To: Distribution

From: Keith Loepere

Date: March 15, 1984

Subject: Bootload Multics Phase 1

Bootload Multics (also known as the bootload command environment (bce)) is a new phase of Multics initialization. Its purpose in life is to allow Multics to run without BOS which in turn allows Multics to run on hardware on which BOS cannot run. Bootload Multics is being provided in two phases. The first phase, which this MTB describes, provides enough functionality so that Multics may be run, albeit possibly not always as easily as desired, without BOS. In phase 1, Multics will be normally booted from BOS, however, and will rely on BOS for certain functions, when it is desired. Phase 2 is an ongoing process in which all of BOS' functionality is added to bootload Multics.

This first phase is being installed at this time for two reasons. First of all, it contains the most important and crucial aspects of bootload Multics, which will need considerable exposure. Also, it is felt that certain enhancements provided by bootload Multics would be desirable if installed now. Most notable of these is the ability to set "probe" breakpoints in hardcore segments and the ability to perform a "warm" boot from disk.

This MTB describes the structure of bootload Multics in general and its use and operation in particular. Detailed descriptions of the internal operation of bootload Multics do not appear here. For this information, the reader should refer to MTB-652, the proposed new Initialization SDN.

Comments on this MTB should be sent to the author:

Keith Loepere (Loepere.Multics)

or via the bootload_multics forum.

---

TABLE OF CONTENTS

This MTB is split into five main sections. The first four are the basis of the MTB proper as regards to describing bootload Multics. The fifth section contains the documentation that goes with bootload Multics; as such it also contains the detailed description of the operation of bootload Multics. The page numbers within this fifth section are made to align, section number-wise, with the manual to which they belong. As such, these page numbers jump in no apparent order.



## I. INTRODUCTION

In the current scheme of things, Multics is booted from BOS. BOS is a very crusty and cryptic set of programs that runs outside of Multics itself. It is entirely written in an obscure dialect of alm. As such, it is difficult to maintain and difficult to modify. Each hardware change that comes along requires modifications to BOS (as well as Multics). To remove future needs to modify BOS for such reasons, it is desired to remove BOS.

To understand what functions are required of a BOS replacement, it is best to consider, as a start, the normal sequence of events that BOS puts into motion to boot Multics.

First of all, BOS is capable of being booted directly from its tape via operator's console (or iom) switches. It places firmware into the various mpcs so that disk and tape i/o can proceed. It generates a config deck describing all hardware units at the site. It sets system controller clocks. Once in memory, it can start the boot of Multics from the Multics System Tape (MST). Multics takes over from there. When Multics shuts down, BOS regains control, ready to re-boot Multics. If Multics crashes, BOS, by virtue of being outside of Multics, can run to take a dump, for later analysis. It also forces Multics to perform an emergency shutdown.

To replace BOS, then, at a minimal level, requires being able to boot Multics directly from a MST on a completely cold machine, to load firmware, to set clocks and to provide a config deck before Multics, as it currently stands, can boot service. Also, a safe platform that Multics can crash/shutdown to must be provided from which dumps and emergency shutdown can be initiated, as well as re-booting.

## II. SIGNIFICANT CHANGES AND FEATURES

The most obvious operator visible change resulting from this installation will be the installation of a new command level (bce). This command level provides more power than the equivalent BOS command level; in particular, the power of Multics lies behind it. Once bce is booted, either from BOS or from the "switches", most operator activities previously performed at BOS will be performed here.

A long desired feature for Multics provided by bce is the ability to perform a "warm" boot from disk. (Sorry, no "cool" or "cold" boots.) That is, when at bce, one can boot Multics purely from disk without a MST (Multics System Tape) being mounted on the proper tape drive. Not only is this more convenient, but it makes it easier to set up for automatic operation.

Changes to the config deck are now made with the config deck editor (although BOS' "config" command can still be used). This new editor uses qedx_ for text editing operations, providing more convenient changing. Also, the config deck editor understands various labels for fields on config cards; it is no longer necessary to remember the order of fields on these cards.

The bce equivalent of the BOS "runcom" facility will be version 1 exec_coms, a much more powerful and sensible facility.

BOS' "patch" and "dump" facilities are being merged into the bce "probe" facility. This facility provide more power in performing patch/dump operations than BOS. First of all, it allows dumping of memory and disk in various formats, including instruction. It can display machine conditions in interpreted form. It allows various formats for data that is to be patched into memory or disk. It can also trace stack frames.

An important feature of the bce "probe" facility is its ability to set breakpoints in Multics and bce itself. The normal probe requests of "before", "reset", "status" and "continue" apply to them. It is hoped that this facility can reduce future Multics hardcore and bce debugging times.

bce functions can be aborted in a cleaner manner than in BOS. Within BOS, hitting REQUEST aborts a function (even if hit accidentally). bce allows functions to be aborted to various extents; also, one can cancel an accidental hit of REQUEST.

The bce equivalent of the BOS firmware loader is more intelligent also. Where as BOS requires one to specify the location of an mpc and what firmware images to load into it, bce can determine all of this information purely from an operator supplied mpc name (and the config deck).

bce's operations on bce files are more powerful and more Multics-like than the equivalent BOS functions.

Provided in this installation are various paramaterizations for Orion support. Admitedly, this is no longer meaningful but the placement of the paramaterizations will make future hardware support easier.

## III. IMPACT

The various paragraphs in this section discuss the impacts resulting from this installation.

The foremost impact is the appearance of the new command level. The operator must be trained to understand this and the new commands.

Not all features of BOS are present in this installation. Although it is claimed that one could survive without these missing features, it will probably be desirable to utilize them from time to time. Thus, one must remember which features one must return to BOS for; one must also know how to return to BOS and then back to bce.

Certain BOS functions become broken by this installation. First of all, performing a dump, esd, etc. type operation from BOS that examines the Multics image will not work since they will examine the bce image. Also, they do not work generally because of significant changes to segment generation (see below).

This installation significantly modifies the method of allocation and creation of hardcore segments. All hardcore segments, with the exception of fault_vector, iom_mailbox, dn355_mailbox, isolts_abs_seg and the abs_seg's used by page control for examining memory frames, will now be created as paged segments. Thus, they all take up an integral number of pages. This causes more memory to be used for wired supervisor segments than was true previously. Certain packaging of hardcore entities was performed to regain some of this space. This making paged of segments was added for several reasons. First of all, it is required for future processor support. Also, it is a part of hardcore breakpoint support. The page tables for these segments are kept in segments maintained for the purpose. This making paged breaks various BOS analysis tools; in particular, BOS fdump ceases to function. Also, the methods used by the Multics dump analyzers to determine absolute memory addresses tends to fail.

bce requires two new disk partitions on the rpv. One is the "file" partition, used to hold bce exec_coms and config deck sources. It is 255 pages long. The second is the "bce" partition, used to hold the saved Multics image and to restore

bce, bce paged work segments, and the contents of the MST so that "warm" boots may be performed. This partition is 2200 records long. Thus, 2455 rpv records are used up by bce. Since rebuilding the rpv to create these partitions can be a problem, bce includes a program that will dynamically rearrange the pages on the rpv to lay down these partitions (assuming that there are enough free records to do so).

Two new collections are being added to the MST. Collection 1.2 contains config decks and exec_coms (ascii files) that are auto-loaded into the bce file system. Thus, a site may generate a MST with their desired files on it and also generate a tape that another site may cold boot from, without the other site having to enter a config deck. Collection 1.5 contains paged bce programs, as well as certain firmware objects. check_mst is modified to understand the new collections.

## IV. DETAILED PROPOSAL

This section describes the set of modules to be installed and their purpose. It is divided into several sub-sections pertaining to areas of changes.

### Multi-processor Paramaterization

The following modules were modified/added to support different processor types.

| | |
|---|---|
| adp_scu.incl.alm | adp_scu.incl.pl1 |
| bootload_0.alm | bootload_console.alm |
| bootload_cpu_macros.incl.alm | bootload_error.alm |
| bootload_faults.alm | bootload_flagbox.alm |
| bootload_formline.alm | bootload_info.cds |
| bootload_io.alm | bootload_linker.alm |
| bootload_loader.alm | bootload_slt_manager.alm |
| bootload_tape_fw.alm | bootload_tape_label.alm |
| dbr_info.incl.pl1 | dbr_util_.pl1 |
| ptw_info.incl.pl1 | ptw_util_.pl1 |

### Paging Hardcore Segments

The making paged of almost all hardcore segments runs through many initialization programs. First of all, those segments whose sdws were built by template_slt_.alm had to have page tables also built by template_slt_. Thus, template_slt_'s macros for generating slte/sdw entries have become considerably more involved to also generate page tables. These macros

generate the page tables (and sdw's) for all processor types. Also, bootload_abs_mode.alm, which prepares the way to leave abs mode, must copy and activate these page tables. bootload_loader.alm no longer establishes unpaged sdw's for the segments it loads. Instead, it uses the new entrypoint make_core_ptw in bootload_dseg.alm to generate ptws. These programs use the new include files unpaged_page_tables.incl.(alm pl1) to describe the page tables for these formally unpaged segments. The page tables are placed into one of two places. For permanent hardcore segments (those that previously stayed unpaged), the page tables are placed into the segment unpaged_page_tables. Those segments to be made paged have their page tables in int_unpaged_page_tables (initialization and temp). The page tables for these two segments are within themselves, respectively. With these new segments, coll0_segnos.incl.alm had to be modified. These changes take care of collection 0 generated segments.

Collection 1 generated segments are made through get_main. (Segments created by make_sdw are automatically paged.) get_main.pl1 was modified to also generate page tables in the manner of bootload_loader. It was also provided the entrypoint, given_address, for use by init_sst.pl1 (which allocates the sst and core map) and others to generate the page tables for the segments they hand allocate.

scs_and_clock_init$early.pl1 was also changed to make the early form of scas paged.

move_non_perm_wired_segs.pl1 had to be modified to know how to move such paged segments.

Many other programs had to have their notion of "unpaged" (i.e., not paged under the auspices of page control) fixed. These programs now ask whether the address field in the sdw is within the bounds of the sst (implies page control paged). Also, some programs were just very bad at keeping straight the difference between abs-segs and other segs and would trip over some of these new segments. These included:

collect_free_core.pl1
delete_segs.pl1
emergency_shutdown.alm
get_ptrs_.alm
init_sst_name_seg.pl1
make_segs_paged.pl1
page_fault.alm
privileged_mode_ut.alm
syserr.alm
wire_proc.pl1

idle_dsegs and idle_pdses were paged. The allocation algorithm used by tc_init.pl1 (calling get_main) was optimized to save space. start_cpu.pl1 understands the new layout.

To keep tabs on how much page table space is being used, announce_chwm.pl1 (core high water mark) was modified to print the usage of the two unpaged page table segments.

The header of unpaged_page_tables contains the absolute addresses of the start and end of unpaged_page_tables, int_unpaged_page_tables and sst_seg. This information is here so that dump analyzers can determine in which of these segments the page table for a given segment lies given the absolute address found in the segment's sdw.

Finally, for other bootload Multics flagbox changes and so that the flagbox (as a segment) could start at a page boundary, a new flagbox.incl.(pl1 alm) include file was made to avoid conflict with the old (fgbx.incl.(pl1 alm)) bos versions. The flagbox was also moved into the bce toehold. This means that the bos_toehold flagbox has no use in BOS/bce communication. Modules that needed recompilation are:

accept_rpv.pl1
azm_why_.pl1
bootload_error.alm
bootload_flagbox.alm
flagbox_mgr.pl1
init_sst.pl1
ol_dump_why_.pl1
setup_dump_segments.pl1
shutdown_file_system.pl1
start_cpu.pl1
stop_cpu.pl1
structure_library_2_.cds
sys_trouble.alm
syserr_real.pl1

## Multiple Initialization Passes

Bootload Multics, as installed in MR10.2, was capable of making an "early" initialization pass for booting without BOS. For the real bootload Multics installation, bce needs to make many possible passes for various purposes. To clean this up, real_initializer.pl1.pmac was modified to make collection 1 an internal subroutine, keying off the new variable sys_info$collection_1_phase (whose values are described in collection_1_phases.incl.pl1). The purpose of the various passes is described below. Basically, the difference between passes is the extent of resources used by them.

## INITIALIZATION PASSES

The first pass, made only when bce is not booted from BOS, is called the "early" pass. It is a special pass in that it must generate a config deck, or at least reach a point where an initial config deck can be entered ("cold" boot). Normally this pass determines certain attributes of the site configuration from polls made during collection 0 and from operator queries (on the order of "Where is rpv?"). Once rpv is found, bce reads in the config deck and is ready for other initialization passes. It comes to an "early" bce command level so that the config deck can be straightened out.

The second pass, which to bce's point of view is the normal pass, is called the "boot" pass. This pass uses the config deck to describe available peripherals. It, however, limits itself to 512k and one processor. When finished initializing tables for such hardware, it comes to the "boot" command level, from which most bce activities occur, such as booting Multics service.

The third pass, known as the "service" pass, uses all peripherals and all memory. It starts by re-arranging itself to utilize all available memory.

Other passes exist. If the "boot" pass fails, a "re_early" pass is run. This pass is really identical to a "boot" pass, but it is run with the safe config deck (that determined during initial hardware polling) to re-establish an "early" command level.

If the "service" pass fails, a "bce_crash" pass is made, using the saved config deck that was capable of running the original "boot" pass. This new pass (and resultant "bce_crash" command level) is provided under the assumption that a bce utility died or that the operator screwed up the config deck at the "boot" command level.

A "crash" pass exists, similar to the "boot" pass, to set up to examine a crashed Multics image. It differs from the "boot" pass only in its verbosity and the actions that occur if it fails.

Finally, the "shut" pass is run when Multics shuts down, as opposed to crashing. It is similar to the "boot" pass except for skipping the checking/loading of disk mpcs.

## Addition of bce

Aside from creating multiple initialization passes, various other levels of support were added to provide the bootload command environment.

The disk space to run bce is obtained through three programs. First, fill_vol_extents_.pl1 provides the necessary partitions in its default list when "cold" booting the rpv. init_root_vols.pl1 checks for the presence of these partitions. When not found, it calls the new routine create_rpv_partition.pl1 to generate them. This program determines the desired placement of the partitions on the rpv and then runs through all vtoces on the drive, finding pages within this region, moving them elsewhere, updating the file map to show it and updating the vtoce (done in the correct order, of course).

The first set of disk i/o's are performed only through the bootload disk mpc found by find_rpv_subsystem. Once the config deck is read from the rpv, though, other disk mpcs may have been discovered; these need firmware loaded into them. The routine load_disk_mpcs looks for disk mpcs that do not seem to be running. It lists these mpcs and gives the operator a chance to load them. The loading is done by bce_fwload.

Mapping bce's pagible temp segments onto the "bce" partition is done by establish_temp_segs.pl1. Mapping bootload_file_partition (bce file system) unto the "file" partition is done by find_file_partition.pl1. These use the new routine, map_onto_disk.pl1, to find and build an aste/page table for the disk space.

The bce command level is implemented via several routines, whose functions are pretty much obvious from their names. These are:

bce_execute_command_.pl1
: used by command_processor_$subsys_execute_line to find and invoke a bce command.

bce_get_to_command_level.pl1
: calls bce_listen_ appropriately. It contains the commands that leave command level, namely boot and reinitialize. These commands leave command level through non-local goto's.

bce_list_requests_.pl1
: implements the list_requests command via bce_map_over_requests_.

bce_listen_.pl1
: bce standard listener to read and execute command lines.

bce_map_over_requests_.pl1
runs a function over the entries in a request table (in this case, bce_request_table_). Used by bce_execute_command_ and bce_list_requests_.

bce_ready.pl1
prints bce ready messages.

bce_request_table_.alm
ssu_ style request table, parsed by bce_map_over_requests_. It contains all commands for all bce command levels. The last unused flags in the request data structure are used to record which levels allow a given command.

One of the most important requirements of bce is to be able to regain control when Multics crashes or shuts down in a safe way. This transfer is performed by toehold.alm, imbedded at the beginning of collection 0 at a known memory address. It contains the logic to save 512k of memory to disk, saving the machine state, read in bce and invoke bce. It also can swap Multics back in again and restart it. The toehold keys off two pieces of information; the first is the iom, channel and drive number of the rpv, maintained by reconfiguration; the second is a set of dcw lists to use to read/write memory. These lists are generated by init_toehold.pl1. init_toehold also saves a good copy of bce to read in upon a crash. The saving of bce's machine conditions so the toehold may start bce is done by save_handler_mc.alm.

The bce file system is provided by the routine bootload_fs_.pl1. It manages a primitive file system on the segment bootload_file_partition, mapped on top of the "file" partition on rpv. It maintains the directory entries contained therein.

## Warm Boot from Disk

Warm boot from disk consists of two parts. The first is to save what is needed for a boot of Multics service (collections 2 and 3) away on disk. The second part is to boot by reading these from disk.

The second part is relatively easy. segment_loader.pl1 and load_system.pl1 were simply modified to call disk_reader instead of tape_reader. disk_reader.pl1 is a program that slides an abs-seg down the MST area of the "bce" partition to read records, instead of reading tape records as did tape_reader.

The first part is embodied in load_mst.pl1. load_mst is the master reader of the MST from after collection 1 on. After reading collections 1.2 and 1.5 into their proper places,

load_mst saves the rest of the tape (collections 2 and 3) on disk pretty much as is for disk_reader to find.

Since tape reading is not needed past collection 1, tape_reader.pl1 and boot_tape_io.pl1 are now in collection 1 temp segs. Also, boot_tape_io no longer needs to wire physical_record_buffer.

## Multics support added to bce

The addition of Multics features to bce was done in one of several ways.

First of all, some modules already on the MST were simply moved down into collection 1. Some of these modules were modified on the way to do the right thing in both (Multics and bce) environments. These modules are:

active_fnc_err_.alm
: added pass through to bce_error.

com_err_.pl1
: provided pass through to bce_error$com_err.

cv_dec_.pl1
: added cv_binary_ and cv_binary_check_ entrypoints.

date_time_.pl1
decode_descriptor_.pl1
filemap_checksum_.alm

get_temp_segments_.pl1
: made to use bootload_temp_N during bce usage for temp segments.

match_star_name_.pl1
ondata_.alm
requote_string_.pl1
stack_header_util_.alm
sub_err_.pl1

The second category are routines moved from other libraries for bce usage. These modules will be deleted from their original libraries. These modules are:

convert_date_to_binary_.rd
cp_data_.cds
get_equal_name_.pl1
move_r_or_t_.alm
numeric_to_ascii_.pl1
pl1_decat_char_.pl1

The next group are modules copied from other libraries. They will be present on the MST and in their respective libraries. These are modules paged within bce and therefore not visible in >sl1.

```
check_entryname_.pl1
command_processor_.pl1
equal.pl1
get_addr_.pl1
op_mnemonic_.cds
plus.pl1
search_file_.pl1
substr.pl1
```

The next group of modules are also copied from other libraries. In these cases, though, significant modifications were necessary. Depending on the type and amount of modifications, one of two courses was followed. In the first course, the module was converted from foo.pl1 to foo.pl1.pmac. These modules generate from the single pmac source two different versions, one for bce and one for Multics. They must be pmac'ed with the control argument

```
-pm target """bce"""
```
or
```
-pm target """multics"""
```

They will not pmac if neither option is supplied. These modules are:

```
edx_util_.pl1.pmac
qedx_.pl1.pmac
```

The other group are those for which the changes made sense to produce a different module which was largely stolen from an original.

bce_abs_io_data.incl.pl1
: removes unneeded variables, moves work area.

bce_display_instruction_.pl1
: needed to display just one instruction (or multi-word instruction), providing status of how much displayed.

bce_display_scu_.pl1
: needed not to try to follow addresses, etc. found within machine conditions.

bce_exec_com_.pl1
bce_exec_com_input.pl1
: formed out of exec_com, abs_io_, etc. Very much simplified but maintaining all pertainent functions of the originals. The main changes dealt with differences

in storage management, switch handling and error recovery.

bce_get_flagbox.pl1
does not call phcs_ and hphcs_ to do its work.

bce_inst_length_.pl1
does not try to follow addresses found in xec instructions.

bce_relocate_instruction_.pl1
does not try to fiddle with xec instructions.

bootload_qedx.pl1
uses different storage techniques.

The next group of modules are those already present within bce (as of MR10.2) but modified to either fix problems or enhance features. These are:

bce_console_io.pl1
for new bce switch strategy and for put_chars_alert entrypoint.

bce_error.pl1
made to have its messages follow com_err_. Also provided com_err entrypoint for com_err_ to call.

bce_query.pl1
for new bce switch strategy.

bootload_0.alm
bootload_1.alm
renumbered to match their containing collections.

dctl.alm
adds the entrypoints bootload_read and bootload_write. They match the entries read_sectors and write_sectors except that the system is wired at the time and that the routine used for posting i/o completions is bootload_disk_post.pl1 instead of vtoc_interrupt. (bootload_disk_post posts completions in an area described by bootload_post_area.incl.pl1, maintained by the caller of dctl$bootload_(read write).) These entries are used for high volume, overlapped disk i/o by bootload Multics. Although not used in this installation, they have been tested by use in programs currently under development. Bootload disk i/o's use the "bootload" flag in a queue entry (see dskdcl.incl.(alm pl1)), replacing the obsolete "swap" flag.

disk_control.pl1
added support for bootload disk i/o's queued through dctl.

establish_config_deck.pl1
simplified, corrected as to when to read/write config deck.

execute_sc_command_.pl1
renames the BOS command to bce.

fill_vol_extents_.pl1
adds the default partitions "file" and "bce".

fim.alm
fixed a few fault paths that apparently only bce ever encountered.

find_rpv_subsystem.pl1
allows the "skip" or "skip_load" command before entering rpv data. This gives one a way to bypass loading firmware into the bootload mpc. The program was also modified to use hc_load_mpc to test the rpv disk mpc.

formline_.alm
understands bce switches.

hc_load_mpc.pl1
added the entrypoint "urc", which accepts a set of firmware images to load. This entry does the right thing for loading firmware overlays into urc mpcs. Also added was the entrypoint "test_controller", used to see if a controller is dead (needs firmware).

hphcs_.alm
entry hphcs_$call_bos modified to hphcs_$call_bce.

init_bce.pl1
sets up bce switches.

init_clocks.pl1
makes clock setting friendlier.

init_early_config.pl1
follows latest config card conventions.

init_pvt.pl1
correctly sets the write_limit for bce operation.

ioa_.pl1
simplified, by virtue of allowing formline_ to understand bce switches.

ocdcm_.pl1
maintains the status bit wired_hardcore_data$abort_request, checked by bce_check_abort. Also doesn't list consoles on "crash" initialization pass.

page_fault.alm
fixed a bug in the find_core loop when the paging pool is small.

privileged_mode_ut.alm
has renamed entrypoints bce_and_return and bce. These know enough to invoke the bce toehold, rather than the BOS toehold (since BOS is now useless at a crash).

pxss.alm
modified to poll disks when a page wait within collection 1 times out. This makes bce's paging operations more robust and more consistent with disk recovery within Multics service.

read_disk.pl1
added no_test entrypoints to skip the call to test disks. This speeds up certain bce disk operations.

sc_parse_.pl1
(with system_control_commands.incl.pl1) renamed bos command to bce.

scas_init.pl1
to be quiet during crash initialization and to know address of bce toehold.

scs.cds
bound with hardcore_sct_seg and wired_hardcore_data for space saving.

shutdown_file_system.pl1
calls pmut$bce.

sys_trouble.alm
invokes the bce toehold instead of the BOS toehold.

syserr_real.pl1
now calls pmut$bce_and_return.

system_startup_.pl1
calls hphcs_$call_bce for the renamed "bce" command.

wired_hardcore_data.cds
bound with scs and hardcore_sct_seg for space saving.

wired_shutdown.pl1
calls pmut$bce.

Other minor changes took place for bce sake. To make certain routines work in bce where the segment length of buffers, etc. is shorter (because of lack of disk space to page off of), during bce operation, sys_info$max_seg_size is set to the max length of bce work areas. The correct value is saved and restored when bce completes. The value of sys_info$max_seg_size used during bce can be found in sys_info$bce_max_seg_size, used by bootload_fs to avoid inserting too big a file within the bce file system. The value of sys_info$max_seg_size used is found by dividing the amount of the "bce" partition reserved for such purposes (a constant of 128 pages) by the number of temp segments found in the MST header (bootload_temp_1..N). Thus, a site can trade off buffer size for numbers of buffers and work areas.

## New bce Command Routines

The following routines have been written for bce use.

Accessing locations in the saved Multics image is performed through the routine bce_appending_simulation.pl1. This routine knows how to find any absolute or virtual address in the saved image, part of which is on disk and part of which is in memory. It can also switch to different process address spaces by being supplied different dbr values. This program provides virtual access for the bce_dump and bce_probe programs.

bce_dump.pl1 performs the equivalent of the BOS fdump program. It is pretty much modeled after the later, as far as its decisions toward what to dump. As such, its operation should not surprise anyone. It does clean up, though, some aspects of the argument processing faulty in BOS. The operation of the program is best found in the description in the documentation below.

bce_probe.pl1.pmac and bce_probe_data.cds provide most of the functions of bce's probe facility. bce_probe contains a request line parser (for dividing lines into tokens) as well as most of the functions of probe. Separate internal routines exist to parse addresses and values and to display data. A few special routines are kept separate to resemble their Multics counterparts. Also, bce_probe uses bce_appending_simulation. Support routines for bce_probe are:

bce_display_instruction_.pl1
routine to display a single (possibly multi-word) instruction without trying to follow addresses within the instruction.

bce_display_scu_.pl1
: displays scu data found within machine conditions, again, without trying to go beyond the data found in the machine conditions.

bce_inst_length_.pl1
: returns the length of an instruction, again without examining address values.

bce_name_to_segnum_.pl1
: maps segment numbers to names, names to numbers, etc. It traverses the slt in the saved image.

bce_probe_fetch_.pl1
: contains the logic to, given a generalized address, fetch the required memory/disk/whatever.

bce_relocate_instruction_.pl1
: performs address relocation on an instruction to be breakpointed. Does not play with anything other than what is supplied.

The config deck editor is imbedded in config_deck_edit_.pl1. At the base level, this routine merely calls qedx_ to do its work. However, it uses the caller_does_io option of qedx_ to perform config deck operations. Whenever the file to be read/written by config_deck_edit_'s buffer i/o routine is a normal file, the routine uses bootload_fs_ for the i/o. When the filename is <config deck> (buffer 0), it performs the desired direction of translation between the ascii form and the binary deck. The new subroutine config_deck_parse_.pl1 understands the conventions for labeling fields, types of fields and conversions, etc. This routine is unusually tolerant of errors and changes; in particular, upon reading a card, if the format of a card changes without config_deck_data_.cds.pmac (descriptions of cards and field names) being updated, it can sense it and make the card into a "user format" card with no further errors being detected.

bce_alert.pl1 writes a message on the operator's console with audible alarm.

bce_die.pl1 and bce_alm_die.alm query the operator and then irretrievibly disable bce.

bce_check_abort.pl1 manages bce operation interrupting. It is called within bce_console_io and within any purely computational loop that can become an infinite loop to see if the current operation is to be aborted. (Within ring 0 it is very difficult, if not impossible, to conceive of a routine that would intercept the interrupt from the operator's console and manage to signal quit on the correct stack at the right time. So, instead, ocdcm_ simply records the desire to abort (wired_hardcore_data$abort_request) which is checked by this

routine. Since this routine is called on the output side of operations (in bce_console_io), it follows that most operations will not proceed far (from the operator's point of view) before noticing the request to abort. Unfortunately, the few possibly infinite loops in programs must call this routine also.) This routine handles the protocol for aborting functions, as specified by the operator's response.

bce_continue.pl1 checks the validity of continue requests and calls pmut$special_bce_return to invoke the toehold to restart Multics. bce_esd.pl1 modifies the machine conditions to cause an emergency shutdown and calls bce_continue.

bce_fwload.pl1 implements firmware loading. It scans the config deck to determine what firmware is required for the specified mpcs. The actual loading is performed by hc_load_mpc.

bce_get_flagbox.pl1 implements flagbox setting and getting.

bce_query_af.pl1 implements the query/response active functions.

bce_severity.pl1 knows where to find the severity indicators for various commands (currently only dump).

bce_shutdown_state.pl1 reads the shutdown state from rpv.

bootlood_fs_cmds_.pl1 contains the bce commands to invoke bootload_fs_' file primitives.

## System Debugging Support

As mentioned elsewhere, bce provides a facility to dump Multics (dump) and to patch and probe it (probe). These make up the main part of system debugging support. However, since this installation breaks BOS fdump, and since BOS will not be available to provide a dump in the future, a way was needed to provide a dump of early initialization. This is provided through the early dump facility.

A simple program imbedded in collection 0, bootload_early_dump.alm is capable of dumping 512k of memory to tape. (During times of failures like this, only the 512k of memory used by bce is meaningful; nothing will be on disk.) The tape produced by this dump is in non-standard form. It is read and converted into a normal style on-line dump by read_early_dump_tape.pl1. The early dump program is automatically invoked upon any failure in collection 0 and any collection 1 failure when the normal toehold is not active. With this, failures rather early in initialization can be dumped. Thus,

dumps can be taken even during new hardware testing, assuming one can run far enough to get to the early dump program.

## Breakpoint Support

Providing the ability for bce to set breakpoints in bce itself and in hardcore in general required modifications to hardcore segment creation/operation to add room for the breakpoints and to various routines to handle the breakpoints.

The mechanism used to implement a breakpoint revolves around a "drl -1" instruction being interpreted (in ring 0) as a breakpoint. This requires fim.alm to special case this (when it special cases derails in general). fim contains the breakpoint handler which simply saves away machine conditions in breakpoint_page (after modifying the machine conditions to pass the derail instruction) and then calls pmut$bce_and_return. When returned to, it restores the machine conditions.

For this to work for bce, initialize_faults.pl1 had to be modified to set up fim$drl_entry as the fault handler for derails in collection 1 as well as later.

The method of providing areas for breakpoints is imbedded in various programs. First of all, bootload_loader.alm looks for all segments that are executable. For them, it makes their page tables one word longer. This extra word holds a ptw describing breakpoint_page. All executable wired segments share this page to hold breakpoints. Up to 120 breakpoints may be set in breakpoint_page (see bce_breakpoint_page.incl.pl1). make_sdw.pl1 also checks for executable segments and adds an extra page. If the segment is wired, it threads breakpoint_page as that extra page. Otherwise, it uses another "hc" partition page. This policy means that only one page of wired memory is used up for hardcore breakpoint support.

A few programs must special case breakpointable segments (slte.breakpointable is on). delete_segs nulls out ptws referencing breakpoint_page before truncating them to avoid having page control become unhappy. make_segs_paged, collect_free_core and move_non_perm_wired_segs also need not to free the breakpoint page found when paging/moving/freeing a segment.

## New Tools and Tool Changes

Because of the addition of two new MST collections, check_mst needs another change to its tables to describe them. Because some of these collections are loaded paged but their

linkages, etc. are wired, a new attribute was added to check_mst's data: "last_text_wired_collection" (to complement the existing "last_[anything]wired_collection"). Various other bug fixes were also made to check_mst and friends so that the checker output would be correct.

To handle the tapes generated by bootload_early_dump, the new program read_early_dump_tape.pl1 (redt) was created. It reads the tape, creating a simulated 512k memory. With the help of ed_appending_simulation.pl1, it pretends to be the bce dump program and thereby creates a standard format dump.

get_flagbox was modified to be able to set/get the new flagbox field "return_to_bce_command", named "bce_command" to the get_flagbox.

A primitive command (which may get enhanced *one day*), bootload_fs allows access during service to the bce file system. Using hphcs_$(read write) partition, it can insert a new bce file system. Options to bootload_fs allow insertion, deletion, renaming, etc. of files within a copy of the bce file system, which may be inserted during service.

To provide a level of compatibility between the labeled config deck form, used by the config deck editor, and the old unlabeled form shown by print_configuration_deck, the user ring config tools were updated to allow the new labeled form. Both print_configuration_deck and compare_configuration_deck now take the "-label" ("-lbl") control argument to display the output with labels. (Also, compare_configuration_deck was changed to allow two pathnames to be supplied.) The new routine convert_configuration_deck takes the output from print_configuration_deck (with or without labels) and converts it back to binary. This operation is provided to allow a test of a given ascii config deck (trying a convert performs some level of validation of cards) as well as allowing one to convert an ascii form to binary for comparison with the current binary version.

## Auto mode support

Auto mode support includes facilities added so that bce may auto re-boot Multics upon a crash. Generalized, it is a set of instructions that may be left for bce from either Multics or bce to be executed whenever bce finds itself in control. This is controlled mainly through the "return_to_bce_command" field in the flagbox (refered to as "bce_command" to (get set)_flagbox). This field overlays the old "blast" message field, which doesn't work. Auto mode support starts with access to this new and other flagbox variables, given the new flagbox_mgr.pl1, get_flagbox.pl1, and hphcs_.alm and phcs_.alm changes therefor. It also includes the bce command/active function for flagbox

queries, bce_get_flagbox. Also in the realm of status for bce to examine is the shutdown_state of the rpv (bce_shutdown_state.pl1).

To provide the equivalent of the BOS auto runcom, the exec_coms auto.ec, dump.ec, go.ec and rtb.ec are provided. Their use is described in the documentation below.

## V. DOCUMENTATION CHANGES

The documentation changes described below are meant to describe only the initial installation of bce. As such, they purposefully contain information describing the presence of both BOS and bce in ways that will be removed once the other sections of bce are completed.

The page numbers in this section align, section number-wise, with the manual to which they belong and therefore appear to jump about.

The documentation items that follow are (in order):

Commands and Active Functions
System Release Bulletin
Installation Instructions
Hardware and Software Formats
Multics Operator's Handbook

SECTION C-AF

COMMAND DESCRIPTIONS

Add the new command read_early_dump_tape to the System Maintainer's Guide:

Name: read_early_dump_tape (redt)

The read_early_dump_tape command reads the contents of a tape produced by the early dump facility of bce to produce a standard format dump in a specified directory.

Syntax: read_early_dump_tape reel_num {-control_args}

Arguments:
reel_num
is the reel number of the early dump tape. This argument may by placed anywhere on the command line.

Control_args:
-erf N
generates a dump with erf (error report form) number of N. This control argument is required.
-dump_dir directory
places the dump into the specified directory. The default is to place the dump into >dumps.
-density, -den N
sets the tape density to N. Unless site modified, early dump tapes are written at 1600, which is the default.
-ring, -rg
mounts the tape with a write ring.

Add to the description of print_configuration_deck and compare_configuration_deck:

-label, -lbl
displays cards with mnemonic labels for each field.

-no_label, -nlbl
     does not display field labels. This is the default.

     Change the description of compare_configuration_deck as follows:

Syntax: compare_configuration_deck path1 {path2} {-control_arg}

Syntax as active function: [compare_configuration_deck path1 {path2}]

Function: compares either a saved copy of the configuration deck or the configuration deck for the running system to a saved copy. When used as an active function, returns either "true" or "false" to indicate whether the two configuration decks are equivalent.

Arguments:
path1
     is the pathname of a saved copy of the configuration deck.
path2
     is the pathname of a copy of the configuration deck to be compared against path1. If this argument is not supplied, >sl1>config_deck (the configuration deck for the running system) is used.

     Add the description for the new command convert_configuration_deck:

Name: convert_configuration_deck

     The convert_configuration_deck command converts an ascii source form of a configuration deck, as produced by print_configuration_deck, into a binary (system) form.

Syntax: convert_configuration_deck ascii_path binary_path

Arguments:
ascii_path
     is the pathname of an ascii source form of a config deck. Both labeled and unlabeled fields may appear on the config cards. The archive convention is allowed.

binary_path
     is the pathname of the resultant binary config deck. The form is compatible with the system config deck.

Notes:
This command is intended to be used to perform a level of validation on a proposed new ascii config deck. It may also be used to convert an ascii config deck into the form required by compare_configuration_deck.

Add the description for the new command bootload_fs:

Name: bootload_fs

The bootload_fs command allows the user to operate on a copy of the bootload Multics (bce) file system, including the ability to extract the real bce file system and to replace it with this operating copy.

Syntax: bootload_fs operation {args}

Arguments:
operation
is an operation listed below under "List of Operations".
args
are arguments required by the designated operation.

List of Operations:

The operations are grouped into two categories. The first group determines the location of the user's copy of the bce file system; operations in this group can also extract the real bce file system and overwrite the bce file system with the user's copy. The second group operates on objects in the user's working copy of the bce file system.

Operation: get_partition, get_part

The get_partition operation reads the bce file system from a specified disk partition into the user's working copy thereof overwriting the previous contents of the user's copy.

Syntax: bootload_fs get_partition pv_name part_name

Arguments:
pv_name
is the name of a mounted physical volume.
part_name
is the name of a partition on the specified volume to be read.

Notes:
Acess to hphcs_ is required.

Operation: put_partition, put_part

The put_partition operation replaces the bce file system found in the specified disk partition with the user's local copy.

Syntax: bootload_fs put_partition pv_name part_name
or bootload_fs {-force}

Arguments:
pv_name
is the name of a mounted physical volume.
part_name
is the name of a partition on the specified volume to be read.

Notes:
If no arguments are supplied, put_partition will use the identity of the partition last specified in a get_partition operation. Specifying "-force" will suppress the query as to overwriting the old partition.

Access to hphcs_ is required.

Operation: use_partition, use_part

The use_partition operation copies the contents of a user specified segment to become the user's working copy of the bce file system.

Usage: bootload_fs use_partition path

Arguments:
path
is the pathname of a segment which will overwrite the current contents of the user's local copy of the file system.

Operation: save_partition, save_part

The save_partition operation saves the current contents of the user's local copy of the bce file system into a segment.

Usage: bootload_fs save_partition path

Arguments:
path
is the pathname of a segment which will be overwritten with the user's working copy of the file system.

Operation: discard_partition, discard

The discard operation discards the contents of the working copy of the bce file system. This operation must be followed by another get_partition, use_partition of init_partition operation.

Usage: bootload_fs discard_partition {-force}

Operation: init_partition, init

The init_partition operation clears out the contents of the working copy of the bce file system. It differs from discard_partition in that the result is a file system containing no files; the result of discard_partition is no file system at all.

Syntax: bootload_fs init_partition {-force}

Operation: get_file, get

The get_file operation extracts a file from the working copy of the bce file system and places it into a Multics storage system file.

Syntax: bootload_fs get_file file_name path

Arguments:
file_name
is the name of a file within the working copy of the bce file system.
path
is the pathname of the Multics file into which the bce file is to be copied.

Operation: put_file, put

The put_file operation places a copy of a Multics storage system file in the working copy of the bce file system.

Usage: bootload_fs put_file path file_name

Arguments:
path
is the name of a file in the Multics hierarchy to be copied into the bce file system.
file_name
is the name the copy is to have within the bce file system.

Operation: list_files, list

The list_files operation lists the names and lengths (in characters) of the files in the working copy of the bce file system.

Usage: bootload_fs list_files

Operation: delete_file, delete

The delete_file operation deletes files from the working copy of the bce file system.

Usage: bootload_fs delete_file file_name

Arguments:
file_name
is the name of a file that is to be deleted from the bce file system.

Operation: rename_file, rename

The rename_file operation renames a file within the working copy of the bce file system.

Usage: bootload_fs rename_file old_file_name new_file_name

Arguments:
old_file_name
is the name of an existing file in the bce file system.

new_file_name
is the new name to be given to the old file.

# SECTION SRB

## SIGNIFICANT CHANGES IN THIS RELEASE

This release contains the first installation of Bootload Multics, also known as the Bootload Command Environment (bce). Bootload Multics is a new phase of Multics initialization. It allows the operation of Multics with or without BOS. The ability to "warm" boot Multics from disk is provided by Bootload Multics; that is, to boot without the MST mounted on a tape drive.

Bootload Multics provides a new ring zero command level. The functions of warm booting, dumping and examining memory, and emergency shutdown are performed from this command level. These functions may no longer be performed from BOS. Also, automatic operation is driven from the Bootload Multics command level. The BOS functions of SAVE/RESTOR, SAVE COPY, CORE SAVE/RESTOR, as well as a few specialized functions, are not yet available. Also, printer support is not yet available.

The operation of Bootload Multics is described in detail in Section 5.5 of the Multics Operator's Handbook, Order Number AM81. This material must be read prior to attempting a boot of this release. The presence of Bootload Multics will have no effect on any user's process or application.

Bootload Multics requires 2455 pages of disk space on the rpv for its operation, split between the new "bce" and "file" partitions. These partitions will be automatically created when this release is booted for the first time; the site, however, must assure that a sufficient amount of space on the rpv exists.

For a better discussion of the changes involved with Bootload Multics, refer to Appendix X of this SRB.

This release provides the ability of site maintenance personal to set "probe" breakpoints in hardcore. When a breakpoint is encountered, Bootload Multics will be invoked to allow the analysis of the machine conditions. The breakpoint conditions may be modified and then Multics restarted.

SECTION SRB-X

# BOOTLOAD MULTICS (BCE)

MR11 includes a new phase to initialization known alternately as Bootload Multics or the Bootload Command Environment (bce). This release provides the first installment of bce; future releases will provide further enhancements. The goal of bce is to allow Multics to be operated without BOS. This installation provides the basic facilities that a site must have to run without BOS. Certain facilities present in BOS, used at some sites, may not be present in this installation of bce; for these facilities, the site will use BOS, just as in previous releases.

The intent of this appendix is to describe bce in terms of its difference from the previous method of operation (i.e. BOS). This information should be used in conjunction with the description of bce appearing in the MOH.

## THE MST

bce, is not, as was BOS, on a separate tape from Multics. Both bce and Multics originate on the same tape, the Multics System Tape (MST). bce is an integral part of the Multics initialization software. It both uses and is used by the Multics initialization software.

bce/Multics may be booted from BOS or via the IOM/IMU boot function. When booted from BOS, it is not necessary to load firmware into the various controllers or set the system clocks from bce, as this will already have been done from BOS. Also, the configuration description needed by Multics is passed up from BOS. When booted from the IOM, the firmware loading, clock setting and config deck preparation are all done from bce.

Once booted, bce has no further use of the MST. Multics service can be booted directly from bce without the aid of the MST tape. This is because the needed contents of the MST tape are saved in a partition of the rpv. This is an important difference from the previous BOS method of operation. Note,

then, that an MST tape is not kept mounted on a tape drive during periods of auto-reboot-mode operation.

## BOOTING

The equivalent of booting BOS is now to boot bce. Under normal circumstances, bce is booted once within a given series of boots of Multics service. It serves an equivalent function to BOS in that it forms a platform from which Multics is booted and to which Multics crashes or shuts down.

The booting of bce has the same significance as did the booting of BOS previously, even if bce is booted from BOS. That is to say that Multics service, although grown from bce, is to be considered as a separate entity from bce, just as Multics and BOS were considered separate and distinct entities in the past. When Multics crashes or shuts down, Multics, as an entity, relinquishes control of the system; bce, as an entity, takes over control. bce can then perform emergency shutdown and dumping of Multics.

bce can be booted from BOS, if BOS will be needed later, or bce can be booted alone. The actual sequences for booting bce appear in the Installation Instructions and in the MOH.

The directive "boot" now has three possible meanings. When used in BOS, it means to boot an MST, thus starting up bce. When used at the bce "early" command level, it means to boot bce (actually, to continue to boot bce). When used at the bce "boot" (or "bce_crash") command level, it means to boot Multics service.

The next two sections provide some comments on the new booting procedures.

### Booting bce from BOS

When BOS is booted first, and bce is booted from BOS, BOS is used for those hardware and configuration initialization functions for which it has always been used. Once bce is booted from BOS, though, BOS is out of the picture as far as system operation is concerned. bce/Multics will not return to BOS under any circumstances unless the operator so directs. For this and even more fundamental reasons, certain functions, previously performed by BOS, can no longer be performed at BOS. These include ABS, BLAST, DUMP, ESD, FDUMP and PATCH.

The process of booting BOS, as well as BOS itself, must perform certain initialization functions before booting bce. These are listed below, in order as they are performed.

The IOM/IMU INITIALIZE/BOOT function is invoked. The FWLOAD function of BOS is read into memory in the process.

The bootload tape MPC is loaded by answering the prompt, "Enter tape controller type:".

The other MPCs (disk MPCs as well as other tape and unit record controllers) are loaded by supplying their types and channel addresses to the FWLOAD prompts.

BOS is booted (which is irrelevant as far as bce is concerned).

The config deck is generated or corrected.

bce/Multics is booted.

The BOS BOOT function is used as before to boot the bce/Multics tape. The booting of Multics will appear as it would in the past. The only visible difference is that Multics stops at a new command level it did not have before. This is the bce (ring-0, if you wish) command level. The bce command level can be detected by the presence of the bce ready message:

bce (boot) TIME:

The word "boot" is sometimes replaced by other names, depending on the system state. These are discussed later.

For more details on booting bce from BOS, refer to the MOH.

The normal day-to-day functions previously performed by BOS will now be performed at this bce command level. The function of booting Multics service takes place from bce. Also, when Multics crashes or shuts down, the system will return to bce, instead of BOS, so that emergency shutdown and dumping can be performed.

## Booting bce from the IOM/IMU

The IOM/IMU INITIALIZE/BOOT function can be used to boot bce. In this case, all hardware and configuration initialization functions previously performed by BOS are performed by bce itself. In most cases, the system requests the performance of these functions in the correct order. It might be worthwhile, though, to describe the initialization functions that must be performed, in order, during the process of booting bce. For more details, refer to the MOH.

The IOM/IMU INITIALIZE/BOOT function is invoked. Collection 0 of bce will be read in as a result.

Firmware is loaded into the bootload tape controller. This is done by answering the "Enter boot tape MPC model" query.

The bootload disk controller is booted. This is done by answering the query "Enter RPV data". This also locates the RPV.

The config deck is generated or corrected. During a cold boot, the config deck is generated by using the config deck editor at the "bce (early) TIME:" prompt. For a normal boot, the config deck is read from the rpv specified in the previous step and brought up to date, if necessary, by the config deck editor. When this is done, enter "boot" to complete the booting of bce. (bce is not fully booted until it reaches the "boot" state.)

The system clock is set. The time value is requested after entering "boot" above.

All other disk mpcs are booted, if necessary. This is done in the load_disk_mpc dialog.

All other controllers are booted. This is done with the fwload (fw) command at the "bce (boot) TIME:" prompt.

Notice that the same initialization functions are performed as were previously performed by BOS, but their order is different. The only function that the operator must explicitly remember to do is to load the other MPCs at the "bce (boot) TIME:" prompt.

Once the other mpcs have been loaded, bce can be considered fully initialized. At this point, the system will be sitting with the prompt

bce (boot) TIME:

This prompt signifies that the system is at bce command level, a new (ring-0) command level. In the course of booting Multics, this may be simply viewed as another command level (along with the ring-1 and ring-4 command levels) in the process of booting. However, this command level signifies that bce has control with the same significance as we used to say that BOS has control in the past. The function of booting Multics service takes place from this command level (bce "boot" command level).

## NORMAL OPERATION OF BCE

The types of operations to be performed from bce are a subset of those previously performed at BOS. (Eventually all

functions BOS performed will be performable from bce.) These operations include booting Multics service, taking dumps of a crashed Multics system, and invoking emergency shutdown of Multics. The commands to perform these basic operations are similar in operation and appearance to their BOS counterparts. In particular, these equivalences are:

```
BOOT -> boot
ESD -> esd
FDUMP SHORT -> dump -short
etc.
```

The invoking of the standard BOS runcoms is replaced by invoking the corresponding bce exec_coms. Thus:

```
AUTO -> ec auto star
GOGO -> ec go
RTB -> ec rtb
```

Other than these standard operations, the commands within bce differ considerably from BOS. The MOH should be consulted for the operations of these commands.

The MOH lists the functions currently performed by bce. Certain other functions such as SAVE/RESTOR must still be performed from BOS. If such operations are desired (those performable by BOS but not yet performable by bce), it is necessary to return to BOS. If bce was booted from BOS, simply use the bce "bos" request. The BOS GO request will restart bce where it was. If bce was not booted from BOS, it will be necessary to boot BOS. Boot BOS only after successfully shutting down Multics service.

## STATE OF THE SYSTEM

bce can be in various states. The state of bce can be found from the bce ready message/prompt:

```
bce (STATE) TIME:
```

The "early" state normally appears only once, when booting bce initially. When bce is in this state, the purpose is to have the operator generate or correct the config deck, followed by setting the clock when the system leaves this state (by entering "boot").

The "boot" state is the normal state of bce. In this state, Multics service may be booted.

If bce should fail, the "bce_crash" state is entered. When this occurs, the dying bce is saved and can be dumped (if this is desired). From the "bce_crash" state, one can enter

"reinitialize" to return to the "boot" state from which one can then boot Multics service. As a short cut, Multics service can be booted directly from the "bce_crash" state by entering "boot"; this performs a reinitialize and a boot of Multics service without stopping at the bce "boot" command level.

When Multics crashes, bce is in the "crash" state. This state exists just so that the operator is reminded that a dead Multics exists which should be dumped and shut down (esd).

## THE TOEHOLD AND 'EXECUTING SWITCHES'

BOS has a toehold. The toehold is a small program that was the main driver when switching between Multics and BOS. It also held the communication flags between BOS and Multics (the flagbox). The toehold is located in memory at absolute address 10000 (octal).

bce also has a toehold used in much the same way as the BOS toehold.

Since the BOS toehold is being kept around for this release (as the driver for switching between BOS and bce (when BOS is used)), the bce toehold (used for switching between bce and Multics) must be at a different location. The bce toehold is at absolute location 24000 (octal).

Thus, "executing switches" to force a manual return to bce uses a different switch value than does forcing a return to BOS. This switch value is "024000717200". When "executing switches" on a L68 processor, this is the value to enter into the data switches.

Since the toehold address to crash Multics has changed, the DMP's BOS command is no longer used. Instead, it is necessary to enter the above switch value into the data switches manually (CO DATA 024000717200) and then force execute the data switches (EX2).

The system will warn the operator if the data switches on any processor are not set to the above value.

## GENERAL OPERATION OF BCE

The operation of the bce command level differs both from the old BOS command usage and the Initializer ring-1 and ring-4 command level usage. The syntax and usage of this bce command level is much more that of standard Multics command level. (This is described in the MPM Reference manual and will not be repeated here.) In particular, active functions and command iteration are used. exec_com's (version 1) are available. Normally, though,

commands are typed simply as a command name followed by a collection of arguments, separated by spaces.

The commands within bce attempt to resemble their counterparts (if any) within service Multics. For details of the execution of any given command, refer to the MOH.

Some things to remember about bce operation follows.

The text editor used to edit bce files (equivalent of BOS runcoms) is qedx. It bears no resemblance to the BOS EDIT command. The description of qedx appears in the manual, Commands and Active Functions. The version of qedx within bce differs from the standard version in that there is a query if one tries to exit qedx with unwritten modified buffers.

Config decks are edited with the config_edit (config) request. The config command in bce bears absolutely no resemblance to the CONFIG command in BOS. One must remember when one is in the config deck editor that one is really in qedx.

The bce equivalent of BOS runcoms is version 1 exec_coms. These have absolutely no resemblance to BOS runcoms. An important thing to remember when converting BOS runcoms to bce exec_coms is that, when a BOS runcom invokes another runcom, that second runcom will never return to the first. In bce, an exec_com will return to its invoking exec_com. Also, a BOS runcom that boots service regains control when Multics crashes or shuts down. In bce, the invoking exec_com (and all exec_coms which invoked the exec_com that booted service) lose control whenever a "boot", "esd" or "go" operation are performed. The exec_com that bce invokes whenever Multics crashes or shuts down is determined solely by the "bce_command" flagbox variable. It is this variable (and other flagbox flags) that are manipulated by the auto exec_com procedures.

Within BOS, functions were aborted by hitting RETURN or EOM on the operator's console. There was no way to indicate that this was an accident. Within bce, hitting RETURN or EOM allows the operator to indicate the intention to abort a function. When this occurs, the system will ask "Abort?" to which the operator may answer "no" or "yes" appropriately to abort the current operation. Other responses are allowed; refer to the MOH for more details.

SECTION II-4

# INSTRUCTIONS FOR SITES UPDATING FROM PREVIOUS RELEASE

## STEP-1

Using the current BOS System, (i.e., not system MR11) perform SAVE. A double save is recommended to avoid any possible tape problems later.

With this release two new partitions are required on the rpv for bce operation. These are the "bce" partition of length 2200 records and the "file" partition of length 255 records. These may be created in one of two ways.

### Automatic Partition Creation

A first boot of MR11 is capable of creating the required bce partitions. To allow this to work, at least 3000 records (2455 for the new partitions and around 500 for rpv only segments created during later initialization) must be free on the rpv. To determine if this is so, execute the following (using the current system):

```
list_vols -pv rpv
```

the second number appearing after the drive name is the records available on rpv. If this number is greater than 3000, MR11 may simply be booted at this time. The required partitions will be created at the high end of the disk, just below any partitions currently at the high end. This generation takes on the order of 10 minutes during this first boot.

If there are not enough free records on the rpv, some segments will have to be moved to other drives. Use the sweep_pv command to move small collections of segments to other physical volumes in the RLV until enough free space exists.

## Manual Partition Creation

The required partitions may also be created manually, using rebuild_disk performed with the current system.

Before starting the rebuild_disk of the rpv, it will be necessary to add to the config deck's parm card:

```
PARM DIRW
```

Sites that don't normally run with the "dirw" parameter should remove this from the config after the rebuild_disk is complete.

Assume that the rpv is mounted on dska_01 and a scratch pack is mounted on dska_02. Boot, using the current Multics System Tape (MST), not the MR11 MST, to ring-1 command level by executing the following:

```
boot N (where N is the drive number of the current MST)

rebuild_disk rpv dska_01 -copy dska_02
```

The rebuild_disk command will report information about the size of the partitions found on the source rpv. This information should be noted for use when constructing the new partitions on the target rpv.

The rebuild_disk command will prompt for input, at this point the new partition layout should be entered as in the example that follows:

```
request:  part alt high 141 (on MSU451/400 only)
request:  part bos high 270
request:  part log high 256
request:  part dump high 2000
request:  part file high 255
request:  part bce high 2200
request:  part hc low 2500
request:  part conf low 4
request:  nvtoce <number>
request:  list
request:  end
```

When the rebuild_disk is complete, shutdown and boot with the new rpv located on dska_02. For sites with MSU500 type units, the BOS SAVE COPY command can be used to move the temporary RPV back to the original device.

## Library Cleanup

Boot using the current Multics System Tape (MST) to Initializer ring-1 command level and type:

```
boot N (where N is the drive number of the current MST)
alv -all
standard
admin
<admin password>
```

Due to the method by which unbundled software is dumped, normal trimming during reloading of new software does not occur. To ensure that unbundled directories are clean execute the following:

```
ldl >system_library_unbundled>**
ldl >ldd>unbundled>(source object)>**
```

Exit admin mode and continue on to the next step.

```
ame
shutdown
```

STEP-2

Step 2 involves booting bce for the first time. This step will create the partitions required by bce, if they were not so created in Step 1. A complete description of the procedure for booting bce can be found in the Multics Operator's Handbook, AM81. A sample dialog follows.

Place the MR11 Multics System Tape (MST) on a convenient drive and initiate the INITIALIZE/BOOT sequence of the IOM/IMU. The system will then proceed in the manner shown below.

```
bootload_0:  Booting system MR11 generated 11/01/84 0000.0
     est Thu.
bootload_0:  Enter boot tape MPC model:  t500
bootload_0:  Booting t500 A 12.  with mtc500 rev.u1
     firmware.
bootload_0:  Booted tape MPC.
0000.1 announce_chwm:  347.  pages used of 512.  in wired
     environment.
0000.2 announce_chwm:  620.  words used of 1024.  in
     int_unpaged_page_tables.
Enter RPV data:  query
Enter RPV subsystem base channel, as Icc, or "cold".  A22
find_rpv_subsystem:  Enter RPV subsystem MPC model:  609
hc_load_mpc:  Booting channel A22 with dsc500 Revision j1.
find_rpv_subsystem:  Enter RPV disk drive model:  451
find_rpv_subsystem:  Enter RPV disk device number:  5
find_rpv_subsystem:  RPV is a model 451 drive, number 5 on
     MPC A22.
    Is this correct?  yes
0000.4 init_root_vols:  Adding bce file partitions to rpv.
0007.0 find_file_partition:  Initting file partition.  Data
```

```
     not in expected format.
0008.0 load_mst:  627.  out of 1048.  pages used in disk mst
     area.
bce (early) 0001.5:
```

At this time, bce has been booted. The previous boot of MR10.2 saved the config deck in the "conf" partition of the rpv; the current config deck will be set to this. The operator should ensure that this is so. The config deck should be made correct at this time using the config deck editor within bce. Proceed with the boot if the config deck is okay.

```
bce (early) 0001.5:  config
1,$p
          (The config deck will print at this point.)
q
bce (early) 0001.7:  boot
          (The operator continues to boot bce.)
Current system time is 01/01/01 0001.8 est Tue.
Is this correct?  no
Enter time as yyyy mm dd hh mm {ss} :  1984 11 02 12 00
Current system time is:  11/02/84 1200.0 est Fri.
Is this correct?  yes
load_disk_mpcs:  Disk mpcs mpca mpcc appear not to be
     operating.
Enter disk mpc names to be loaded, or "none" or "abort":
     mpca mpcc
          (The operator enters the  names of other disk mpcs
          to be loaded.)
hc_load_mpc:  Booting channel A20 with dsc500 Revision j1.
hc_load_mpc:  Booting channel B20 with dsc500 Revision j1.
Enter disk mpc names to be loaded, or "none" or "abort":
     none
bce (boot) 1200.5:
```

At this time, the operator must load firmware into all other controllers (i.e., not the bootload tape controller nor any disk controllers). bce is then considered to be fully initialized.

```
bce (boot) 1200.5:  boot
Multics MR11 - 11/02/84 1201.0 est Fri.
Command:
```

SECTION II-5

INSTRUCTIONS FOR SITES INSTALLING FOR THE FIRST TIME

STEP-3

Mount the Multics System Tape (MST) on Magnetic Tape Handler (MTH) nn (nn is usually equal to 01). Mount the disk pack formatted by T&D on the drive selected to be the RPV. Initialize and boot the MST. Multics will prompt with:

```
bootload_0:  Booting system MR11 generated 11/01/84 0000.0
     est Thu.
bootload_0:  Enter boot tape MPC model:  t500
```

Normal response to this question should be "t610", "t601", "t500" or "ipc". The system will boot the bootload tape controller, if necessary, and continue. At this time, the intention to cold boot is given. Multics will request the location of the rpv. Once this is done, the init_vol request loop will be entered to accept the layout of the rpv.

```
bootload_0:  Booting t500 A 12.  with mtc500 rev.u1
     firmware.
bootload_0:  Booted tape MPC.
0000.1 announce_chwm:  347.  pages used of 512.  in wired
     environment.
0000.2 announce_chwm:  620.  words used of 1024.  in
     int_unpaged_page_tables.
Enter RPV data:  query
Enter RPV subsystem base channel, as Icc, or "cold".  cold
Booting cold will destroy all data on the RPV.
   Are you sure that you want to boot cold?  yes
Enter RPV subsystem base channel, as Icc.  A22
find_rpv_subsystem:  Enter RPV subsystem MPC model:  609
hc_load_mpc:  Booting channel A22 with dsc500 Revision j1.
find_rpv_subsystem:  Enter RPV disk drive model:  451
find_rpv_subsystem:  Enter RPV drive device number:  1
find_rpv_subsystem:  RPV is a model 451 drive, number 1 on
     MPC A22, and this is a COLD boot.
    Is this correct?  yes
```

```
Default RPV layout:  (Respond "end" to use it.)

Average seg length = 2.00
VTOC size = 2792 pages, 13920 vtoces.
27840 paging records.
Constrained by average seg length.
part hc 2792.  2500.
part conf 5292.  4.
part alt 38117.  141.
part bos 37847.  270.
part dump 35847.  2000.
part log 35591.  256.
part file 35336.  255.
part bce 33136.  2200.
```

These are the default partition assignments. Any changes to the default partitions or RPV parameters can be redefined by using the "startover" request in init_vol. The system installer should review the write-up of init_vol in the MOH prior to the installation.

Sizes for the various partitions and their locations can be modified based on the needs of the site.

```
request:  end

init_empty_root:  Begin rpv initialization.  This will take
     some time.
init_empty_root:  rpv initialized; 27840 records.
find_file_partition:  Initting file partition.  Data not in
     expected format.
0010.0 load_mst:  627.  out of 1048.  pages used in disk mst
     area.
bce (early) 0010.2:
```

Build the configuration description as follows:

```
config
a
.
.  (Configuration fields as defined in the MOH.)
.
\f
w
q
```

Do not enter any part cards at this time, except for those partitions defined on the rpv. Also, make the root card specify only the rpv.

Continue booting bce.

```
bce (early) 0020.0:  boot
Current system time is:  01/01/01 0020.1 est Tue.
Is this correct?  no
Enter time as yyyy mm dd hh mm {ss} :  1984 11 02 12 00
Current system time is:  11/02/84 1200.0 est Fri.
Is this correct?  yes
load_disk_mpcs:  Disk mpcs mpca mpcc appear not to be
     operating.
Enter disk mpc names to be loaded, or "none" or "abort":
     mpca mpcc
          (The operator entered the names of other disk mpcs
          to be loaded.)
hc_load_mpc:  Booting channel A20 with dsc500 Revision j1.
hc_load_mpc:  Booting channel B20 with dsc500 Revision j1.
Enter disk mpc names to be loaded, or "none" or "abort":
     none
bce (boot) 1200.5:
```

At this time, the operator must load firmware into all other controllers (i.e., not the bootload tape controller nor any disk controllers). bce is then considered to be fully initialized.

```
bce (boot) 1200.5 :  boot cold
Do you really wish to boot cold?  yes
hdx: reregistered public lv root lvid 727353262340
hdx: Entry is not a branch.  cannot make mdcs in lv root
hdx: reregistered pv rpv pvid 727353262301 in lv root
disk_table_:  New disk_table created.
Multics MR11 - 11/02/84 1201.0 est Fri.
```

Ignore the messages prefaced by disk_table_ and hdx.

SECTION HSF-7

MULTICS ENVIRONMENT

Changes to Hardware and Software Formats PLM:

The changes to the Hardware Software Formats PLM are obviously enough all in the software section, Section 7, "Multics Environment." Other sections in this manual are incorrect and out of date but corrections to them do not appear here.

## MAIN MEMORY MAPS

The following paragraphs describe the gross allocation of main memory during the three distinctly different Multics operational environments: BOS, bce and service.

In the address charts that follow, the addresses are absolute octal memory addresses. Whenever an address appears in brackets ([]), this means that the object described is contained within the segment listed above it.

### Common Areas

Certain areas are common between the three modes of operation; these areas are dictated mostly by hardware requirements.

#### FAULT_VECTOR

The fault_vector area holds vectors and its pointers used for handling interrupts and faults. This area is described below.

address

000000 interrupt vectors
contains interrupt pairs, each containing a scu/tra pair specifying absolute addressing. The target of the addresses is in the "its" area.

000100 fault vectors
contains fault pairs, one for each defined fault, each containing a scu/tra pair specifying absolute addressing. The target of the addresses is in the "its" area.

000200 its pointers for fault and interrupt vectors
contains the its pointers that are the targets of the scu and tra instructions above. Only these its pointers are normally changed; the scu and tra instructions remain.

MAILBOXES

The mailbox area holds control areas used to converse with the ioms and the fnps.

address

001200 IOM imw area
is used to determine which channel of the iom generated an interrupt.

001400 IOM A mailbox
002000 IOM B mailbox
002400 IOM C mailbox
003000 IOM D mailbox

003400 FNP A mailbox
003700 FNP B mailbox
004200 FNP C mailbox
004500 FNP D mailbox
005000 FNP E mailbox
005300 FNP F mailbox
005600 FNP G mailbox
006100 FNP H mailbox

BOS Environment

BOS operates in segmented, nonpaged appending mode with exactly eight defined segments. The eight pointer registers are loaded with fixed segment numbers and the segment base and bound values are manipulated according to the requirements of the code.

address

```
 000000   fault_vector
 000600   padding
 001200   mailbox area
 006400   padding

 007740   ds (descriptor segment)

 010000   toehold (BOS)
[010020]  flagbox (BOS)
 011000   setup

 020000   bf (buffer)

 022000   com (common variable storage)

 031000   pgm (program area)

 040000   util (utilities)

 060000   rest of BOS memory, unused
```

The standard pointer register/segment assignments for BOS are:

```
pr0 -> ds
pr1 -> pgm
pr2 -> bf
pr3 -> setup
pr4 -> (prog temporary)
pr5 -> flagbox
pr6 -> com
pr7 -> mem (first 256k of mem)
```

bce Environment

The memory layout after the running of collection 0 (the loading of collection 1, i.e. bce) follows. All segments are paged with the exception of fault_vector, iom_mailbox and dn355_mailbox.

address

```
 000000   fault_vector
 000600   padding
 001200   iom_mailbox
 003400   dn355_mailbox
 006400   padding
```

```
 010000   bos_toehold
 012000   config_deck

 024000   bound_bootload_0
[024000]  toehold (bootload Multics)
[024000]  flagbox (bootload Multics)
[030000]  bootload_early_dump
 046000   toehold_data

 052000   unpaged_page_tables
 054000   int_unpaged_page_tables
 056000   breakpoint_page

 060000   physical_record_buffer

 066000   dseg

 070000   name_table
 100000   slt
 104000   lot

 106000   wired segments, fabricated segments, all other segments
```

Service Environment

The memory layout after the running of make_segs_paged, collect_free_core and the deletion of init and temp segs is as follows. All segments are paged except for fault_vector, iom_mailbox and dn355_mailbox.

```
address

 000000   fault_vector
 000600   padding
 001200   iom_mailbox
 003400   dn355_mailbox
 006400   padding

 010000   bos_toehold

 012000   paging use

 024000   toehold (bootload Multics)
[024000]  flagbox (bootload Multics)
 030000   paging use
 046000   toehold_data

 052000   unpaged_page_tables
 054000   paging use
 056000   breakpoint_page
```

| | |
|---|---|
| 060000 | paging use |
| 106000 | wired segments, fabricated segments, paging use. sst_seg is located at the high end of the bootload memory. |

SECTION MOH

MULTICS OPERATOR'S HANDBOOK

Global directives:

Change all references to the BOS console, iom, cpu and scu to refer to the bootload console, iom, cpu and scu.

Change all references to configuration cards to be in lower case to emphasize that they should be in lower case.

Specific directives follow.

SECTION MOH-1

# OPERATOR RESPONSIBILITIES

Change the paragraph describing the responsibility of the operator to understand BOS to:

The operators must also understand the functions of the bootload operating system (BOS) and the bootload command environment (bce), which load Multics and perform various system software maintenance activities. BOS controls the bootloading of bce and can provide one type of save of the contents of the storage system. bce controls the bootloading of Multics service as well as performing memory dumps. Initially, BOS is contained on a tape and must be bootloaded from the console into the system. If a copy of BOS already exists on disk, it can be "warm booted", preserving the contents of the disk that already contains BOS. If there is no disk copy of BOS, it must be "cold booted". bce makes up the first part of the Multics tape and is usually bootloaded from BOS. bce can be booted from the console, if necessary, without using BOS but then it cannot utilize the BOS functions.

Acronym list

BCE bootload command environment

## GLOSSARY OF TERMS

bce
: the bootload command environment; a set of programs within Multics initialization that perform functions such as bootloading Multics, dumping main memory and initiating emergency shutdown of Multics.

bootload
: to load a fresh copy of a set of programs. BOS, bce and Multics can be bootloaded. Bootloads of BOS are "cold" if they completely re-create BOS' operating environment and

"warm" if they assume that some information from previous bootloads is to be used. Bootloads of bce and Multics are "cold" if they re-create the file system, "cool" if they maintain the file system but completely re-create bce's operating environment and "warm" if they assume that some information from previous bootloads is used. The period of time between Multics bootload and shutdown is also spoken of as a bootload, or service session.

BOS
: the bootload operating system; a set of programs that perform functions such as loading bce and dumping disks.

initializer process
: change the reference to BOS to refer to bce

SECTION MOH-3

CONFIGURATION

CALENDAR CLOCK

... The BOS time command, or the bce invoked clock setting function, if BOS is not used, is used to set the clock for the 4MW SCU ...

... if the setting is inaccurate. Use the BOS TIME command, if BOS is used, and the "clok" config card to check for inaccuracies in the clock setting.

... For further information, refer to the BOS TIME command in Section 5 and the bootload sequence in Section 5.5.

change:

Obtaining Number to Set Calendar Clock

and what follows in the clock setting section to:

Setting Calendar Clock in 4MW System Controller Unit with BOS

1. At the operator console, enter BOS (if not already in BOS). Make sure the "clok" card is loaded in the configuration deck, and always type in the time in local time as indicated on the "clok" card. Issue the TIME command to BOS.

2. Type the date and time (according to your local time zone) as follows:

   MM DD YY hh mm ss

   where:

MM is the month
DD is the day
YY is the year
hh is the hour
mm is the minute
ss is the second

When the date and time are typed, press EOM. The seconds figure can be omitted; if it is, a value of zero seconds is assumed. Choose a figure that is slightly (a minute or less) in advance of the current time, to allow time for the next step to be performed.

3. S is entered on the operator console and EOM is pressed at the instant when the current time reaches the time that was typed.

4. R is entered and EOM is pressed to read back the time to verify correctness.

5. EOM is pressed to exit from the TIME command.

Setting Calendar Clock in 6000 System Controller with BOS

1. Type: TIME as above

2. Type: MM DD YY hh mm ss as above.

3. A series of numbers in the following form is returned:

   NNNNN,NNNNNN NNNNNN TTTTTT TTTTTT MM/DD/YY HH::MM::SS.S

   where TTTTTT TTTTTT is the number to be entered in the switches on the 6000 SC maintenance panel in step 5 below.

4. At the CPU, the STEP CONTROL selector switch on the maintenance panel is placed in the MEM position.

5. At the SC (which must be in TEST mode), the number TTTTTT TTTTTT is entered in the upper row of the DATA switches. All zeroes are entered in the lower row of the DATA switches.

6. The INITIALIZE and the LOAD CLOCK pushbuttons are pressed simultaneously, at the instant when the current time reaches the time that was typed.

7. The STEP CONTROL selector switch on the CPU is turned to OFF and the STEP pushbutton is pressed.

8. R is entered and EOM is pressed to read the time from the calendar clock and verify correctness.

9. EOM is pressed to exit from the TIME command.

## Setting Calendar Clock in 4MW System Controller Unit without BOS

1. When BOS is not used, bce will automatically invoke a clock setting function after leaving the "early" bce command level. The operator must ensure that the "clok" configuration card specifies the correct time zone. All times entered are to be in local time.

2. The clock setting routine will start by asking a question of the form:

```
The current system time is DATE TIME.
Is this correct?
```

   to which the operator should respond accordingly. The operator may respond with "abort" to return to the "early" command level.

3. If the operator's answer to the above question is "no", bce will prompt with:

```
Enter time as yyyy mm dd hh mm {ss} :
```

   to which the operator should provide the current local time. The values have the same meaning as they did for the BOS time command, above. The seconds field need not be specified. Choose a figure that is slightly (a minute or less) in advance of the current time, to allow time for the next step to be performed.

4. After the time is entered, bce will re-prompt with:

```
The current system time is DATE TIME.
Is this correct?
```

   If this is not correct, the operator should respond with "no" or "abort" as above. If this is correct, the operator should answer "yes", pressing EOM at the instant when the current time reaches the time that was typed. bce will then continue with its initialization.

## Setting Calendar Clock in 6000 System Controller without BOS

1. After leaving the "early" bce command level, the bce clock setting function will be invoked.

2. bce will ask the correctness of the current time, as above.

3. The operator may reply "abort" or "yes" as above. If the operator answers "no", the time will be requested. It is entered as above. bce will then respond with:

   SCU Switches (octal) TTTTTT TTTTTT

4. bce will prompt with:

   Enter anything after the switches have been set.

   at which time the operator should perform steps 4 though 7 of the BOS instructions. When this is completed, the operator should enter "y".

5. bce will repeat the question in step 2. This should be answered appropriately.

SECTION MOH-4

# I/O DEVICE OPERATION

## USE OF THE OPERATOR CONSOLE

The operator may use the operator console to issue Multics initializer commands, commands to the daemons, standard Multics commands, commands to bce when bce is in operation and BOS commands when BOS is in operation.

SECTION MOH-5

BOOTLOAD OPERATING SYSTEM

BOOTLOAD OPERATING SYSTEM DESCRIPTION

remove the reference to initiating an emergency shutdown of Multics

Summary of BOS commands

remove ABS, BLAST, DUMP, ESD, FDUMP and PATCH

Name: BOOT

remove the command and keywords fields from the command and description of their use. Remove the BOOT STAR example from the notes.

SECTION MOH-5.5

# BOOTLOAD COMMAND ENVIRONMENT (BCE)

Add a new section describing bce after the section describing BOS as follows.

## BCE DESCRIPTION

The bootload command environment comprises a set of programs for performing functions such as the bootloading of Multics service, dumping and patching main memory and disks and initiating an emergency shutdown of Multics service.

bce is contained within the first two collections of modules on the Multics system tape; it consists of the following major parts:

1. collection zero routines
   a series of programs capable of loading the other bce programs into memory; this series is also capable of loading firmware into the bootload tape mpc, if necessary.

2. collection one initialization
   a series of programs that are part of Multics initialization proper that also initialize the bootload command environment.

3. toehold program
   a small program permanently residing in main memory at absolute location 24000 (octal). It communicates closely with bce and with service Multics to perform administrative functions.

4. bootload command utilities
   a series of programs to provide the bce command level.

5. bce command programs
   a number of programs that perform the operator directed functions of bce.

## CONFIGURATION REQUIREMENTS

bce requires the operator console; standard Multics error recovery is used, however, in case of the failure of the main console.

bce uses 512k of contiguous low order memory. All of bce's functions can be performed within this memory.

Two special regions of the rpv are used by bce. These two special regions have locations recorded in the label of the rpv. The first is the "file" partition, which contains a simple file system used by bce to hold bce exec_coms and ascii sources of configuration files. The second is the "bce" partition, used by bce to hold the following:

a saved copy of memory used by service Multics when bce is invoked upon a crash

bce itself and bce command programs

the programs needed to boot service Multics

## LOADING BCE

bce can be loaded in two ways, via BOS or via the operator's console. When booted via the operator's console (performed if BOS cannot run on the current hardware configuration), the facilities of BOS cannot be used.

### Loading bce from BOS

bce can be booted from BOS very easily by entering:

BOOT drive_number

where drive_number is the number of a tape drive on the bootload tape controller that holds a Multics system tape. The first message should read:

Booting system SYS_ID generated TIME.

This may be followed by various informative messages, depending on various parameters in the config deck. The entire Multics system tape will be read in stages. After this, bce will prompt with the ready message:

bce (boot) TIME:

You are now at the normal bce command level.

## Bootloading bce from the operator's console

bce is loaded from a Multics system tape into memory and into the bce partition as follows:

1. Mount and ready the Multics system tape on a tape drive appropriate for the density of the tape.

2. Set the tape MPC switches 5, 6, 7 and 8 to the number of the tape drive on which the system tape is mounted. If the tape MPC is not wired to be initialized when the INITIALIZE button is pressed, it must be initialized at the MPC control panel. The Honeywell field engineer can advise the operator whether or not the MPC is wired to be initialized when the INITIALIZE button is pressed (if the reset out line (RSO) is grounded, then initialization is suppressed).

3. Make sure that the CARD/TAPE switch on the IOM is set to the TAPE position and that the tape channel number is set correctly in the IOM switches.

4. At the operator console, press the RESET CONSOLE button. (This button may not be present on some console models.)

5. For all consoles except the CSU6601, press the INITIALIZE and then the BOOTLOAD button.

   For the CSU6601, after pressing the INITIALIZE button, press the RETURN key on the keyboard. Wait for the console to respond with "CONSOLE READY", and then press the BOOTLOAD button. If the system indicator panel is not present, the boot sequence from the keyboard is:

   ```
   esc ctl I return esc ctl B
   ```

   Alternatively, press the INITIALIZE button and then the BOOTLOAD button on the IOM to which is attached the tape MPC.

6. If all goes well, the message:

   ```
   Booting system SYSID generated TIME.
   ```

   will appear on the console with an alarm. This will be followed by the query:

   ```
   Enter boot tape MPC model:
   ```

This information is requested so that firmware may be loaded into this MPC. If firmware should not be loaded (or the MPC does not allow being so loaded), the operator should answer with "ipc". An answer of "shut" will stop (crash) initialization at this point. A question mark will list the valid MPC model names. Otherwise, the MPC model name should be entered. Acceptable names are:

t500
t601
t610

A message of the form:

Booting MODEL IOM CHANNEL with FWID REVISION firmware.

followed by

Booted tape MPC.

signals successful booting of the boot tape MPC.

7. bce will proceed through various initialization programs, possibly producing various status messages. The first collection will be read from the system tape into memory. After this is done, bce will request the location of the rpv:

Enter rpv data:

The operator may answer "shut" at this time to abort booting, typing "help" will provide some explanation and typing "?" will cause bce to prompt the operator for each item of information separately. Otherwise, the question should be answered as:

rpv Icc MPC_model DRIVE_model DRIVE_number

or

cold Icc MPC_model DRIVE_model DRIVE_number

where:

I
: is the IOM number containing the base channel of the MPC containing rpv

cc
: is the channel number on the IOM of the MPC (in decimal)

MPC_model
is the model of the disk mpc (in decimal). Valid models are:

191 400 451 601 603 607 609 611 612

DRIVE_model
is the model number of the drive containing rpv
DRIVE_number
is the number assigned to the drive on the MPC

"cold" is specified only if this is a "cold" boot, that is, one in which the Multics storage system is either non-existent or has been destroyed.

When a satisfactory answer is entered, the mpc described will have firmware loaded into it, if necessary. Entering "skip" or "skip_load", by itself and before entering "rpv" or "cold" will suppress this load.

If this is a cold boot, the init_vol loop (described in Section 7, Initializer Commands) will be entered. At this time, the attributes of the rpv must be entered.

8. If the previous is successful, bootload Multics will come to the "early" command level. This command level allows a subset of the normal bce commands to be entered. The ready message at this time is:

bce (early) TIME:

The purpose of this command level is to insure that the config deck (obtained from the "conf" partition on disk) is good. If this is a cold boot, the config deck will need to be entered at this time. (The commands to do all of this are described below.) Reaching the "early" command level, however, is only part of booting bce. To completely boot bce, enter "boot".

9. bce will enter its clock setting phase. (See the description of clock setting in Section 3, Configuration.)

10. Another initialization pass is then run to enable usage of the peripherals described by the config deck. The various disk mpcs so described will be tested to see if they appear to be running. If any are not, the message:

load_disk_mpcs: Disk mpcs NAMES appear not to be operating.
Enter disk mpc names to be loaded, or "none" or "abort":

The operator is to enter the names (from the set displayed as NAMES, above) of disk mpcs into which firmware is to be loaded. The operator should continue to enter names (on multiple lines, if desired), until all disk mpcs to be used are loaded. After this, "none" should be entered. If "abort" is entered, a return is made to the "early" command level.

11. Initialization will then continue until normal bce command level is reached, prompting with:

    bce (boot) TIME:

    The operator should load firmware into all other tape and unit record controllers at this time, using the bce "fwload" command. At this time, bce is fully initialized.

## Error Recovery During bce Boot

Several attempts are made to allow for error recovery during the boot process. The methods depend on the point within the boot sequence. It is best to describe the recovery by describing some aspects of the internal operation of the boot sequence.

When booted from the switches, bce will pass through collection 0 initialization, whose objective is to read in collection 1 (bce proper). A config deck is synthesized from the knowledge of the hardware found during this pass and through questions to the operator. A first pass is made through collection 1 to find the rpv and to read in the config deck last saved in the "conf" partition on disk. If an error should occur before this point (most likely a hardware or software failure), the early dump facility is invoked (see below). Otherwise, this environment (memory and the synthesized config deck) is saved on disk. The "early" command level is then entered. The operator must then make sure the config deck (read from disk) is correct. The operator then enters "boot" to actually boot bce. Initialization continues with a second pass through collection 1. If this pass fails (most likely either a hardware problem or an error in the config deck), the saved environment will be restored and the operator returned to the "early" command level. The operator then retries the boot. Eventually this will succeed and bce will come to the "boot" command level, having saved this new environment and config deck.

When bce is booted from BOS, collection 0 is still run to read in collection 1. In this case, though, the config deck need not be synthesized; the config deck used by BOS is used. The first pass through collection 1 will be the "boot" phase. If an error should occur during this first pass, the early dump facility will be invoked. BOS can be manually entered at this

time, if desired. If the pass is successful, this environment (memory and the config deck) is saved to disk. The "boot" command level is entered.

Once at the "boot" command level, the operator may perform whatever bce functions are desired. "boot" is then entered to boot Multics service. Another pass through collection 1 is made to set up for Multics service. If an error occurs during this pass (most likely hardware or a bad config deck), the environment saved above is restored and the operator is returned to the "bce_crash" command level. Also, if a bce utility should fail or should encounter a breakpoint, this environment is restored and "bce_crash" level entered. At this time, the operator may enter "crash" level commands to examine the failed image (or to debug bce), or "boot" level commands may be used to fix the config deck (if necessary) and to retry the boot of Multics service.

An important thing to remember about coming to the "bce_crash" or returning to the "early" command levels is that they use an environment and config deck declared safe on a previous initialization pass. As such, not all devices listed in the "current" config deck (the one visible with the config deck editor) may be accessible at this level. Generally speaking, to access all devices, it is necessary for the config deck to be correct and for an initialization pass (the "boot" pass) to be made. If in doubt, entering "reinitialize" will run another initialization pass.

Once the "service" pass of collection 1 completes, any further failures of initialization or of Multics itself returns to the "crash" command level, used for examining the crash. At this time, the config deck as used by Multics is used. This is done to take into account any reconfigurations performed by Multics service. At the "crash" level, a dump should be taken and an emergency shutdown performed.

## Config Deck and Device Accessibility

During Multics service, the set of devices that are accessible (to the system as a whole) are precisely those described by the config deck. The config deck is kept up to date with the state of the devices. However, the real state of devices and their accessibility is described by various control tables within Multics. One of the main purposes of bootload Multics is to set up these control tables. Since bootload Multics allows arbitrary text editing upon the config deck, it follows that the state of the control tables may not match that of the config deck. This section describes some of these subtleties.

When at "early" command level, the control tables describe only those hardware units truly known, the bootload tape drive,

the rpv, the bootload processor, etc. At the "early" command level, the operator is to make sure that the config deck describes all hardware units. These units are not accessible at this time, however.

Attempting a boot to "boot" command level builds control tables describing all of these hardware units. If this boot succeeds, all of these units are accessible from bootload Multics. If it fails, bce returns to "early" command level with only the initial hardware units accessible.

At the "boot" command level, the operator may again change the config deck. Any units added, for example, will not be accessible at this time, since the control tables do not describe them. However, if the operator boots Multics service, Multics will be able to access them all, since Multics boot will build control tables for them all. If this boot fails, bce will return to the "bce_crash" command level, with these new changes not described in the control tables (but visible in the config deck).

Any changes made to the config deck will be reflected in the control tables in only one of two ways. The first is to boot to the next command level, or to Multics service. If the config deck is correct, the devices become accessible. The other method is to enter "reinitialize" which runs a new initialization pass and returns to the "boot" command level. If this succeeds, the devices become accessible. If it fails, bce returns to "bce_crash" level, without the changes having been affected.

## BOOTLOAD MULTICS TOEHOLD

The bootload Multics toehold is a program that resides in main memory. The toehold communicates very closely with the control program in the manner described below.

When Multics is running, the toehold may be invoked by manually forcing the processor to execute an XED 24000 (octal) interrupt inhibited instruction. The CPU must be in TEST mode when the XED instruction is executed. The toehold saves the processor registers and the 512k of low memory. It then reads in a saved copy of bootload Multics from the rpv and transfers control to it. Bootload Multics then enters its command level with a prompt of:

bce (crash) TIME:

The toehold is also invoked as a result of the "go" or "continue" commands issued within bce. In this instance, the toehold restores the memory image that it had previously saved and restarts the program that was originally running.

The toehold contains a flagbox of bits that may be ON or OFF and which can be read and set both by bce and Multics.

To enter bce manually, set the processor STEP switch to MEM, enter 024000717200 (XED 24000 interrupt inhibited) in the instruction switches (data switches), set the EXECUTE switch to the EXECUTE SWITCHES position. Then, press the EXECUTE button, set the STEP switch to OFF and press the STEP button. bce is entered.

If your site has a DPS 8 system, the procedure for executing switches will be different. Refer to Appendix M, "DPS 8 Operating Procedures", for details.

## THE EARLY DUMP FACILITY

The early dump facility is a primitive facility within bce that is capable of saving an image of memory to tape upon a system failure early within initialization. It resides at a fixed location in memory whenever bce is running (30000 octal). It is invoked automatically whenever a hardware or software error is detected prior to the establishment of the bootload Multics toehold. It can also be entered manually, whenever bce is present (but definitely NOT when service Multics is running), by forcing a transfer to 30000 (octal). This is done in a manner similar to forcing a manual return to bce, except that the value entered into the data switches is 030000710200 (tra 30000 interrupt inhibited).

Once entered, the early dump facility may print a flagbox message and then prompt with:

Enter tape drive number for memory dump:

to which the operator should provide the drive number on the bootload tape controller on which a tape is mounted for writing. Memory will be dumped onto this tape at a density of 1600. After performing the dump, bce will disable itself. If bce was booted from BOS, BOS may be entered manually at this time.

The tape written by this facility can be read by the read_early_dump_tape (redt) command, described in the System Maintainer's Guide.

## BCE COMMAND LANGUAGE

The command language used within bce is the normal Multics command language (actually the ssu_ request language), not to be confused with the command language used at the Initializer's ring

1 and ring 4 command levels. (Refer to MPM AG91 - Reference for a description of Multics command/subsystem language.) Full support for active functions, iteration sets, etc. is provided.

Commands to bce are obtained from the bootload console, using standard typing conventions. It is also possible for bce commands to be placed into exec_coms. exec_coms are ascii files containing commands and possible input to commands. They are edited within bce via the "qedx" command and placed into operation with the "exec_com" command.

Also, a command may be placed in the flagbox within bce or Multics for bce to execute whenever Multics crashes or shuts down.

Whenever at bce command level, bce responds with:

bce (boot) TIME:

(or "early" or "bce_crash" or "crash", depending on the circumstances). Some commands have sub-requests to them, such as qedx and probe. The conventions for request lines entered for such commands varies from command to command.

## ABORTING BCE COMMANDS

Whenever the REQUEST button is pushed on the console (or the RETURN key on the CSU6601) when such a request was not solicited by bce, the bce abort routine is entered. This routine allows bce operations to be aborted to various extents. When called, the abort function prompts (on the console) with:

Abort?

to which various answers may be given. If the REQEUST button was hit accidentally, the operator may enter "no" or "n" to return to the interrupted operation. Answering "yes" or "y" aborts the immediate operation. If this operation was a sub-request, only this sub-request is aborted. Otherwise, the command in question is aborted, returning either to the exec_com which called it, if one was present, or to bce command level. Answering "r", "req" or "request" is equivalent to "yes". An answer of "command", "com" or "c" aborts the current command, regardless of whether a sub-request was in execution or not. Finally, an answer of "all" or "a" aborts anything in execution, returning to bce command level.

## BCE COMMANDS

The current set of bce commands and active functions is listed below. Various commands are valid only at certain command levels; the valid levels for each command is provided in the description of the command.

bce also includes most of the standard active functions; in particular, the standard arithmetic, character, boolean and comparison active functions are included. The current list includes:

after, af
before, be
ceil
collate9
date_time_after, dtaf
date_time_equal, dteq
decat
equal
greater
high9
length, ln
low
ltrim
min
mod
ngreater
not
plus
quotient
reverse
reverse_before, rvbe
reverse_index, rvindex
reverse_verify, rvverify
search, srh
time_after, taf
time_equal, teq
trunc
verify
and
bool
collate
copy_characters, cpch
date_time_before, dtbe
date_time_valid, dtv
divide
floor
high
index
less
lower_case, lowercase
max
minus
nequal
nless
or
query
response
reverse_after, rvaf
reverse_decat, rvdecat
reverse_search, rvsrh
rtrim
substr
time_before, tbe
times
upper_case, uppercase

### Summary of bce commands

alert
: Write an alert message on the console.

boot
: Boot Multics.

bos
Return to bos, if present.

config_edit, config
Enter the config deck editor.

continue, go
Restart the interrupted Multics image.

delete, dl
Delete a bootload file.

die
Abort bce.

dump
Create a dump of Multics in the dump partition.

emergency_shutdown, esd
Perform an emergency shutdown of Multics.

exec_com, ec
Execute a file of bootload Multics commands.

fwload, fw
Load firmware into an mpc.

get_flagbox, gfb
Get the value of a flagbox variable.

init_files
Initialize the bootload file system.

list, ls
List bootload files.

list_requests, lr
List bootload requests.

print, pr
Print a bootload file.

probe, pb
Examine/modify the Multics image.

qedx, qx
Edit bootload text file.

reinitialize
Re-perform Multics initialization.

rename, rn
Rename a bootload file.

set_flagbox, sfb
: Set the value of a flagbox variable.

severity
: Returns the severity of a bce request.

shutdown_state, sds
: Returns the shutdown state of the storage system.

Name: alert

The bce alert command writes a message on the operator console with an audible alarm. This is useful in auto exec_coms to inform the operator that the system has crashed. This command is valid at all bce command levels.

Usage

```
alert The system has crashed!!!
```

Name: boot

The bce boot command causes the next phase of initialization to proceed. If bce is at the "early" command level, this causes a boot of bce itself (bce passes to its "boot" state where it is fully initialized). If bce is at the "boot" or "bce_crash" command levels, this will boot Multics service. It is not valid at the "crash" command level. The command can also supply certain parameters that will apply to the bootload of Multics.

Usage

boot {command} {keywords} {cold}

where:

1. command is one of the following ring 1 command abbreviations:

| | |
|---|---|
| star | startup |
| mult | multics |
| salv | salvage_dirs |
| stan | standard |

2. keywords can be one or more of the following:

nodt
recreates the disk table; renames and ignores the existing one.

nolv
recreates the logical volume registration directory (>lv); renames and ignores the existing one.

rlvs
performs a volume salvage of the rpv (root physical volume), a directory salvage of all directories used in initialization and a volume salvage of all other member volumes of the rlv (root logical volume).

rpvs
performs a volume salvage of the rpv and a directory salvage of all directories used in initialization.

3. cold
specifies that the root dir is to be re-created, thus destroying the old file system hierarchy. This option should only be used when a cold boot of bce was also performed. The operator will be queried as to whether bce should continue.

Name: bos

The bce bos command causes bce to return to BOS, if bce was booted from BOS. BOS may return to bce with the use of the BOS "CONTIN" or "GO" commands. This command is valid at all bce command levels.

Usage

bos

Name: config_edit, config

The bce config command enters the config deck editor. This editor is identical in function to the qedx text editor, except that buffer 0 contains an ascii source form of the config deck. This command is not valid at the "crash" command level.

Usage

```
config_edit {file_name}
```

Notes

If a file_name is supplied on the command line, the specified file is read into the config deck without entering the config deck editor.

If not supplied a file_name, upon entry, the current config deck (that found in the "conf" partition on the rpv) is read into buffer 0. It is converted to a labeled ascii form which is an expanded form of that used in the configuration card description section. Arbitrary text editing operations may be performed upon this buffer, as well as any other. Performing a "w" (write) request upon buffer 0 writes the edited buffer back into the config deck.

In the labeled form, each field, except for the card name, may be optionally preceded by a label. Labeled fields may appear in any order. The interpretation of a card in labeled form is that all labeled fields are placed into their proper places; any unlabeled fields then fill in the missing spaces. Thus,

```
iom -state on -port 1 a nsa
```

becomes

```
iom a 1 nsa on
```

in its standard form.

The various labeled forms appear in Section 6 (Configuration Description). If a card is to be entered whose format has been locally changed or of a otherwise unknown format or type, a "." may be placed in front of the card name to avoid errors during parsing of the card. Such a card may not have any labeled fields.

The operator should keep in mind the discussion in "Config Deck and Device Accessibility", above for details on the implications of this command.

Name: continue, go

When Multics is interrupted as the result of a manual return to bce or as the result of encountering a bce probe breakpoint, the machine image is saved. The bce continue command restores the machine image and continues running the interrupted activity (usually Multics). This command is valid at the "bce_crash" and "crash" command levels.

Usage

continue

Name: delete, dl

The bce delete command deletes files within the bce file system (not the Multics storage system). The star convention is allowed. This command is valid at all bce command levels.

Usage

```
delete star_name {... star_names}
```

Name: die

The bce die command aborts all bce activities. It wipes out the bce toehold, preventing any returns to bce, manual or otherwise. It should be used only when it is desired to absolutely kill off any remnants of bce. This command is valid at all bce command levels.

Usage

die

Note

The die command queries the operator as to whether bce should really be killed off. This query may be avoided by using the "-force" ("-fc") control argument.

Name: dump

The bce dump command produces a diagnostic dump of system memory and tables after a hardware or software failure, for later analysis. The dump is produced by copying binary images of segments and directories into the dump partition of the disk described by the part dump config card. Arguments to the dump command specify which processes are to be examined and which segments from these processes are to be dumped. (See "Notes" for a general purpose command line.) This command is valid at all bce command levels.

Usage

```
dump {macro_keyword}
     {-process_group segment_option {...segment_options}}
     {-force | -fc} {-dump #} {-crash} {-bce}
```

where:

1. macro_keyword
   specifies one of the following default group of processes and segments to dump.

   ```
   -brief, -bf     is equivalent to -run hc pp dir
   -short          is equivalent to -run hc pp dir -elig hc
   -long, -lg      is equivalent to -all wrt
   ```

2. process_group
   specifies a group of processes to be considered for dumping. The segments that get dumped for processes in this group are specified by segment options that follow the process group keyword. Allowed groups are:

   -running, -run
   processes running on a processor (apte.state = running or stopped)

   -initializer, -inzr
   the initializer process (first apte entry)

   -eligible, -elig
   all running and eligible processes (processes being considered for running)

   -all
   all processes

3. segment option
   specifies a class of segments to be dumped for the group of processes specified by the process group keyword. Segment classes are:

directories, dir
directory segments (aste.dirsw = "1"b)

hardcore, hc
the pds, kst, dseg and ring 0 stack for the process(es). If a process is running, this also dumps the prds for the processor in question.

per_process, pp
the segments contained within the process directory of the process(es) (aste.per_process = "1"b)

stacks, stk
all stack segments in the process(es) not already dumped by the hc or pp keywords.

writeable, wrt
all segments to which the process(es) have write access. This keyword produces a very large dump.

Writable ring zero segments (system data bases) other than directories are dumped regardless of what keywords are specified.

Prefixing a segment option with a circumflex (^) reverts an earlier occurence of the given segment option. Thus, one can turn use a macro_keyword and turn off a specific segment option within it.

4. crash or bce
specifies what bce should dump. The default is to dump the saved Multics image. A dump of bce itself (the dumper) can be made by specifying -bce.

Notes

For general purpose dump analysis, the command line:

dump -run hc pp dir -elig hc stk -inzr hc stk

should give the user all of the useful processes and segments (to produce a smaller dump, remove the "dir" keyword). For simplicity and to remove the possibility of operator error, this command line should be placed into a bce exec_com, either by itself or in a site supplied crash exec_com.

The dump command examines the active process table entries (apte) within the specified image. For each entry, the criterion specified through the keywords is used to decide if any segments from this process are to be dumped. If any segments are to be dumped, the segment options are applied to each segment active within that process to decide whether or not they should be

dumped. As each process is dumped, dump will produce an output line showing the apte number and the dbr value for the process. After scanning all apte entries, if the process in control when Multics crashed was not one of the processes dumped, it is dumped with a status line showing an apte number of zero. This process is dumped with the running and initializer segment options.

Within the dump partition is kept a counter and a valid flag. When a dump is placed into the partition, the valid flag is set. It is reset when the dump is copied out during Multics service (by the copy_dump exec command). If the dump in the partition has not been copied, dump will query the operator if it should be overwritten. This query can be avoided by specifying the "-force" ("-fc") control argument to the dump command.

Dumps are assigned dump numbers sequentially by default. The dump number may be changed to a desired value with the -dump control argument.

The dump command provides a severity indicator, indicating the successful of its operation. This indicator may be obtained with the severity command/active function. The interpretation of the severity status is:

0 - dump was never called.
1 - dump was entered but never completed.
2 - dump was aborted because the partition contained an old dump.
3 - the dump was successfully generated.

Name: emergency_shutdown, esd

The bce esd command starts an emergency shutdown of Multics. It is only valid at the "crash" command level. It should be used whenever the system crashes to prevent storage system damage. Performing an emergency shutdown destroys the saved crash image and should therefore only be done after a dump is taken.

Usage

esd

Name: exec_com, ec

The bce exec_com command invokes a bce exec_com. An exec_com is an ascii file consisting of a series of commands to invoke. bce uses exec_com version 1, described in AG92 (Commands and Active Functions). This command is valid at all bce command levels. This may also be used as an active function, as described in AG92.

Usage

```
exec_com ec_name {ec_arguments}
```

Name: fwload, fw

The bce fwload command loads firmware into the specified mpcs. It scans the config deck to determine the location of the mpcs and the type of peripherals involved to determine the firmware and overlays needed. This command is not valid at the bce "early" command level.

Usage

```
fwload mpc_name {...  mpc_names}
```

Name: get_flagbox, gfb

The bce get_flagbox command is used to determine the values of various variables maintained in the bce flagbox. These variables are also accessible from Multics service and therefore allow a small method of communication between bce and Multics service. This command is valid at all bce command levels. It also works as an active function.

Usage

gfb flagbox_variable

where flagbox_variable is one of the following:

N
where N is from 1 to 36. The returned value is the Nth flagbox flag. These flags have true or false values. Some of them are named and can be refered to by their names, as listed below.

auto_reboot
(also flag 1) Used by the auto bce exec_com. Refer to Appendix I (Continuous Operation Exec_coms) for more details.

booting
(also flag 2) Used by the auto bce exec_com.

rebooted
(also flag 4) Used by the auto bce exec_com.

unattended
(also flag 5) Used by the auto bce exec_com.

bce_command
a command that is invoked by bce whenever it reaches a command level. The result is a character string, quoted. This command may be set so that bce can be set to automatically boot Multics upon a crash, etc. Refer to Appendix I for more details.

ssenb
a flag set by Multics service indicated whether or not the storage system was enabled at the time of a crash. A value of true indicates that an emergency shutdown needs to be performed (or did not succeed).

call_bce
indicates that bce was called through a program calling call_bce. This may be the result of the operator having entering the bce command.

shut
indicates that Multics successfully shutdown. If neither shut nor call_bce is set, Multics either encountered a breakpoint, crashed or was manually brought to bce.

manual_crash
indicates that bce was invoked manually, either by the operator manually forcing a return to bce (XED 24000) or by hitting the EXECUTE FAULT button.

Name: init_files

The bce init_files command wipes out all files in the bce file system. It is to be used only if a problem is encountered with the bce file system. This command is valid at all bce command levels.

Usage

```
init_files
```

Note

init_files will query the operator as to whether the bce file system is to be cleared. This query may be avoided by using the "-force" ("-fc") control argument.

Name: list, ls

The bce list command lists the names of bce files matching a set of star names. It is valid at all bce command levels. It can also be used as an active function to return the set of names. When used as a command, providing no star names will list the names of all bce files.

Usage

```
list {star_names}
```

Name: list_requests, lr

The bce list_requests request lists all requests valid at the current command level.

Usage

list_requests

Name: print, pr

The bce print command prints the contents of a file in the bce file system. This command is valid at all bce command levels.

Usage

pr file_name

Name: probe, pb

The bce probe command is used to examine, patch and generally debug Multics hardcore, bce itself as well as providing a general memory and disk patch/dump facility. Its requests have a fair resemblance to those of Multics probe. It can be used at all bce command levels.

Usage

pb {control_arguments}

where valid control arguments are:

-bce to examine bce itself

-crash to examine the saved crash image

-break to examine the active breakpoint

The default, when invoked at the "boot" command level is to examine bce, when invoked automatically upon encountering a breakpoint is to examine the breakpoint and otherwise is to examine the crash image.

Notes

bce probe reads request lines from the bootload console. Multiple requests may appear on one line separated by semi-colons. The syntax of these requests varies from request to request. The recognized requests are listed below. Various other aspects of bce probe are described in the following sections.

ADDRESSES

Several requests in bce probe take an address describing what should be displayed, modified, etc. These addresses can take many forms, depending on what is desired. Valid address forms are:

N
: specifies absolute memory location N. N may describe any location in all of memory. N is specified in octal.

M|N
: specifies the virtual location N in segment M. The interpretation of this virtual address depends on the

address space being examined; refer to the "dbr" and "proc" requests. Both N and M are octal values.

NAME|N
specifies the virtual location N in the hardcore segment with the specified NAME. This interpretation is also subject to the address space being examined. N is specified in octal.

.{+|-N}
specifies the last location referenced (of any address type) optionally offset by the value N. N is an octal value.

reg(NAME)
specifies the named register in the crash image. This address is not valid when examining the live bce. Valid registers are:

prN (N = 0 to 7)
xN (N = 0 to 7)
a, q, e
t, ralr
fault, ext_fault, mode, cache
dbr, bar

disk(DRIVE_NAME,RECORD_NUM,OFFSET)
refers to a specific page of a disk drive. The drive name is in the standard form, dska_04, for example. Both RECORD_NUM and OFFSET (within the page) are octal values.

## PROBE REQUESTS

before, b {ADDRESS}
sets a breakpoint before the specified address. If no address is specified, "." is assumed. Refer to Appendix O for more details about hardcore breakpoints.

continue, c
continues the saved image. It is the same as exiting probe and entering "continue".

dbr VALUE1 {VALUE2}
sets the dbr (descriptor base register) value used in the appending simulation used to access virtual addresses in the Multics image. If VALUE2 is omitted, the second word of the dbr value is obtained from the dbr in effect when Multics crashed. Both VALUE1 and VALUE2 are octal values.

display, ds ADDRESS {MODE {LENGTH}}
displays a set of locations in a specified mode. If LENGTH is omitted, a value of 1 is assumed. For virtual addresses, a LENGTH of "*" may be specified to display to the end of the segment. If MODE is omitted, octal is assumed. Valid modes are:

a - ascii characters d - decimal words i - instruction format o - octal words (default) p - symbolic pointer (double words)

The locations are displayed four to a line in the desired format. The value of "." after this request finishes is the first location displayed.

let, l ADDRESS = VALUE {... VALUE}
modifies a series of locations starting at the address specified. Each value is converted to a number of words and catenated together to form the new value. Valid values are:

"STRING"
a quoted string of characters. To place a quote character into the string, it must be doubled.

N
a decimal number

No
an octal number

Nb
a binary number

M|N
a pointer to segment M offset N (double word)

NAME|N
a pointer to the named hardcore segment offset N (double word)

list_requests, lr
lists the valid bce probe requests.

mc ADDRESS {-long | -lg}
displays, in interpreted form, the scu data found within the machine conditions at the specified address. Specifying "-long" also dumps the machine registers from the machine conditions.

name SEGNO
displays the name of the hardcore segment with segment number SEGNO.

proc N
changes the address space used by the appending simulation for displaying virtual addresses to the Nth process in the active process table. A value of 1 specifies the Initializer's process.

quit, q
exits probe.

reset, r {ADDRESS}
resets the breakpoint at the specified address. If address is not specified, the currently active breakpoint (if so existing) is reset. Refer to Appendix N for more details of hardcore breakpoints.

segno NAME
displays the segment number of the named hardcore segment.

stack, sk ADDRESS
displays a stack trace starting at the given address. If the word offset of the address is 0, the address is assumed to refer to a stack header. Otherwise it is assumed to refer to a stack frame. For each frame, the stack frame offset, entry pointer, return pointer and argument pointer is displayed.

status, st {SEGNO|NAME}
displays a list of breakpoints set. If no argument is supplied, all segments with breakpoints set are displayed. If a SEGNO or NAME (of a hardcore segment) is provided, then all breakpoints within that segment are displayed.

Name: qedx, qx

The bce qedx command invokes the qedx text editor to edit a bce file system file. All requests of the standard Multics qedx editor are supported except for the "e" request. This command is valid at all bce command levels.

Usage: qedx {-control_args} {macro_file} {macro_args}

Notes

Refer to the description of the qedx command in AG92 (Commands and Active Functions).

Name: reinitialize

The bce reinitialize command causes bce to perform a new initialization pass, thereby reflecting any changes to the config deck made since the last such pass. This command returns the operator to "boot" command level. It is valid at the "boot", "bce_crash" and "crash" command levels. When used at the "crash" command level, the operator is asked whether to continue, thereby destroying the saved Multics image. This query may be avoided by using the "-force" ("-fc") control argument.

Name: rename, rn

The bce rename command renames files in the bce file system. The star and equal conventions are used. This command is valid at all bce command levels.

Usage

```
rename STAR_NAME EQUAL_NAME {...  STAR_NAME EQUAL_NAME}
```

Name: set_flagbox, sfb

The bce set_flagbox command changes the values of various flagbox variables. When used as an active function, it also returns the previous value of the variable. It is valid at all bce command levels.

Usage

set_flagbox VARIABLE VALUE

where

VARIABLE
is a valid flagbox variable, as listed above under get_flagbox.

VALUE
is either a character string (for the bce_command variable) or the string "true" or "false" for other flagbox variables.

Name: severity

The bce severity command returns the severity, or extent of completion, of a preceding bce command. This command is valid at all bce command levels. Currently, the dump command provides such a severity status. Future bce commands may also. This command may also be used as an active function.

Usage

```
severity PROG_NAME
```

Name: shutdown_state, sds

The bce shutdown_state command returns the state of completion fo the shutdown of Multics service. It does this by examining the shutdown_state flag in the label of the rpv. This request is valid at all bce command levels. It may also be invoked as an active function.

Usage

shutdown_state

Notes

The interpretation of the shutdown states follows.

0 - Normal Multics shutdown (no esd)
1 - esd part 1 started (memory flush of modified pages of segments)
2 - esd part 1 completed
3 - shutdown or esd completed with lock errors
4 - shutdown or esd completed with no errors
other - shutdown completed with errors, or not completed for one or more disk errors

SECTION MOH-6

MULTICS CONFIGURATION DESCRIPTION

Besides changing config cards to appear in lower case, most config cards now have a labeled form. Add to each applicable config card description a new entry labeled:

Labeled format

with the appearance listed below.

Name: chnl

Labeled format

```
chnl -subsys device_name  -iom iom1 -chn  chn1 -nchan nchan1
     {...  -iom iom4 -chn chn4 -nchan nchan4}
```

Name: clok

Labeled format

```
clok -delta delta -zone zone -boot_delta boot_delta
```

Name: cpu

Labeled format

```
cpu -tag tag -port port -state state -type type -model model
     {-cache cache_size -exp_port exp_port}
```

Name: iom

Labeled format

```
iom -tag tag -port port -model model -state state
```

Name: mem

Labeled format

```
     mem -port port -size size -state state

Name: mpc

Labeled format

     mpc -ctlr  ctlr_name -model  ctlr_model -iom  iom1 -chn chn1
          -nchan nchan1 {...  -iom iom4 -chn chn4 -nchan nchan4}

Name: part

Labeled format

     part -part partname -subsys subsystem -drive drive

Name: prph

Labeled format

     prph -device ccuN -iom iom# -chn channel# -model model#

     prph -subsys  dskN  -iom  iom#  -chn  channel#  -nchan nchan
          -model  model1   -number  d1  {-model   model2  -number
          d2...-model model5 -number d5}

     prph -device fnpN -iom iom# -chn channel# -state state

     prph -device opcN -iom iom#  -chn channel# -model model# -ll
          line_length -state state

     prph -device  prtN  -iom  iom# -chn  channel#  -model model#
          -train train# -ll line_length

     prph -device punN -iom iom# -chn channel# -model model#

     prph -device rdrN -iom iom# -chn channel# -model model#

     prph -subsys  tapN  -iom  iom#  -chn  channel#  -nchan nchan
          -model  model1   -number  d1  {-model   model2  -number
          d2...-model model5 -number d5}

Name: root

Labeled format

     root -subsys   subsystem1   -drive   drive1   {...   -subsys
          subsystemN -drive driveN}

Name: schd

Labeled format
```

```
schd -wsf wsf  -tefirst tefirst -telast  telast -timax timax
     {-mine mine {-maxe maxe {-maxmaxe maxmaxe}}}
```

Name: sst

Labeled format

```
sst -4k sst1 -16k sst2 -64k sst3 -256k sst4
```

Name: tcd

Labeled format

```
tcd -apt apt -itt itt
```

Name: udsk

Labeled format

```
udsk -subsys subsystem  -nchan nchan {-drive  drive1 -number
     count1 ...  -drive drive6 -number count6}
```

SECTION MOH-7

INITIALIZER COMMANDS

Overall System Control

change the BOS command to bce

COMMAND DESCRIPTIONS

delete the BOS command and replace it with:

Name: bce

The bce command causes bce to be entered. All Multics operation is suspended. When the system is in trouble, it is sometimes necessary to enter bce to use the dump or probe commands. This command may be issued in ring 1 or ring 4.

Usage

bce

causes the system to enter bce. Type:

! go

on the bootload console to cause Multics to be restarted.

(The notes pertaining to the BOS command also pertain to the bce command.)

Name: cripple

replace references to BOS with bce.

Name: init_vol

add to the description of the default partitions the partitions "file" of length 255 and "bce" of length 2200, both added to the high end of the disk.

Name: message

The message command invokes the Multics qedx editor to edit the file message_of_the_day, which most (but not all) users print out automatically when they log in.

Usage

message

to edit the message. Editing requests may then be entered. Usage of qedx is described in MPM Commands and Active Functions, Order No. AG92.

SECTION MOH-8

SYSTEM STARTUP AND SHUTDOWN

Replace the entire section "Overview of System Startup) with the following. Keep the sub-section entitled "Bringing up Multics (Step by Step)" but change its title to "Bringing up Multics from BOS (Step by Step)".

## OVERVIEW OF SYSTEM STARTUP

There are several steps to bringing up Multics service:

- Configure the system. The drive rpv must be mounted.
- Bootload BOS. This step is optional.
- Mount the RLV (if not already mounted) and all logical volumes required at the site for starting.
- Boot bce from the current Multics system tape.
- Boot service Multics from bce.
- Start up the answering service and log in the daemons to perform backup, input/output, and any other specialized procedures (such as network interaction).

### Bootloading

A BOS bootload is the process of loading the programs that make up the essential parts of BOS. BOS is used to bootload bce. See "Loading BOS" in Section 5 for details on bootloading BOS. It is not necessary to bootload BOS to bootload bce. However, if BOS is not bootloaded, the functions of BOS may not be used in conjunction with bce.

A bce/Multics bootload is the process of loading in the programs that make up the Bootload Command Environment, who in turn build up from themselves the Multics operating system. The bootloading process loads the programs into memory, links them so that they may refer to one another, and sets up any necessary data bases. Whenever BOS is booted, bce must be re-booted. Any number of service Multics boots may be made from a single bce boot.

The programs on a Multics system tape are divide into several collections. The first program on the tape, imbedded within the tape label, is called bootload_tape_label. It reads in the next set of programs, collection 0. Collection 0 is responsible for reading in collection 0.5, used to boot firmware into the bootload tape controller. Collection 0 then reads in collection 1 and links the programs therein. This operation allows programs written in PL/I to be used. Collection 1 contains the necessary programs to enable paging, as well as to start up bce. Collection 1 uses collections 1.2, which contains canned bce exec_coms and files and collection 1.5, which contains some of the bce programs. bce also reads collections 2 and 3, needed to bootload Multics service, onto disk.

When bce is finished, collection 2 is run to initialize and set up the Multics storage system and the environment to do reloads and other system startup activities. These programs (the reloader) are found in collection 3.

## Bringing up Multics with BOS (Step by Step)

(was Bringing Up Multics (Step by Step))

- Bootload bce by typing:

  BOOT N

  (where N is the tape drive on which the Multics tape [MST] is mounted).

- When bce responds with:

  bce (boot) TIME:

  bootload Multics service by typing:

  boot star

  or by invoking the continuous operation exec_com:

  ec auto star

## Bringing up Multics without BOS (Step by Step)

To bring up Multics, proceed as follows:

- If not already in bce:

  Configure the system hardware (see Section 3).

  Mount the Multics system tape on a convenient tape drive. Set the switches on the tape MPC to indicate the tape drive, as described in "Bootloading bce from the Operator's Console" in Section 5.5.

  Boot bce from tape as described in Section 5.5.

- bce types the ready message:

  bce (early) TIME:

  This time may not be correct since the time zone is not necessarily known at this time.

- Make sure the config deck is correct. Enter or modify it if necessary.

- Enter

  ```
  boot
  ```

  to proceed to bce "boot" command level. This step will check the validity of the clock. If the clock is not valid, follow the steps described in Section 3 (Calendar Clock). This step also loads firmware into all disk mpcs (except for the bootload disk mpc).

- Load firmware into all controllers, except for disk mpcs and the bootload tape controllers (who have already been loaded in the bce bootload sequence).

- Mount all required disk volumes.

- Press the INITIALIZE AND CLEAR pushbutton on the processor configuration panel for all CPUs except the one on which you are running.

- Bootload Multics service by typing:

  ```
  boot star
  ```

  or by invoking the continuous operation exec_com:

  ```
  ec auto star
  ```

- After Multics types an introductory message and requests a command, press the REQUEST button at the operator console.

- Operate the initializer process as outlined in the following paragraphs.

Administrative Ring Commands

...If a ring 1 command was specified in the boot command, this command is executed automatically. For example, typing:

```
boot star
```

executes the star (startup) command in ring 1....

SAMPLE STARTUP SEQUENCE

```
! ec auto star
```

UNATTENDED AND AUTOMATIC MODES

Change all references to the BOOT command to refer to the boot command and all references to the AUTO runcom to refer to the auto exec_com.

SECTION MOH-10

# RECOVERY FROM SYSTEM FAILURE

## SYMPTOMS OF SYSTEM FAILURE

o The system returns to bce without operator intervention.

### Returning to bce

If the system loops or crashes and does not return to bce, the operator presses the EXECUTE button to force the system to return to bce. This can be done in one of two ways.

1. Usually bce is entered by causing an execute fault. For this to occur, the EXECUTE SWITCHES switch must be down when the EXECUTE button is pushed. (This switch may have been left in the up position.) DO NOT put the processor in STEP when using the execute fault. An execute fault must be used to force a return to bce when two or more cpus are in use.

2. bce can also be entered by executing the instruction pair located at octal location 24000 in memory. In this case, the 24000 is set in switches 0-17 and an interrupt-inhibited execute double (XED) instruction is set in switches 18-35 (the octal value of the switches should be 024000717200). If the EXECUTE SWITCHES switch is in the up position when the EXECUTE button is pressed, the XED is executed; the processor should be in STEP when the button is pushed. bce should be entered with an XED only in cases noted below, or as a last resort. This method does not stop any additional CPUs. If, as a last resort, this method is used when more than one CPU is running, all CPUs other than the bootload CPU should be put into STEP, and the forced execution performed on the bootload CPU.

If your site has a DPS 8 system, the procedures for executing switches and placing a CPU in step will be different. Refer to Appendix M, "DPS 8 Operating Procedures", for details. (The procedure for executing fault will be the same.)

IOM Alarm

change all references to BOS to bce.

Recovery after a System Failure

change all references to BOS to bce.

Change as follows:

...There are 36 switches set up in the bce toehold for intercommunication between Multics and bce. These switches are set either by the bce set_flagbox command or the Multics privileged command, set_flagbox (described in Appendix I). One of these switches means "automatic reboot mode is on". When the system is running in automatic mode and returns to bce, the flagbox bce_command variable is set to a command that tests the "crashed" indicators to discover whether the system failed or shut down normally. If the test indicates a system failure, automatic recovery procedures begin. These procedures are described under "Recovery Procedures" below.

RECOVERY PROCEDURES

Automatic recovery procedures do the following:

- Take a dump (using the bce dump command). (See section 5.5 for description of this command.)
- Perform emergency shutdown (esd).
- Bring up the system again (boot). Required salvaging is done automatically as the system is brought up....

RECOVERY FAILURE

change:

- System loop or failure to return to bce.
  In this case, the operator enters bce via XED 24000.

bce senses this manual intervention and does not perform the automatic operation specified in the flagbox bce_command. The operator may invoke automatic recovery by entering:

```
ec rtb
```

- The system_start_up.ec never finished.
  In this case, the booting flag is still on. The exec_coms take a dump and do an emergency shutdown, but abort automatic mode.

- The auto_reboot flag is off.
  Automatic mode may be turned off by the set_flagbox command executed while Multics is running. As such, the exec_coms print a message and exit after recovering.

- Some disk volume cannot be accepted.
  In this case, the initializer process has typed a message and inhibited automatic startup. The system waits at operator command level in ring 1 or ring 4, depending on where the error is detected.

- dump failed.
  In this case, the operator may choose to try again (type "ec rtb") or to try an emergency shutdown. If the dump failed because the previous copy_dump was not successful (or not reached), and if the dump partition is still full, the dump partition may be saved on tape by BOS. This allows the new dump to be taken without losing the old one. See "Saving the DUMP Partition" later in this section.

- Explicit call to bce.
  If bce is entered as a result of a call to hphcs_$call_bce, the system assumes this is due to operator intervention. The exec_coms print a message and await console input. The operator is queried as to whether automatic recovery should be performed.

- Lock error during shutdown.
  If errors are encountered during an attempted shutdown, the exec_coms print a message and await console input.

- Reboot loop.
  If the system attempts to reboot itself repeatedly, this may be a sign of some system problem that does not prevent answering service startup but crashes the system later. The standard system_start_up.ec does not reboot the system twice without operator intervention, because automatic mode gets turned off. If this plan seems to be too conservative for certain installations,

the system_start_up.ec can be modified to take other action.

Saving the Dump Partition

add a line specifying that bce must first return to BOS before BOS can perform the dump. Also, BOS must perform a "go" to restart bce.

Failures that do not crash

remove the reference to the BLAST command. Also change references to BOS to bce.

Change subsequent references to BOS to bce within the chapter except for references to the BOS save and restore commands. Make all bce command names lowercase.

SECTION MOH-12

STORAGE SYSTEM MAINTENANCE OPERATIONS

## HOW TO MOVE A PACK

change:

...If any BOS runcoms or bce exec_coms name specific drives...

...Load bce and Multics using BOOT.

# SECTION MOH-A

## SUMMARY OF OPERATOR COMMANDS

...Commands used within the bootload operating system (BOS) or the bootload command environment (bce) are not included in this list; for these see Section 5, "Bootload Operating System", Section 5.5, "Bootload Command Environment", and the summaries in Appendices C and N.

In the summary, change the bos command to be named bce and to refer to bce.

SECTION MOH-B

SUMMARY OF INITIALIZER COMMANDS

Change the bos command to be named bce and change references to BOS to bce.

SECTION MOH-C

SUMMARY OF BOS COMMANDS

Delete the ABS, BLAST, DUMP, ESD, FDUMP and PATCH commands.

SECTION MOH-H

OPERATOR'S STARTUP CHECKLIST OF SWITCH SETTINGS

PROCESSOR UNIT (MAINTENANCE PANEL) SWITCHES

DATA SWITCHES Set to XED - Location 24000 (024000 717200)

SECTION MOH-I

# CONTINUOUS OPERATION EXEC_COMS

This appendix describes the bce exec_coms supplied with the system to implement automatic recovery after system crashes. The operator usually types only the two command lines:

ec auto star
: to initiate system bootload, with automatic restart if a crash occurs.

ec go
: to restart automatic operation after a manual return to bce.

Descriptions of the get_flagbox and set_flagbox commands are included at the end of this section.

## FLAG USAGE

Several flags and indicators coordinate the bce and Multics modes of operation. The bce and Multics get_flagbox and set_flagbox commands are used to examine and set, respectively, flags in the toehold. Four flags have preassigned meanings and are known by keywords in these commands:

1. auto_reboot
   TRUE if the system is to attempt to reboot itself after it has crashed.

2. booting
   TRUE during bootload. It is turned off at the end of part 3 of system_start_up.ec, when bootload is over. This flag prevents the system from looping to reboot if it crashes before coming up.

3. rebooted
   TRUE if the system has rebooted as a result of automatic operation.

4. unattended
   TRUE if the system is not attended by an operator.

In addition, the "call_bce" and "shut" flags may be examined to determine the mode of bce entry. The "ssenb" flag may also be tested to see if the storage system has been enabled.

## EXEC COMS

auto.ec starts automatic operation.

```
&command_line off
&- automatic reboot ec for bce
&- Keith Loepere, January 1984.
&-
&print Begin auto boot.
set_flagbox bce_command ""
set_flagbox auto_reboot true
set_flagbox booting true
&input_line off
&attach
config_edit
gp/^cpu/
gp/^mem/
q
&detach
set_flagbox bce_command "exec_com rtb"
boot &rf1
&quit
```

rtb.ec
determines what operations to perform upon a return to bce.

```
&command_line off
&- ec to handle returning to bce
&- Keith Loepere, January 1984.
&-
&if [not [get_flagbox call_bce]] &then &goto non_call_entry
&-
&print bce invoked via hphcs_$call_bce.
&-
&if [not [query "Should normal recovery procedures be used?"]]
&then &goto abort_auto_mode
&-
&label non_call_entry
&-
&- look at the state of things
&-
&if [not [get_flagbox ssenb]] &then &goto ss_not_enabled
&-
&- storage system enabled; take a dump and esd
```

```
&-
exec_com dump
&-
&if [nequal [severity dump] 3] &then &goto dump_okay
&-
&print Dump failed.
&quit
&-
&label dump_okay
&-
emergency_shutdown
&- return from above is back at rtb
&-
&label ss_not_enabled
&-
&- Is everything okay?
&-
&if [nequal [shutdown_state] 4] &then &goto okay_shutdown
&-
&if [nequal [shutdown_state] 3]
&then &print Shutdown with locks set.
&else &print Error during shutdown.
&goto abort_auto_mode
&-
&label okay_shutdown
&-
&- normal shutdown - see if we should reboot
&-
&if [not [get_flagbox auto_reboot]] &then &quit
&if [get_flagbox booting] &then &goto system_cant_boot
&-
set_flagbox rebooted true
&-
&- inform a.s. that we are doing an automatic reboot
&-
exec_com auto star
&quit
&-
&label system_cant_boot
&-
&print System crashed during boot.
&-
&label abort_auto_mode
&-
set_flagbox bce_command ""
set_flagbox auto_reboot false
set_flagbox rebooted false
&quit
```

dump.ec
performs a standard dump.

```
&command_line off
&- standard bce dump defaults
&- Keith Loepere, January 1984.
&-
dump -run hc pp dir -elig hc stk -inzr hc stk
&quit
```

go.ec

restarts automatic operation after a manual return to bce.

```
&command_line off
&-
&- restart auto operation after manual bce entry
&- Keith Loepere, January 1984.
&-
set_flagbox auto_reboot true
set_flagbox rebooted false
set_flagbox booting false
go
&quit
```

SECTION MOH-M

DPS 8 OPERATING PROCEDURES

Change all references to the BOS whatever to the bootload whatever.

EXECUTING SWITCHES

2. Having typed "VIP", you will receive the CPU CMD prompt. When you see this, type "CO DATA 024000717200" followed by "EX2".

3. When the system has returned to bce, you'll see the bce ready message displayed on the system bootload console.

PLACING A CPU IN STEP

5. Having typed "VIP", you will receive the CPU CMD prompt. When you see this, type "CO DATA 024000717200" followed by "EX2".

6. When the system has returned to bce, you'll see the bce ready message displayed on the system bootload console.

VIP MODE COMMANDS (UNT CMD PROMPT)

Delete the BOS command.

SECTION MOH-N

# SUMMARY OF BCE COMMANDS

The Multics bootload command environment is described in detail in Section 5.5. All of the commands available to bce are summarized in this appendix for quick reference. This summary is formatted so that it can be removed from the manual for use as reference cards or for machine-room posting.

## alert

Usage: alert message

writes a message on the operator console with an audible alarm.

## boot

Usage: boot {command} {keywords} {cold}

causes bce to pass through the next phase of initialization, or to boot Multics service.

Valid commands: star, mult, stan, salv

Valid keywords: nodt, nolv, rlvs, rpvs

## bos

Usage: bos

causes bce to return to BOS.

## config_edit, config

Usage: config_edit {file_name}

enters the config deck editor.

continue, go

Usage: go

restores the machine image and continues running the interrupted activity.

delete, dl

Usage: delete star_names

deletes files within the bce file system.

die

Usage: die {-force | -fc}

aborts all bce activities.

dump

Usage: dump {macro_keyword} {-process_group segment_option {...segment_options}} {-force | -fc} {-dump #} {-crash} {-bce}

produces a diagnostic dump of system memory and tables into the dump partition.

Valid macro_keywords: -brief, -short, -long

Valid process_groups: -running, -initializer, -eligible, -all

Valid segment options: directories, hardcore, per_process, stacks, writeable

emergency_shutdown, esd

Usage: emergency_shutdown

starts an emergency shutdown of Multics.

exec_com, ec

Usage: exec_com ec_name {ec_arguments}

invokes a bce exec_com.

fwload, fw

Usage: fwload mpc_names

loads firmware into the specified mpcs.

get_flagbox, gfb

Usage: get_flagbox variable

determines the value of a variable maintained in the bce flagbox.

init_files

Usage: init_files {-force | -fc}

wipes out all files in the bce file system.

list, ls

Usage: list {star_names}

lists the names of bce files matching a set of star names.

list_requests, lr

Usage: list_requests

lists all requests valid at the current command level.

print, pr

Usage: print file_name

prints the contents of a file in the bce file system.

probe, pb

Usage: probe {-break | -bce | -crash}

used to examine, patch and generally debug Multics hardcore and bce itself.

Allowed requests:

before, b {ADDRESS}

sets a breakpoint before the specified address.

continue, c

continues the saved image.

dbr VALUE1 {VALUE2}

sets the dbr value used in the appending simulation.

display, ds ADDRESS {MODE {LENGTH}}

displays a set of locations in a specified mode.

let, l ADDRESS = VALUE {... VALUE}

modifies a series of locations.

list_requests, lr

lists the valid bce probe requests.

mc ADDRESS {-lg}

displays machine conditions.

name SEGNO

displays the name of the hardcore segment with segment number SEGNO.

proc N

changes the address space used by the appending simulation to the Nth process in the active process table.

quit, q

exits probe.

reset, r {ADDRESS}

resets the breakpoint at the specified address.

segno NAME

displays the segment number of the named hardcore segment.

stack, sk ADDRESS

displays a stack trace starting at the given address.

status, st {SEGNO|NAME}

displays a list of breakpoints set.

qedx, qx

Usage: qedx {-control_args} {macro_file} {macro_args}

invokes the qedx text editor to edit a bce file.

reinitialize

Usage: reinitialize {-force | -fc}

causes bce to perform a new initialization pass.

rename, rn

Usage: rename star_name equal_name {... star_name equal_name}

renames files in the bce file system.

set_flagbox, sfb

Usage: set_flagbox variable value

changes the value of a flagbox variable.

severity

Usage: severity prog_name

returns the severity, or extent of completion, of a preceding bce command.

shutdown_state, sds

Usage: shutdown_state

returns the state of completion of the shutdown of Multics service.

SECTION MOH-O

# HARDCORE BREAKPOINTS

The hardcore breakpoint facility is a collection of facilities within Multics and bce that allow probe style breakpoints to be set at most bce and hardcore instructions. They may be used largely as they are within normal Multics probe but with a few caveats.

## BREAKPOINT MECHANISM

This section describes the mechanism by which hardcore breakpoints is implemented. This is largely for academic interest; however, a understanding of the mechanism will prevent the user from setting a breakpoint in an incorrect path; in particular, breakpoints may not be set in the breakpoint handler's path.

When a hardcore breakpoint is set at an instruction, the instruction at that location is relocated to the end of the segment containing it. Its addressing is changed to reflect its new location. The original location is replaced with a transfer instruction to a breakpoint block at the end of the segment which executes a "drl -1" instruction. This causes the breakpoint to happen. If the breakpoint handler returns without changing the breakpoint, the next instruction in the block will be executed. This is the relocated original instruction. After this, a transfer is made back to the correct place in the original program. It should be noted that the instruction moved cannot be the second or later words of an eis multi-word instruction.

Derail faults are handled in fim. A "drl -1" instruction is special cased to be a breakpoint. fim makes a call to pmut$bce and return to implement the call to bce. Any program in this path (this part of fim, pmut, connect handling in stopping other processors, etc.) cannot have a breakpoint placed therein. Also, the special casing of a "drl -1" to be a breakpoint only applies for derails in ring 0. Thus, breakpoints should not be set in segments that will be executed in other rings.

When bce is invoked via the toehold, it notices that a breakpoint was the cause of the return to bce and invokes bce probe directly. Probe is free to perform a continue operation which eventually returns to pmut, restarts other processors, returns to fim who restarts the breakpointed operation.

Breakpoints may be set within bce also. However, they should be set only at the "boot" command level. When set at the "early" command level, a breakpoint will cause a return to the "early" command level. Also, a breakpoint set at the "crash" level is useless since, upon a breakpoint/crash of the "crash" command level, the toehold purposely does not save the crash image to avoid overwriting the Multics image already saved.

## BCE PROBE BREAKPOINT OPERATIONS

This section describes bce probe support of breakpoints.

### Breakpoint requests

before, b {address}
: sets a breakpoint to be executed before executing the instruction at the specified address. If no address is specified, "." is assumed. The address must be a virtual address. The breakpoint is added to the list of breakpoints for the segment. Up to 120 breakpoints may be set per hardcore segment; however, all wired hardcore segments share the same breakpoint area so only 120 breakpoints in total may be set in wired segments.

continue, c
: continue from a breakpoint. Multics is restarted.

reset, r {address}
: resets a given breakpoint; that is to say, Multics will no longer break when the instruction is encountered. The breakpoint causing the return to bce can be reset by not specifying an address.

status, st {name|segno}
: either lists all segments with breakpoints set in them (if no name or segno is specified) or lists all offsets within a single segment at which a breakpoint is set.

## Breakpoint references

When a breakpoint causes a return to bce, bce does not execute the bce_command in the flagbox. Instead, it enters probe directly. Probe will assume a default of "-break". Probe may be exited at this time. This does not effect a return to Multics however, only a return to bce ("crash" or "bce_crash") command level. Probe may also be entered with the control argument "-break" to force examining the breakpoint conditions. The only difference between "-break" and "-crash" for probe is the machine conditions to use. "-crash" uses registers contained within the toehold when the toehold was invoked. These registers are mostly interesting when bce is manually entered. "-break" uses the registers at the time of the breakpoint; these were saved by the breakpoint handler. The registers will show the register contents at the time of the breakpoint; however, the instruction counter will show the relocated instruction, not its original location.