To: Distribution

From: James A. Bush

Date: August 1, 1980

Subject: Hardcore Support for the DPSE Hardware

INTRODUCTION

This memorandum describes the software changes necessary for Multics (and BOS) to support the DPS8/70M (DPSE) hardware, for first customer ship.

HARDWARE OVERVIEW

So that the reader may better understand the necessity of the software changes presented in the MTB, it might be useful at this time to present a picture of the hardware changes from the current L68 CPU.

Physical Characteristics

The DPSE processor (and later the IOM and SCU) will be packaged in a new low profile cabinet, with a height of approx. 6 feet and the horizontal dimensions of a 4MW SCU. The configuration switch panel will be visible and usable on the skin exterior and will include an execute fault push button and a "busy" light. The maintenance panel on the current L68 has been replaced by a micro processor controlled dynamic maintenance panel. The dynamic maintenance panel can be controlled from a calculator like keyboard or optionally from the Diagnostic Processing Unit (DPU). All functions performed from the current maintenance panel can be performed with the dynamic maintenance panel, including setting and executing the data switches (read with rsw (0)).

History Registers

The DPSE will have 4 sets of 64 history registers compared to 4 sets of 16 on the L68 CPU.

Associative Memories

The PTW and SDW associative memories on the DPSE will be 4x16 4 way set associative, compared to a 1x16 full associative on the L68.

Cache Memory

The DPSE CPU will have an 8k store through cache memory compared to a 2k store through cache memory on the L68. The 8k cache will be selectively cleared by hardware control via write notify logic in the 4Mw SCUs, as compared to the software cache clearing in the L68. The encacheability of a segment will still be controlled by SDW bit 57.

Hexadecimal Floating Point

A mode will be provided in the DPSE CPU, to extend the floating point range significantly by making the floating point exponent a power of 16 instead of a power of 2. Hex floating point mode will be controlled by a bit in the cpu mode register in conjunction with a bit in the indicator register.

Memory Ports

There will be only 4 memory ports on each DPSE CPU, as compared to 8 memory ports on the L68. With the increased density of mos memory, 4 - 4Mw SCUs configured to their maximum of 4MW per SCU will give a full address range of 16MW.

Changes to the Read Switch (RSW) Instructions

Since there are only 4 memory ports, rsw (3) has been eliminated. Interlace information contained in rsw (4) has been moved to rsw (2) and rsw (4) has been eliminated. Configuration information located in rsw (2) has been restructured to identify the cpu as a DPSE, identify the presence of 8k cache memory and accommodate memory interlace information moved from rsw (4).

Diagnostic Processing Unit (DPU)

A DPU will be provided that will allow field Engineering to perform remote maintenance from a Tactical Assistance Center (TAC). The DPU is a Honeywell Level 6 minicomputer connected externally to the central system via an rs-232 communication link within the Low Cost console (LCC). The LCC is therefore required for all shipments of DPSE systems.

FUNCTIONAL SOFTWARE CHANGES

Instead of defining each module that must be changed for the DPSE, the functional area of change will be detailed below.

Saving History Registers

Currently history registers are stored after each fault which is handled by the fim, and under certain conditions, (mc_trace) history registers are saved after every fault. History register data is stored in each process's pds in a 128 word wired buffer. If the fault is handled by the signaller, the history registers are copied into the stack frame for return_to_ring_n_, where they are available to be inspected by the user. If we were to save all 64 history registers for the DPSE, then the wired area in the pds would have to be increased to 512 words. This would cause the pds wired area to have to be increased to 2 pages and would prove to be a severe performance penalty for the added information in the additional 48 history registers. Therefore, it has been decided to save the 16 MRU history registers so that the net effect will be the same as it is today. Unfortunately, the instructions that store the history registers do not allow you to start at the 16th MRU history register, but rather start at the LRU (64th) history register storing one history register, incrementing the history register counter so that the next instruction to store the history register will store the next LRU (63rd). This procedure must be repeated until all 64 history registers have been stored.

The saving of history registers will be accomplished for the DPSE by storing 4 sets of 16 history registers in place in the pds$history_reg_data area, so that the final contents will have the 16 MRU history registers. This procedure is expensive in time since we are throwing away 75% of the history register data anyway. Also, history registers in most cases have not proven very interesting anyway. Therefore the rules governing when to save history registers will be changed for the DPSE and will also prove to be a performance improvement for the current L68 as well. My proposal for saving history registers follows:

- Turn the per-process switch, pds$save_hist_regs, off by default. (this switch is currently compiled as on).

- Add an hcs_ gate entry to allow user control of the pds$save_hist_regs switch. (Probably hcs_$save_history_regs ("0"b) or ("1"b)). Programs that really want history registers will use this gate entry to save the history regs. (mc_trace, test_cpu, eis_tester, etc).

- Add a user command to interface to this gate entry as well (save_history_regs on or off).

- Add a force_save_history_regs entry to the history register save subroutine, that will save history regs regardless of the state of pds$save_hist_reg_sw. This feature is needed by parity faults, since the cache parity error logging routine relies on history regs being present. Also the other fault types that are handled by hardware_fault (op-not-complete, command, startup, shutdown and parity), need the history registers since they are copied into the syserr_log for use by analysis tools like heals and elan.

- The history register save routine would then set another per-process switch (pds$hregs_saved), if the history regs were stored. (Switch would be reset if they were not saved).

- The signaller would then check the pds$hregs_saved flag instead of the pds$save_history_reg switch before copying the hregs into the return_to_ring_n_ stack frame. If the pds$hregs_saved switch was not set, the history register area in the return_to_ring_n_ stack frame would be filled with zero. This would allow the history register analyzer to heuristically determine that no history regs exist instead of printing out garbage.

Saving Associative Memory

The PTW and SDW Associative memories have increased from a 1x16 full associative, for the L68, to a 4x16 4 way set associative, for the DPSE. This means that it will take 4 times as much storage (and 4 times as long) to save them, on the DPSE CPU as it does on the current L68. There is currently 128 words reserved in the prds for storage of the associative memories. This would have to be increased to 512 words in order to save the associative memories for the DPSE CPU as is done today. Currently, the associative memories are stored in the prds by fim_util$save_history_regs (the same subroutine that stores the history registers), and is conditioned by the state of pds$save_history_reg switch. This means that every time that the history registers are stored, the associative memories are stored as well. Therefore, all faults handled by the fim, including segment faults and linkage faults, would store the associative memory by default.

I propose that this saving of associative memories is a needless and time wasting act and should be eliminated, for both the DPSE and the current L68 as well. The only place

in the system that I know that makes use of the associative memory images in the prds, is ol_dump when looking at a Multics fdump image. By eliminating the associative memory saving, time and storage will be saved.

Changes to Initialization and Dynamic Reconfiguration

Because of the existence of only 4 memory ports and the changes to the read switch (RSW) instructions, dynamic mainframe reconfiguration will have to be changed. Details of these changes are listed below:

- Change the scs include file (and the scs cds segment), to have a bit in the processor_data structure indicating that this CPU is a DPSE. Also add an 8 element array to scs (indexed by cpu tag), that defines the cache size of each configured CPU. This array is necessary in mixed L68/DPSE systems so that cache parity error logging software in fim and cache_tester can quickly determine cache size.

- Change the rsw include file to define the new rsw (2) data format and provide a constant for how many rsw types are valid for each cpu type (e.g. 5 for the L68 (rsw 0, 1, 2, 3, and 4) and 3 for the DPSE cpu (rsw 0, 1, and 2)).

- scas_init - change this initialization module to check if any memories are configured with a tag greater than 0, if a DPSE cpu is found in the config deck.

- start_cpu - set the DPSE indicator bit in scs$processor_data, and set the cache_size to 8 in the scs$cache_size array (assuming rsw (2) indicates that cache is present). Process the interlace information located in rsw (2).

- init_processor - change the code that gets executed when the new cpu is "SMIC'ed" from start_cpu to check the rsw (2) information first. If the cpu type identifier in rsw (2) (bits 4 & 5) indicates a DPSE cpu, then do not execute the rsw (3) and rsw (4) instructions.

- configure_test_cpu - (ISOLTS hardcore reconfiguration). Change read switch masks and templates to reflect absence of rsw (3) and rsw (4) and the format change of the rsw (2) instruction for the DPSE cpu. Execute only rsw (1) and rsw (2) on DPSE cpus, during the read switch test. Return the contents of rsw (1) and rsw (2) to the outer ring caller, so that the cpu type will be known.

o As an aid to dynamic reconfiguration and for better user visibility, the cpu config card will be changed to add a cpu type and model field. The card layout and examples follow:

cpu <tag> <type> <model> <port> <state> <exp. port>

```
cpu  a  dps8  70.  4  on      for a DPS8 CPU with 8k cache
cpu  b  dps8  52.  5  on      for a DPS8 CPU without cache
cpu  c  l68   80.  6  off     for an L68 CPU with 2k cache
cpu  d  l68   60.  7  test    for an L68 CPU without cache
```

The hardware_config_cards include file will be modified to incorporate these changes and since this is an incompatible change, all hardcore and user ring software that modify or inspect the cpu card will be checked (and modified) for conformance.

Support For 8K Cache Memory

Software changes to support 8K cache will be minimal. Hardware cache clearing will provide all of the functions that the current software clearing provides. In addition the CAMS and CAMP instructions on the DPS8 cpu will ignore bit 15 (selective and full cache clear), so that page control will not have to be changed and will be compatible with the L68. Changes that will be required for 8K cache support are as follows.

o fim - change the cache parity snap shot routine to make the abs_seg for looking at cache errors modulo 8K, for a DPS8 cpu and modulo 2k for an L68 cpu. Currently, the abs_seg is forced to modulo 2k.

o cache_tester - change to get a modulo 8K wired contiguous buffer segment for static cache testing, instead of a modulo 2K. Add board and chip call outs relevant to 8K cache.

o There is an extended fault register associated with 8K cache directory parity errors. The information contained in this extended fault register is potentially valuable for hardware failure analysis, since it breaks down the location of a cache directory parity much finer than is done in the L68. This register (11 bits of useful information) should be saved in the machine condition area, when a parity fault occurs. However, there is currently no vacant space in the current 48 word machine condition area to store it.

I propose doing away with the faulting ring field (mc.ring) which occupies the most significant 18 bits of

the fault_time area, where the 52 bit clock value is stored. I have found no place in the system where the mc.ring value is used. These 18 bits would provide ample storage for the extended fault register. It might also be useful to store the cpu type (from bits 4 & 5 of rsw (2)) on which the machine conditions occurred.

Support for Hexadecimal Floating Point

There is currently an MR9.0 PFS item for a study to be conducted on how to best implement Hex floating point on Multics. Until such a time as the findings of this study is released, hex floating point mode will be forcibly disabled on the DPSE. This will be accomplished by having init_processor set bit 33 of the cpu mode register to a zero state, during processor initialization.

BOS CHANGES TO SUPPORT DPSE

There appears to be only minor changes needed to BOS to operationally support the DPSE cpu. Most of the changes necessary, will be in those areas where BOS is concerned with the Multics machine image. The modules that must be changed are as follows:

- Change the setup routine, mcsave, to save all 64 history registers and all 64 associative memories, if the bos processor is a DPSE. This also implies that the machine condition storage area in bos common will have to be increased in size. Also, the fdump image and the include file bos_dump.incl.pl1 will have to be changed to reflect this areas increase in size.

- The cache dumper "abs cache" will have to be updated to accommodate the 8K cache.

LISTS OF MODULES TO BE CHANGED

Included in this section, are lists of modules and include files that must be changed for the DPSE. Each list contains the module name, functional area of the change and the degree of change expressed as "easy", "medium", and "hard". For time accounting purposes, the degree of change could be broken down as follows:

easy - 0 - 1 man days
medium - 1 - 3 man days
hard - 3 - 5 man days

Multics Hardcore Modules

Module	Area	Change
cache_tester.pl1	8k cache	medium
configure_test_cpu.pl1	init. & reconfig	medium
fim.alm	8k cache, history regs	medium
fim_util.alm	history regs, assoc. mem, 8k cache	hard
hardware_fault.pl1	8k cache	easy
hcs_.alm	history regs	easy
history_reg_save.pl1	history regs, (new module)	medium
init_processor.alm	init. & reconfig	medium
page_fault.alm	history regs	easy
pds.cds	history regs	easy
rsw_util.pl1	init. & reconfig	easy
scas_init.pl1	init. & reconfig	medium
scs.cds	init. & reconfig	easy
signaller.alm	history regs	easy
start_cpu.pl1	init. & reconfig	easy

Multics Online Modules

Module	Area	Change
hran_.pl1	history regs	hard
isolts_.pl1	DPSE, (to be done by T&D group)	hard +
ol_dump.pl1	history regs, assoc. memory	medium
save_history_regs.pl1	history regs, (new command)	easy

Multics Include File Changes

The following list of include files will be changed in the course of modifying the previously listed modules. All other modules that reference these include files (identified by using the total cross reference), will be recompiled/reassembled before the DPSE software is installed.

bos_dump.incl.pl1
hardware_config_cards.incl.pl1
history_regs.incl.pl1 (currently not used, but should be, when hrgn_ is modified)

mc.incl.alm
mc.incl.pl1
rsw.incl.pl1
scs.incl.alm
scs.incl.pl1

BOS Module and Include File Changes

Module	Area	Change
abs.alm	8k cache	hard
bos_common.incl.alm	history regs, assoc. memory	easy
dump.alm	history regs, assoc. memory	medium
fdmp.incl.alm	history regs, assoc. memory	easy
setup.alm	history regs, assoc. memory	easy