68040 PORTING PLAN 9/8/89

Attached is an updated plan for porting the current S90 kernel to the DPM040, based on the latest DPM040 Functional Specification and the latest information from Motorola, and incorporates revisions from the earlier design review. Included is an overall functional specification, a more detailed schedule, the plan for the port, and a more detailed list of modifications to be made.

Randy

Introduction

This document discusses the Unix Operating System and associated software being developed for the Motorola 68040 Dual Processor Module. This will be used as one of the alternative processor modules / software sets for the System 90. This specification is intended to cover functional definitions of the software interface and related topics, and later issues of design. No assumptions should be made with respect to functionality or support not included in this document. If there are areas of support, not discussed, which are required in this product, they should be brought to the attention of the writer for inclusion. Throughout this document the system under discussion will be called the "System 90/040".

Related Documentation

MC68040 Preliminary Design Specification Revision 6.0

MC68040 Preliminary Design Specification Addendum Revision 6.2

MC68030 User's Manual

"68040 Dual Processor Module Functional Description", by Dan Peterson, dated 8/25/89

Motorola 68000 Processor Supplement for the System V ABI, First Industry Review Draft, June, 1989

Design Objectives

The major objective of the System 90/040 project is to create a software environment which will have the "Look and Feel" of the 68020 based System 90 as it exists at the time of release, but with all the PM20 boards replaced by a DPM40s. It is intended to track the functionality and standards compliance of the System 90 as closely as possible. The functional objectives are discussed in more detail below.

A The System 90/040 will conform to the System V Interface Definition in the same manner as the most current System 90 software release.

B The System 90/040 will offer the same level of compliance with the X/Open Portability Guide, System V Specification. This will reflect the current System 90 compliance.

C The System 90/040 will demonstrate the same functionality and present the same user interface as the ARIX System 90/020. The procedures for operation, including the standalone environment, will be the same as the System 90/020. The System 90/040 will be source code compatible with the System 90/020 for architecture and compiler independent source code.

D The 68040 runs better when there is strict alignment of data, in particular, longs always on long boundaries. On the PM20s, the superblock (filesystem header) of each filesystem on the disks has longs on short boundaries. In case of an upgrade from a PM20 to a DPM040 based system, tthe 68040 superblock will be incompatible with the existing filesystems. A standalone utility will need to be run to update the superblocks on all the filesystems.

E The System 90/040 will not be ABI conformant. In particular, the ABI specifies a minimum user address space of 3 1/2 Gigabytes, which cannot be supported by the PM20s making common source code difficult, and requires a significant software development effort to implement. This issue can be readdressed in V.4.

F All machine independent S90/020 binaries will run on the S90/040. The System 90/040 will NOT run A1000 binaries.

Software Components of the System 90/040

The following is a description of each of the components of the Arix System 90/040 software.

A System 90/040 Unix Kernel

The initial release of the System 90/040 Unix kernel will be based upon the most current System 90 kernel source. At this time, it appears that it will be V.3, rather than V.4.

B System 90/040 Language Processors - C Compiler

The SGS for the MC68030 needs to be extended with the addition of the new 040 instructions. Additionally, the 68040 will run better if all accesses are on "correct" boundaries, i.e. shorts on short boundaries and longs on long boundaries. The compiler will need to be modified to generate the proper alignment, and libraries may need to be updated. For compatibility, a mechanism will be provided in both the 68020 and 68040 SGSs to force a specified alignment in specific structures. We will probably need to provide a 68020 SGS for third party software, but its general use should be discouraged.

Machine independent codefiles with the 68020 alignment, eg. made for the S90/020, will run without modification. The kernel will be updated to support the differences in structure and data alignment.

C System 90/040 Unix Utilities

No functional changes to the current System 90 utilities are required. Some source modification is required for compilation purposes. The utilities should be recompiled

due to changes in the alignment of structures and for better performance.

D CSS Bus Module Support

All modules (except other types of PMs) will run on the System 90/040. This means that both A1000 IO and IOPMs will be supported.

E System 90/040 Communication Products

The full range of A1000 and IOPM based communication products offered the System 90 will be supported. By changing the alignment of items in structures in the 68040 compiler, there will be some porting required.

F Diagnostics

New firmware is required for the DPM040s and the SPM.

General

There will no hard-coded limitations on the number of processors configurable in the system. Many different areas of the system must be taken into consideration to determine the actual physical limitations of the number of processors including the SPM, configured memory, and backplane priority arbitration to name a few. Recommendations for the optimal number of processors will be made later. At this point the guidelines appear to be:

1 DPM board to 2 MMs (regardless of the size of the MM)
and
3-4 DPMs (dual) per system.

The minimum system will be one PM with a single M68040 processor.

The software will not support reconfiguring of a bad processor out of a DPM, since the hardware will not continue if either processor on a DPM fails.

Hardware upgrade from a System 90 will require the replacement of all of the 68020 PM boards with at least one 040 DPM board. Software upgrade will require a load of new operating system software, including kernel, SPM, and bootimage diagnostics.

Floating point and memory management operations are supported on the M68040 cpu.

Software Porting Effort for the 68040 9/8/89

Kernel (12 - 18 weeks)

4-6 wk modifications for new MMU (new page table entry structure, 3-level tables, relayout of kernel space, IOPM & A1000 I/O interface, binary compatibility issues)

1 wk redo one PM per slot concepts (new structure, new addressing of other boards)

1-2 wk replace iomap with memory map

1-2 wk change all PM20 machine dependencies - redo lio.h (cache, led's, interrupt registers) & startup.c (initialization)

1 wk write new interrupt handlers (eg. level 7)

days redo assembler code for 040 (add MOVE16's and MOVS)

days redo TLB and cache management, check on atomic operations not covered by CAS's now

1 wk handle alignment issues (include A1000 I/O)

1 wk miscellaneous (ptrace, signals, etc.)

MOTOROLA DEPENDENCIES:

1 wk add floating point support [ONCE PACKAGE ARRIVES FROM MOTOROLA - expected in September]

days if get from Motorola [expected Nov./Dec.], 2 wks if not rewrite bus error handler

SPM (3 weeks)
---

1 wk redo one PM per slot concepts

2 wk support new MMU & memory map, rest of DPM040 features (eg. initialization)

? misc.

IOPM (1 - 2 weeks by IOPM group)
---

redo on PM per slot concepts

support new MMU & memory map

SGS (4 - 6 weeks)
---

1 wk add the pack pragma to existing C compilers

2 wk redo the assembler (and get the optimizers to recognize new codes)

1 wk change the alignment in the code generator (this should take an hour - but we should check to see if anything breaks in the process)

1.5-2.5 wk verify alignment in libraries

Utilities (1 - 2 weeks)
---

update makefiles, etc.

check alignment issues

handle any machine dependent issues (disassemblers)

Project time for initial port:

By kernel group:

Kernel	12 - 18 weeks
SPM	3 weeks
debug	3 weeks (initial debug, prior to receiving DPM040)
integration & utilities	1 - 2 weeks
	19 - 26 weeks, or 5 - 6 1/2 months

By IOPM group:

3 weeks

By languages group:

1 - 2 weeks

KERNEL CHANGES:

Changes between the PM20 and DPM040:
---

1. Dual Processor

   Wherever the kernel referenced a CSS slot, we now need an alternate structure to maintain which CPUs are where, eg. (slot, A/B). This then gets extended to how the kernel addresses another PM, i.e. which subslot. The structures and algorithms to maintain this will not assume at most 2 processors per board.

   The architecture of the DPM040 and the System 90 is such that for dispatched interrupts (i.e. interrupts sent via the dispatcher, like from the VAM), only CPU A gets them, even though they may be intended for CPU B. We need to provide a software mechanism for passing the interrupt from one CPU to the other, probably by extending the current software interrupt scheme.

2. Memory Map

   The 68040 generates 32 bit addresses, but the CSS bus requires a 36 bit address, with the slot id of the destination board as the high order 4 bits. A memory map has been added to the DPM040 covering 0 - 0xfeffffff (the high order 16 Megabytes is local io space). All addresses coming out of the 68040s on the PM board must go through the memory map. This includes all of what was in the iomap on the PM20, eg. addressing PMs, ICBs, and IOPMs, plus all memory references. This scheme was partially implemented for the 486 (including memory striping). The kernel and SPM routines for walking through memory (eg. vtop, kmem_to_*) must be totally rewritten.

   Since the PM20 iomap does not appear on the DPM040, we need to replace all references to it in the kernel with accesses through the memory map. In particular, the iomap is set up to address the top Megabyte of each PM so that interrupts from one PM to another is a simple write to an address in the iomap range; a similar mechanism needs to be designed. The allocation of addresses in the memory map for off-board references must be determined; for example, ICBs could stay at the same addresses as they did on the PM20 and cpus map in at, say, 0xd0000000 in 1 Megabyte chunks.

   The initial memory map will be created by the DPM040 firmware having been passed the relevant information by the SPM. This map should allow for accessing all of memory and perhaps the SPM. The procedure for passing the memory map tables to the processor is described in "DPM040 Interface Between Firmware and OS/Kernel", August 8, 1989, by Shih-Nan Huang. The memory map is shared by both processors. This means that during initialization, only one processor on the board should initialize the rest of the map.

3. New registers and address spaces (lio.h)

Most significantly, interrupt handling is different. For example, the format of the interrupt request level register has changed.

We need to account for all the new and different registers, eg. the DPM status register.

4. Addition of a secondary cache

We need to make sure that there are no coherency issues, plus we might want to replace the cache validity test in the kernel for the PM20 with the appropriate one for the DPM040.

5. Interrupts

We now have software interrupts along with hardware interrupts at most levels. What can be done to use these?

A level 7 interrupt is now one of the following:

ipc register write
snoop fifo overrun
invalid map entry
BUS FREEZE
non-read NMI
snoop error

so on the receipt of a level 7 interrupt, we'll need to decipher which interrupt was received and handle it appropriately. Since the DPM040 supports levels 1-6, level 7 does not get cleared by clearing a hardware interrupt in the 'Clear Interrupt Port', but rather by clearing the specific NMI in that register. Once in the NMI handler, all NMIs should be serviced before returning from the handler, that is, the Status Register should be checked that another condition has not arisen while the first NMI was being serviced.

The current interrupt layout on a PM20 is

```
    0 - error
    1 - software interrupt (timein, runqueues, and iopm response)
  * 2 - edt interrupt or iopm interrupt from dispatcher
  * 3 - gc & mac interrupts from dispatcher (done at level 4)
    4 - software interrupt (redirected level 3)
  * 5 - console interrupts from dispatcher
  * 6 - clock (SPM writes into interrupt dispatcher)
    7 - tdb entry from SPM
```

where * means that the dispatcher must be cleared as part of the interrupt handler.

We know that 6 will no longer go through the dispatcher (in the 3.0 kernel release), as the SPM will write it directly to the CPU. 5 can be done the same way, since it also goes through the SPM. In the case of 2 & 3, it doesn't look like it can be changed, especially with the introduction of the VAM.

Because CPU A always gets the hardware interrupts, CPU B's

interrupt service routines need not acknowledge the dispatcher. One way of handling this is to have 2 different sets of interrupt routines using 2 different trap vectors. Those for A
- determines if it received a hardware or a software interrupt
- clears the interrupt in the Clear Interrupt Register
- if it was a hardware interrupt, acknowledges the dispatcher
- calls the appropriate C level routine.

Those for B just clears the interrupt (it is always a software interrupt) and calls the C level handler. Although CPU A has more work to do, CPU B's code is much simpler and faster. Or one set of routines can be maintained, with tests for which CPU is running scattered throughout. In the new interrupt scheme to be introduced in the kernel, the dispatcher is acknowledged at the C level, so this scheme is somewhat incompatible if we allow for hardware and software interrupts at the same level.

6. Faults on DPM040

Handling of such errors as a CSS bus error on a delayed write is TBS. What is known now for NMI handling is:

ipc register write - fatal error since should not occur when UNIX is running (unless this is used for an SPM tdb interrupt).

snoop FIFO overrun -

[bus watcher is automatically turned off by interrupt]
clear chip cache
clear CPU tags by writing 0s to the whole area
[don't clear bus watcher tags - this forces extra snoops, but don't have to clear out the other cpu's cache]
clear FIFO full interrupt
[bus watcher is turned on by clearing FIFO full interrupt]

invalid map entry - treat as bus error?

BUS FREEZE (or SPM interrupt) - enter debugger

Changes due to using the MC68040 instead of the MC68020:

There are 4 major areas:

core processor
on-chip floating point
on-chip MMU
on-chip primary cache (although there is a cache on the 020)

1. Core Processor

There is a new instruction, MOVE16, which transfers data in bursts bypassing the cache. It should be used in the buffer copying routines in userio.c & useriorfs.c.

The major item here is that "the CAS and CAS2 instructions will always have at least one write to the effective address, even if the operand compare is unsuccessful" (MC68040 Design Specification Revision 6.2, pg. 2). In the current kernel, certain items are modified both with CAS's and with regular (i.e. non read-modify-write) assignments. This will result in a race condition, because read-modify-writes only lock out other read-modify-writes and the assignment may occur between the read of the CAS and the write back of its value. [On the PM20, the previous value was not written back, so at worst the CAS would fail and then succeed when retried.] The areas of concern are:

a) Software locks

Software locks are cleared just by writing a 0 and not by an atomic operation. Given two cpus, with the first attempting to get the lock and the second giving it up, if the first attempts to get it with a CAS, the original non-zero value is written back on the failure. If the other clears it by writing 0, the 0 could be overwritten by the write by the first cpu, leaving the lock locked forever. All places in the kernel where writing 0 is done to unlock locks must be changed. In particular, the

```
clr.w   (%a0)
```

in spin_unlock() and exit_short_cr() must be replaced with a CAS with 0.

b) All items modified using the atom_xxx routines must be checked to make sure that all updates done on them are through atomic operations. For example, in the proc structure, p_special, p_flag, and p_sig are changed using the atom_xxx routines. All places where p_special is changed are fine. Changes to p_flag in sig.c and trap.c are done non-atomically and must be changed. Likewise p_sig is not atomically changed in ssig() in sys4.c and in various files in the nudnix directory. Perhaps the omission of the atomic updates on these fields are bugs in the current source.

The queueing routines for the interrupt queues use CAS to enqueue items, but not to dequeue them. No CAS is needed for the dequeue, since the CAS is never to a spot being written when removing an item from the queue.

Many atomic operations are done on the upkern and should be checked. For each atomic operation, all the other software (eg. IOPM or SPM) that may use that field in the upkern structure must be checked to ensure that those do atomic updates also. This not only applies to the upkern, but to any other shared structure.

c) In the code executed when returning from a trap (trap_ret in trap.s), queueflag is set with a CAS, but cleared with a 0. This must change if this area is not redesigned.

d) Oracle previously called the kernel to do locking via a CAS for its synchronization routines (trap 2 in trap.s) but cleared locks by writing a 0. In the S90 version of Oracle, the kernel is no longer called to do the locking, and both the locks and unlocks are done via CAS's. Hence, no changes are required here. Informix, and perhaps other third party software, does use trap 2. New libraries should be provided such that trap 2 is no longer called, and trap2 should be removed from the 68040 kernel.

Because of the new optimized addressing modes and optimized instructions, all assembly code should be looked at to see if any reordering should be done. For example, branches taken are now the optimized path, rather than branch-not-taken.

One serious matter is the failure of the 68040 to detect aliasing in its write buffer. The greater problem of aliasing in the 68030 does not occur here, since the 040 has a physical internal cache. However, the 040, in allowing reads to jump over writes, checks that there are no collisions using the virtual addresses, rather than the physical ones. According to the Preliminary Design Specification, Revision 6.2, page 7, "given that two different logical addresses map to the same physical address, and a write to one of these addresses closely precedes a read to the other address, an undetectable collision will occur. The read can take place before the write and use the wrong data value." This is a problem for our kernel (and probably all UNIXs) since we access the own structure in two different ways. We have contacted Motorola about this issue.

Another matter which needs to be checked is that, whereas the 68020 uses instruction continuation to support virtual memory, the 68040 uses a restart exception model and does not support a virtual machine. The exception handlers (trap.c & buserr.c) must be examined and updated if necessary.

2. Floating point

The instructions directly supported are a subset of the existing ones; but the programmer's model is the same. In terms of support

for the on-chip unit rather than another chip, the kernel need not be changed (eg. o_fpu_loaded).

A useful optimization would be to restore a process' floating point registers only on demand after a context switch, rather than automatically when going back to the user from the kernel. However, there does not seem to be a provision in the 68040 to put the floating point unit in some state such that a fault will be generated the first time there is an attempt to execute a floating point instruction.

The 68040 does not support all the instructions and data types that are supported by the 68881. According to the 68040 User's Manual, Revision 6.2, pg. A-2, "Motorola will have available an emulator software package for unimplemented instructions/data types." Motorola is sending the package to us in September; we will need to port it to the kernel and libraries. To handle the emulation, a new exception vector "unimplemented floating point data type exception" was added at 55. This is for packed decimal or a denormalized or unnormalized operand encountered for a binary floating point data type. The 68040 generates a format $0 stack frame for this. Unimplemented instructions trap to an F-line exception with a stack format of $2. The trap handler must then do an FSAVE to get the unimplemented instruction stack frame. Modifications to state.h for the stack frames, scb.s for the new exception, trap.s for support for the vector at 55, and trap.c for the different F-line exception handling, must be made.

The system interface has been redone on the 68040, so the floating point unit does not appear to be either a 68881 or a 68882. In particular, the state identification in the frames saved by a FSAVE have changed:

	68881		68040	
	version	format	version	format
null	00	00	00	??/00
idle	#	18	#	00
busy	#	b4	40	60
unimpl	-	-	40	28

where '#' is the version number and 'unimpl' is for the unimplemented instruction exception. The S90 kernel distinguishes state by only checking the format, since it is unique for all 3 state frames. In the 68040, there is no difference between the null and idle formats, so in all the kernel routines which check for the state the version must be checked also.

The documentation does not specify how to get the mask of the floating point unit on the chip (nor the processor part either). This may matter in the future if there are revisions to the floating point part.

3. MMU

The 68040 implements a 3-level table rather than the current 2-level

one on the PM20. A 3-level scheme has already been designed, implemented, and tested for SPARC. It will need to be ported to the 040. Of course, since the page table layout is different, the .h files (eg. immu.h) and the use of macros within the source files will require a degree of rewrite.

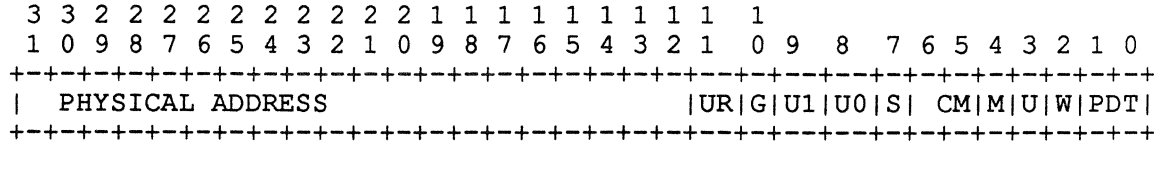
The PM20 has the user address space in the kernel address space automatically, that is, if the kernel is accessing a user address (any address less than 0x80000000), it will use the user root pointer (urxxt). The 68040 cannot do this because it has no knowledge of what a user address is, only that it is doing a particular type of access based on the function code registers. Hence, all the copying routines (copyin/copyout) need to do 'movs' (movsu/movus) [the instruction is already used in the {f,s}u{byte,word,long} routines]. The code originally had the 'movs' instruction, but was modified for the PM20 as an optimization; for the 68040 only, the code needs to be restored.

The MMU can have either 4K or 8K page sizes. We have decided to use 4K for compatibility and because using 8K pages would require an excessively large bitmap to keep track of pages (??). The firmware is responsible for the initialization of the Translation Control Register (TC) for 4K pagesize (P=0) and translations enabled (E=1), and for flushing the TLB before transferring control to the kernel.

The new MMU facilitates the implementation of shared memory, because we can take advantage of the writeable bit in the level-2 page table (as is done in the original AT&T code). [James says it will take very little to change this.] Shared memory between S90 020 binaries and 040 binaries will be possible without the problems incurred in the sharing of A1000 and S90 binaries. However, A1000 BINARIES WILL NOT BE SUPPORTED.

The general plan for mmu support is:

The layout of a page table entry for the M68040 is:



	Usage
UR - USER RESERVED	none
G - GLOBALLY SHARED	entry not invalidated by certain PFLUSHs
U1 - USER PAGE ATTRIBUTE 1	none
U0 - USER PAGE ATTRIBUTE 0	none
S - SUPERVISOR PROTECTED	
CM - CACHE MODE	

00 : Cacheable, Write through
01 : Cacheable, Copyback
10 : Cache Inhibited, Serialized | sequential model of execution
11 : Cache Inhibited, Not Serialized

```
M  - MODIFIED                |
U  - USED                    | referenced
W  - WRITE PROTECTED         | read-only page
PDT- PAGE DESCRIPTOR TYPE    |
   00 : Invalid              | page is not resident or invalid address
   01 : Resident             | page is resident
   10 : Indirect             | descriptor is an indirect descriptor
   11 : Invalid              | same as 00
```

The physical address is the value before going through the memory map, and does not have the slot id in it as was done on the PM20.

A proposed usage of the bits are:

```
UR                           | pg_islocked
G                            | Use on certain kernel pages to keep
                             | them in the tlb??
U1                           | Reserved for copy-back
U0                           | copy-on-write
S                            | use as part of permission bits (see below)
CM                           |
   00 : C'able, Write thru   | CACHEABLE
   01 : C'able, Copyback     | Not valid
   10 : ~C'able, Serializd   | Use for IO mapping
   11 : ~C'able,~Serializd   | NOT CACHEABLE
M                            | MODIFIED
U                            | REFERENCED
W                            | ~PG_W
PDT                          |
   00 : Invalid              | Entry is not a valid descriptor.
   01 : Resident             | PG_PRESENT
   10 : Indirect             | (Not used.  Could it be?)
   11 : Invalid              | Page is not resident in memory.
```

Because many of the bits in the page table entry are inverses of the bits in the PM20, eg. PG_W, or multiple bits map onto a single PM20 bit, eg. PG_PRESENT, the current macros for accessing and setting the bits must be reworked.

Moving the software attribute 'locked' into the page table entry from the disk block descriptor (dbd) can be done now that there is an available bit. (Both 'locked' and 'needs reference' are in the ptes in the 386 and 3b2 implementations of UNIX.) It provides a slightly faster region management code, since the extra access to the dbd need not be done to get at the bit, at the cost of maintaining two versions of the same code.

The U0 & U1 bits are carried out to the pins of the 68040, but these are not used now by the DPM040. U1 has been reserved for future use for implementing copy-back caches.

There are no explicit read or execute bits in the page descriptor entry as is done on the PM20. Execute permissions are not necessary, although we lose some flexibility of permissions (that is, a data/stack section would be executable as well as

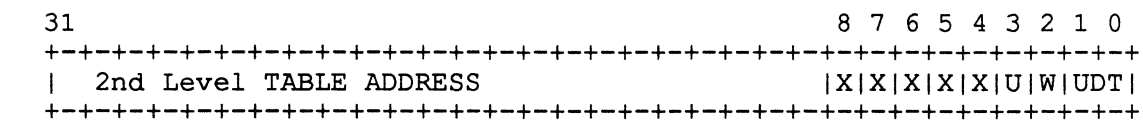
writable). Using the SUPERVISOR PROTECTED and WRITE PROTECTED bits in the page table entry, the permission encodings will be

```
S  W  | user | supervisor
------+------+-----------
0  1  |  r-x |  r-x      # for user text
0  0  |  rwx |  rwx      # for user data/stack
1  1  |  --- |  r-x      # for system text
1  0  |  --- |  rwx      # for system data
```

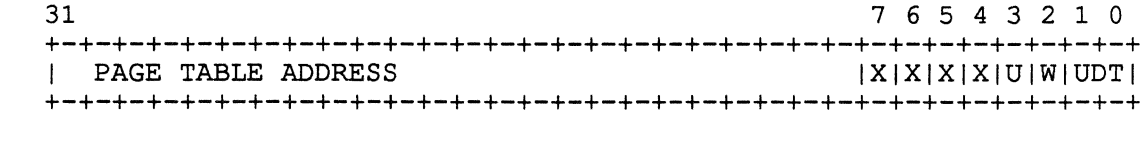
As of now, there is no intended use of the indirect pointer at the page descriptor level (it could be used for shared memory). Additionally, the use of the global bit in the page table has not been defined. However, it seems reasonable to mark the kernel code and data as global, so that TLB flushes will not flush it out. Syssegs data should not be marked as global though.

The layout of the upper level tables is:

1ST LEVEL DESCRIPTOR:



2ND LEVEL DESCRIPTOR:



where UDT- UPPER LEVEL DESCRIPTOR TYPE is
- 00 : Invalid
- 01 : Invalid
- 10 : Resident
- 11 : Resident

WRITE PROTECT in the level 2 table will be used for shared segments which are attached both read/write and read-only. This will get around the problems which appeared in the PM20 implementation.

For optimization purposes, the USED bit in the level 1 table should always be set by software.

The addition of the extra level of page tables will result in the 'loadstbl' routine to be modified greatly. In places where it was called, we now have to check to see if it succeeded, because it now must allocate a page table which it did not have to do before. Because of the high degree of machine dependency of this procedure, it should be moved out and made into a machine dependent .c file. This has been done in the 386 source already. Another place which needs to be modified is 'uballoc'

and 'ubfree' in fork.c; these routines depend on descriptors looking relatively the same at each level, which is true of a PM20 but not a 68040.

User code file layout:

Machine independent binaries compiled on the S90 should run unchanged on the 040 because the changes to the instruction set do not affect user code:

- General coprocessor instructions and module support (CALLM and RTM) were dropped, but never used
- PLOAD and PMOVE were dropped, but they were PMMU instructions and as such, never used
- MOVEC was modified, but it is a privileged instruction, so no user ever used it
- BKPT is now slightly restricted, but it is not used (sdb uses TRAP 1 for breakpoints)
- PFLUSH and PTEST were modified, but were PMMU instructions anyway.

Floating point instructions in S90 codefiles will be supported directly by the 040 or by emulation in the kernel.

Kernel memory layout: (kmem.h)

If we conform to the M68000 ABI, the kernel has only 1/2 Gig of memory, starting at 0xe0000000. Right now it does not seem possible to fit the kernel into 512 Megabytes, so we would have to go to overlapping user and kernel spaces. The initial release of the kernel on the 68040 is V.3 based and is not ABI conformant. Hence, there is no need to radically change the address ranges for user space and kernel space. The kernel addresses just need to be modified to include another level of page tables.

Kernel addresses were chosen so that collisions in the TLB on the PM20 would be minimized. Given the new on-chip TLB, which is 4-way set-associative with different hashing, new addresses could be determined.

Page tables (all levels), now fall on "natural" boundaries. This was not a requirement on the PM20, so certain kernel page tables started on non-page boundaries (two page tables were in one page). This will have to be redone, though it will be trivial.

TLB: (lio.h)

The mechanism for TLB flushing will need to be redone. Now we do

*FLUSH_TLB = 0 where FLUSH_TLB is 0xffe63000 (MTAG_FLUSH)

to flush the entire TLB on the PM20. On the 040 there is the PFLUSHA instruction, which will flush the entire TLB.

Likewise, the mechanism of clearing individual entries from the TLB, now done when flushing out a user's uarea on a context switch (swtch.c) and after handling a buserr on a user address (buserr.c), by resetting the valid bit at the uarea address in the TLB, can be done with the PFLUSH command. For example, to flush the uarea out we could do

```
mov.l   &1, %d0         ; select supervisor data space
movec   %d0, %dfc
mov.l   &ADDR_U, %a1
pflush  (%a1)           ; flushes ADDR_U from TLB
```

For both cases, either a flush of the whole TLB or just the flush of an address, new routines should be created for each processor type with a common interface, eg.

```
for PM20:
    #define flush_whole_tlb()  *FLUSH_TLB = 0
for DPM040:
    #define flush_whole_tlb()  asm("pflusha")
```

Another possibility would be to use inline procedures instead of defines. These are available in the current Motorola compiler. The above example would look like

```
asm void flush_whole_tlb()
{
        pflusha
}
```

and would appear in an inline.h file to be included by all the files in the kernel. This seems to be the direction that the kernel is going, based on inspection of the latest System V.3.2 from AT&T for the 386.

On the PM20, the TLB is also flushed on a context switch by setting the user root pointer. To set the user root pointer on the 040, a

```
mov.l   <new ptr>, %a1
movec   %a1, %urp
```

is done. It is unclear in the current documentation whether the TLB is flushed also. [In the 030, the PMOVE instruction

```
pmove   <new ptr>, %urp
```

flushed out the TLB. In the 040, this instruction was removed and MOVEC was extended to handle its functionality. The current documentation does not discuss flushing.] It seems that an explicit flush is necessary.

The TLB flushing instructions provide more control than the corresponding mechanism on the PM20, eg. flushing out user entries only by using the Global bit in the page tables for the kernel. For each TLB flush in the kernel, appropriate

action for the 040 should be taken, i.e. choose which variant of the PFLUSH instruction should be used.

Bus Error handling:

Formats $a and $b, used on the 68020 for bus errors, are no longer supported. A new format $7, for access error faults, has been added. The kernel routines (buserr.c) have to be rewritten for the new processor board architecture (eg. the general status register on the PM20 is no longer there for deciphering bus error information) and for the new handling of bus error exceptions by the Motorola processor. In particular, we would need to change

- lio.h for general status register (STATUS_REG & CLR_FAULTS)
- *CLR_FAULTS = 0
- state.h for the new format
- add new M68040 directory in the os for the new buserr.c

On the other hand, if we get V.4 for the 040 (there is no V.3.2) from Motorola, then we just need to port their buserr.c. The Motorola documentation is unclear exactly how to handle the buserr and the Software group at Motorola is not planning on working on this until November. In particular, there are certain write back data fields in the exception frame presumably for cache coherency, but how they should be used is not specified in the preliminary design specifications.

Transparent translation registers:

The 68040 has 2 sets of 2 Transparent Translation Registers, 2 each for data and instructions, for use in bypassing the TLB. One data register (DTT0) should be used for local io space and should be initialized by the firmware as follows:

```
logical base address    ff000000
logical address mask    00000000    (care about all bits)
E = 1   enabled
S = 01  supervisor accesses only
U1, U0  ignore for now
CM = 10 cache inhibited, serialized
W = 0   not write protected
```

which turns out to be 0xff00a040 for the initial value. Because the local io space is mapped via a Transparent Translation Register, no page tables need to be built for it. (The firmware could also set up ITT0 in the same range, except with the cache enabled, to speed up its execution during test and initialization.)

The usage of the other registers is TBS. However, it seems reasonable to use the other data register for some part of the system io space (piomap??).

Cacheability:

Additionally, we have 4 modes for cacheability, including whether reads can jump over a write (not cacheable, but either serialized or not serialized), so the kernel page tables must be set up appropriately by the SPM.

For the purpose of bringing up the system, the CACHE ENABLE bits in the Control Register for each CPU will be under software control, so that the primary and secondary caches can be enabled or disabled prior to initiating the kernel [as is done currently by the SPM for the PM20]. The page tables should still be built with the Cache Mode bits set to 00, for Cacheable and Write Through, regardless of the initial status of the caches.

4. Primary cache

There are 2 on-chip caches, data and instruction, each 4-way set-associative and 4K in size. The data cache is either write-through or copy-back; we will do write-through only (but could investigate copy-back, especially for stacks, at some later time).

According to the 68040 Design Specification, "the data cache is optimized for byte, word, and longword accesses; therefore, a misaligned access may result in 2 cache accesses." So, for performance we want strict alignment. This creates two problems in the kernel:

(a) We must find and account for misaligned structures, which especially occur in the areas of A1000 IO. This was already done for SPARC. In particular, the files involved are
   fs/s5filsys.h - for the superblock
   dtreq.h, icb.h, and vreg.h.

(b) We must find the incompatibilities between 020 binaries and the 040 kernel.
   - Of the structures passed to a binary from the kernel, 2 will be misaligned:
         stat structure returned by stat() and fstat(), and
         statfs structure returned by statfs() and fstatfs().
   - Signal may also return a misaligned stack to the user.

OTHER SOFTWARE:

SPM & Other Standalone:
---

1. The ability to support more than one CPU per board has been provided by the work that Mark Schultz, Joe Saunders, and Brent Leever have already done. The implementation is not complete though, because it seems to assume only one processor type, PM20, is supported, rather than PM20s and DPM040s. Additionally, certain commands on the menu will need to include subslots.

2. The SPM must be able to recognize the two new board types (a single cpu DPM040 has a module id of 0x22 and a dual cpu board is 0x26) and manage a mixed system of single and dual DPM040s.

3. For the M68040, the following changes need to be made:

   - The way that interrupts are sent to a cpu will change, because (1) the interrupt request register changed on the DPM040 and (2) there are now subslots to be accounted for [handled above].
   - The routines to walk pagetables will change due to the 3-level MMU, the different page table entry layout, and the addition of the memory map.
   - The initialization code for the system will change due to
     (a) need to start 2 cpus per board
     (b) need to build a memory map and have it available to the IOPMs
     (c) one processor per board must load up the memory map
   - The disassembler needs the new 68040 instructions.

   No significant changes are required in the other areas, eg. the debugger.

   The implementation of the memory map, new memory mapping routines, and the initiation of the kernel was done for the 486. The support for 3-level page tables was already done for SPARC. What is needed is to combine the two.

4. Because of the change in alignments, the superblocks on all the disks will change. The 'mkfs' routine must generate the new superblock rather than the superblock for a 68020. The need to change any other standalone routines is not known at this time.

IOPM:
---

1. The IOPM walks the kernel page tables. The IOPM group will need to modify their code to support the new memory management structures, including the memory map. The SPM and OS must provide whatever structures are necessary for the IOPM to be able to walk the page tables. This work was started for the 486, and we've been working with Craig on his requirements.

2. The IOPM directly interrupts a PM using its slot number and the fixed address for the interrupt request register (PM_INT_REQ_REG) on the board, for a level 1 IOPMRESPONSE interrupt. This will no longer work in a dual PM system. A new mechanism must be done, possibly based on a mapping of PMs generated by the SPM and made available to the IOPMs.

SGS:
---

1. A new magic number is required to keep the .o's from being combined and to help the kernel distinguish between S90 020 and 040 executables.

2. Assembler changes:

   The new 040 instructions must be added to the 030 assembler, and the obsolete ones removed.

3. Compiler changes:

   To take advantage of the performance improvement by correctly aligning data, the compiler must change to generate the new offsets.

   To support old alignments, the "pack" pragma from the System V/386 compiler should be implemented. The pragma is a directive to change the type of alignment the compiler will use on a particular set of data. If we have a structure containing

```
short   x;
long    y;
```

   the 020 compiler will start 'y' immediately after 'x', i.e. on a short boundary and the 040 compiler would place it at the next long boundary, adding 2 bytes of padding between 'x' and 'y'. Using the pack pragma, we can force the compiler to do short alignment via pack(2) and then have it return to long alignment via pack(4). The above example now becomes

```
#pragma pack(2)
        short   x;
        long    y;
#pragma pack(4)
```

   The compiler (in particular the optimizer) must be taught about the new 040 instructions, so that inline asm directives and inline asm procedures can include the 68040 specific instructions.

   The code generation and optimizer in the compiler should be redone for the optimized addressing modes and optimized instructions of the 68040.

   There may be a reason to generate the new floating point instructions added in the 68040 for extended precision.

   These changes must be done in-house, since Motorola will not be providing a 68040 compiler which generates COFF files.

4. Linker changes:

The linker no longer needs to round up text sizes to the next page when the text size is at a page boundary. This was put in to get around a problem on the PM20 of a prefetch causing a buserr.

5. Libraries:

   We should investigate the use of MOVE16 in the mem*() library routines.

   The emulation routines for floating point need to be included in libm.a.

   There may be some changes required for the changes due to the new alignment, in particular, the alignment of the signal stack.

6. Disassemblers:

   The new instructions must be added and the removed ones deleted.

Utilities:

For the non-machine dependent utilities, the 020 binaries could be used; however, they SHOULD be remade for the 040 for improved performance.

1. All disassemblers, eg. sdb, must be updated.

2. All places where the source has
   ```
   #ifdef M68020
   ```
   we must do
   ```
   #if defined(m68k)
   ```
   or
   ```
   #if M68020 || M68040
   ```

3. M68881 ust be set by default.

4. The alignment changes cause some problems in some utilities, especially those dealing with A1000 IO. These have been identified in the work done for SPARC. The addition of the pack pragma should obviate the need for source changes.

Diagnostics:
---

1. The initialization of the DPM needs to be redone. The state of the board and each cpu when the kernel takes over is yet to be specified.

2. The SPM builds the memory map in memory and passes to the firmware running on the PM board the slot number, starting location, and size of the tables. The firmware then fills the on-board memory map (although only once per cpu on the board). See "DPM040 Interface Between Firmware and OS/Kernel", by Shih-Nan Huang, dated 8/28/89.

Other Software:

---

1. The change in alignment might affect Ethernet, communications packages, and database packages. For example, Excelan's Ethernet assumes short alignment.

2. A standalone utility needs to be provided to convert the existing superblocks on S90s, since the alignment changes for the 040 causes the superblock to be incompatible.

3. To facilitate upgrades from 020 binaries to 040 binaries, a Porting Guide should be provided to handle alignment issues.

KERNEL changes:

OS DIRECTORY:

clock.c
    use common routine for TLB flush
    update reference to o_update_iomap and update_my_iomap()
    redo queueing of TIMEIN interrupt for self: writes to slot and PM20 address
    clkstart - scans through sbus_slot_id for SBUS_PM20s
disp.c
    p->p_running is set to own.o_slot; will o_slot now be (slot #, A/B)
    setrq - looks through list of pm_own up to SBUS_NUM_SLOT
exec.c
    recognize 040 magic numbers in codefile headers in gethead()
fault.c
    change in pte bits ? - move lock from dbd to pte
    use common routine for TLB flush
    redo flush_all_tlbs() and wait_for_tlbs_flushed()
fork.c
    redo initialization of segment (page pointer) table for child
getpages.c
    possible changes if we move dbd bits to pte
page.c
    usual memory changes
    redo TLB flush in sptalloc
physio.c
    current implementation uses ku_to_sbus_u32(), kv_to_sbus_u32(), and
        pg_sbusaddr. Do we need to change this?
probe.c
    possible changes if we move dbd bits to pte
    uses pg_sbusaddr - is this ok?
region.c
    3 level memory changes
    implement sharing via higher level page tables
    move bits from dbd to pte
    change TLB flush in loadreg
    redo loadstbl (already done for SPARC)
    remove A1000 support via ifdef's ???
sched.c
    uses p->p_running to set pm_own[p->p_running]->o_runrun - if o_slot
        is now (slot #, A/B), then it can't be used as an index. So
        o_slot & p_running need to be rethought
sig.c
    ifdef out test on COP_MIDINSTR - this doesn't occur on the 68040
    check out potential alignment problems
    may have some ptrace changes, especially due to the way breakpoints
        and interrupts are handled on the 040
    change TLB flush in procxmt()
space.c
    add any necessary memory management stuff
startup.c
    startc()
        redo o_slot for initializing upkern.up_slot ??
        possibly redo call on pm_int_req_reg_init
        change TLB flush

```
    mlsetup()
        potential memory map changes
        kernel layout changes
        implement 3 level tables
        change TLB flush
    pm_init() - may be a total rewrite
        uses up_slot
        remove PM20 references or update:
                *GREEN_LED = PM_LED_ON
                *PERMIT_FLT_ENBL = 0xff
        handle calls on
                iomap_init()
                pm_cache_on()
                cpu_get_type()
                fpu_get_type() - remove
                clear_pm_int_req_reg to clear any pending interrupts
    pm_cache_on()
        REWRITE
    sysseginit()
        add any changes for 3 level memory
        handle going through sbus_slot_id for building own_sptr's
    mktables()
        possible changes for 3 level MMU
    p0init()
        p_running is set to o_slot
    startup()
        goes through sbus_slot_id to find SBUS_IOMs
swtch.c
    swtch_continue()
        redo flush of uarea from TLB
        change references to LEDs on PM20
synch.c
    redo stop_all_processors() because scans through spm_mem.sbus_slot_id
        looking for SBUS_PM20s and sending them a level 7 interrupt
    check use of up_slot in upkern
    upkern_level_one() and upkern_level_four() send level 1 and 4 interrupts
        to spm_mem.upkern.up_slot
sys3.c
    handle alignment problems in stat(), fstat(), statfs(), and fstatfs()
        for 020 binaries
sys4.c
    verify sending set_time interrupt to spm via dispatcher is still valid
sysarix.c
    verify sending power margin interrupt to spm via dispatcher is still valid
text.c
    3 level memory changes ??


IO DIRECTORY:

ints.c
    redo sending interrupts to other PMs
pm_iomap.c
    no longer is valid - replace
```

ML DIRECTORY:

userio.s, useriorfs.s
    use move16 wherever possible
    do movs in copyin/copyout
scb.s
    add floating point exception at 55 and remove obsolete ones
trap.s
    support new exceptions and redo interrupts
    remove trap 2 for Oracle - locking AND UNLOCKING now done by CAS's
        in Oracle libraries for the S90 (still used by A1000 binaries,
        but they will not be supported on the 040).

SYS DIRECTORY:

fs/s5filsys.h
    alignment problem in struct filsys (superblock)
dtreq.h
    alignment problem in struct pd, use pack(x) pragma
icb.h
    alignment problem in struct icbcmdhdr, use pack(x) pragma
lio.h
    move to machine dependent directory, rewrite
state.h
    add new exception stack frames
sysmacros.h
    add alignment macros (use SPARC ones)
vreg.h
    alignment problem in struct bd_desc, use pack(x) pragma

CURRENT MACHINE DEPENDENT DIRECTORIES:

os/M68020:

    buserr.c
        need totally new one
    fpp.c
        rewrite to support new state identification (versions and formats)
    machdep.c
        rewrite, except for backtrace
    tdb.c
        should be the same, except for turning the on-board cache off
            in TDB_CMD_CHECK_TAGS
    trap.c - any changes could just be ifdef'ed
        add new F-line handler for the floating point emulator
        remove obsolete traps

sys/M68020:

    immu.h
        rewrite, but some should be restructured to save common elements
            with PM20
    kmem.h

rewrite
psl.h, reg.h, spl.h
no changes
trap.h
slight modifications for new floating point traps, etc.

GENERAL MMU CHANGES

Where possible, an estimated number of days work is given in the left column. "S" in the left column denotes the work has been already done for sparc.

include changes (TOTAL: 5-7days)

2d . immu.h -
S . modify to include the third level of table. (done)
. modify defines to describe table sizes.
. modify descriptor structures to reflect bit fields of the PMMU.
1d . modify macros to manipulate bit fields of pmmu.
2d . kmem.h -
. modify memory layout to reflect new table sizes and cache alignment.
. vmem.h -
. some addresses change depending on user size.
S . own.h -
. add a pointer to the "private kernel Ssegment table"
. param.h -
. NCPS & CPSSHIFT are mmu dependent. If they could move to another file like immu.h, or be made to symbolically be set according to constants in immu.h that would be best, otherwise, dont' forget to change them.
S . pfdat.h -
. pf_use becomes a ucnt_t type; that is and unsigned short.
S . spm_mem.h -
. add an invalid_ssde entry.
S . types.h -
. add ucnt_t type (unsigned short) for pf_use (alternative?)

iopm changes

5d? . If the iopm accesses page descriptors without the use of macros there may need to be changes. Also, there may need to be changes due to the lack of mode bits.?.
(I talked with craig. He will do any necessary mods. and is trying to reduce any machine dependencies.)

spm changes (TOTAL 6d)

. General
. address conversion routines need to be modified for new iomap scheme.
. Now Map memory that will go through transparent translation registers.

S .dbug/kdebug.c -
. Rewirte vtop() to include third level of mmu (simplifies routine).
S . local/sbus.c -
. Set Spm_Mem->invalid_ssde & send ssde value as root ptr, not sde.
6d os/pagetable.c -
. In general, the routines should be "genericized". Much of this has been done for sparc, already. (there were some areas where sizes were hardwired, or were defined as one thing that should have been another. No big deal).
. PM_own() still needs some work.
. ssde_to_km(), km_to_ssde() needs to be added.
. [psr]de_to_km(), km_to_[psr]de need to be genericized and the pde routine needs to use some sort of macro or routine to set the mode.
. the underlying *pfn conversion routines need to use the new mapping scheme to encode a physical address into 20 bits (not 24) of the pde.

. PM_bad_tbls() must now include building a bad sstbl.
. PM_stbl() should now build both the sstbl and the required stbl's.
. PM_own() needs to be updated to fill the proper tables and lay them out in memory. (This still needs some cleaning up).

kernel changes (TOTAL: 15d)
. Most of the work has been done to add the segment
. local io -- transparent translation
. loadstbl -- needs some major cleanup
. specific routines to allocate/free the upper level segments.
. Modify routines to use pmove, ptest(to look at status), pflush and to use the pmmu status registers
. address conversion routines need to be modified for new iomap scheme
. Set up of transparent translation registers and enable them.
use for upper 4 Mb of address space. (local io)
can it also be used for iopm space, or gc maps?
. How to write findpde? Do we walk ourselves or is there a command?
. When you turn on the cache, does this also turn on 2ndary cache?
. files
2-4d. ml/M68040/mltrap.s -
. are there changes to any findpde, or references to pde bits?
2d . os/fork.c -
. clear out sstbl (not stbl) on process creation.
. uballoc/ubfree must be rethought. it is no longer allocating a page size root (sstbl) table (it is now 512 bytes). it needs to return an rde, it is kludged right now. maybe it could return something like

sstbl	512
upage	1k
stack	??

where sstbl and upage are on same physical page. (problem?) upage would no longer probably be on a page boundary???
S . os/page.c -
. ASSERTS and otehr tests on pf_use must not be signed. pf_use is now an unsigned short. So tests like (pf_use>0) become (pf_use!=0).
. There is one spot where "mode" is OR'd into a long. Since modes are now encoded into the remaining bits something like |= pde_mode(mode) must be done, where pde_mode makes a pattern for the appropriate bits according to the mode given.
. os/region.c -
S . SEOFFMASK define uses 0x200000. This should be PNBPPT (or whatever means the number of bytes pointed to by a pagetable).
S . test a value returned by loadstbl denoting success or failure. (loadstbl may fail now since it may need to allocate a table).
. you may want to move loadstbl to machdep.c since it may differ greatly from the much simplified two level version.
. os/startup.c -
S . you may need to set the root process root pointer u.u_procp->p_urde. (depending upon how the hw user & supervisor root pointers are used)
S . at the spot where you loop through the sysseg tables, referencing ..._stbl, you must now reference sstbl.
. os/buserr.c -

2d . usrxmemflt() - a stack probe may fail @ mid or upper table level, not necessarily at the page level. rewrite to accomodate and grow stack properly.

S . rewrite findpde to add another level and potentially to denote failure at an upper level table.

. os/M68040/machdep.c -

see spm . logical <-> physical address translation routines must be modified to accomodate a different mapping since there are only 20 bits in the pde (not 24) for the physical address.
These routines are common between the spm and the kernel. It would be nice to break these out into a separate file that could be shared by the two different programs. In this way, only one file would need to be maintained.
Other routines are necessary: pde_mode() (if it can't be a macro).

4d . loadstbl() needs a major overhaul. It can potentially (de)allocate an intermediate table (stbl). So, it must be structured to return failure if it can't allocate the table. Also, ptalloc has been used in the past to allocate these tables. Is this a problem? Should it have its own set of manipulation routines? I have coded this routine for sparc and it could be used as a base for the 040 loadstbl, but it should be reviewed and cleaned up.

. io/icb.c -

5d? . it seems that whatever mods are made to the 020 for iopm space will translate easily to the 040. If iopm space is to be transparently translatable, it is best if it sits next to the local io space for each board, that way the transparent translation register can be made to define just that much bigger space.

. M68040/vuifile -

. modify it to get its origin and length values from some include file (this may take some sed'ing) so that it does not need to be modified by hand with each change of kmem.h.

. binary compatibility (see stone)

. I don't know what problems there may be here. The segment size is much smaller than that for the two level table, so a1000 and s90 text/data segment boundaries should not be a problem.

. Shared text?

. I didn't have any problems with this in the sparc implementation, but.