondary storage.

4. The page is not in main memory and has not been modified.

5. A read/write sequence is in progress for the given page.

If the record is not used, it is merely removed by threading its PDME out of the paging device used list.

If the page is in main memory, the core map entry is updated to include the secondary storage device address rather than the paging device address; if the modified bit is in the PDME, the modified bit is set ON in the corresponding PTW. This can cause a slight anomaly in the value of date-time-modified for the segment owning the page.

If the page is not in main memory but has been modified since last being written to secondary storage, a read/write sequence is initiated for the page. In addition, a flag is set in the PDME so that when the read/write sequence completes, the paging device record will be marked as being deconfigured (i.e. the PDME will not be threaded into the paging device used list). The flag used to indicate this is pdme.removing.

If the page is not in main memory, and has not been modified, the secondary storage address from the PDME replaces the paging device address in the PTW.

If a read/write sequence is in progress for a page, the pdme.removing flag is set ON so that the record will not be threaded into the paging device used list when the RWS completes. Any pages that have RWS's in progress are remembered; when all PDMEs have been scanned, the last RWS noticed is waited for. The scan is then started again from the beginning until a pass is completed where no RWS's are seen.

AUTOMATIC PAGING DEVICE RECORD REMOVAL

When page control encounters a fatal I/O error from the paging device, the deconfiguration of that paging device record occurs automatically. This action, performed at page control "done" time, consists of typing out a message on the operator's console and placing the paging device map entry in the deconfigured state. This type of dynamic deconfiguration is not reflected on the page configuration card until the next explicit paging device reconfiguration is performed by the operator.

If the error occurred on a normal page write, the concerned page of the affected segment is emigrated from the paging device by placing the secondary storage address from the PDME into the relevant main memory map entry (CME). On a page read error from the paging device, the secondary storage address from the CME replaces that in the associated PTW. In either case, the migration of the page to the paging device is undone. In the case of a read error, the standard address management policy causes the old copy of the data on secondary storage to replace the bad paging device copy, if it had ever been written to secondary storage. If it had never been so written, the data is replaced with zeroes.

On a read/write sequence error (reading from the paging device), again the paging device record is deconfigured automatically. The live/nulled status of the associated secondary storage address in the PDME is inspected to see if the data on secondary storage indeed belongs to this segment. If the address is nulled (data has never been written to secondary storage), a special null address replaces the secondary storage address in the PDME. This causes a page of zeroes to replace the contents of the page at any future page fault time, and a new secondary storage address to be assigned.

SECTION IX

THE COMMAND INTERFACE

The reconfiguration of main memory, processors, and bulk store is under operator control either from an "initializer" terminal or from a privileged logged-in user. The initializer commands are as follows:

```
addcpu
delcpu
addmem
delmem
addmain
delmain
addpage
delpage
```

These are described fully in the Multics Operator's Handbook, Order No. AM81.

The normal user commands are as follows:

```
addcpu
delcpu
addmem
delmem
addmain
delmain
addpag
delpag
```

The following special command is also provided:

```
reconfigure$force_unlock
```

Note that there is no initializer command to unlock the reconfiguration lock.

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 68) MULTICS RECONFIGURATION PROGRAM LOGIC MANUAL

ORDER NO. AN71, REV. 1

DATED APRIL 1977

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

18106, 3C577, Printed in U.S.A.

AN71, Rev. 1