



gc_vm

Joe Valenzuela

6/15/07

A virtual memory system for NGC

- Will cover implementing paging on 32-bit PowerPC
- Use of Segment Registers to setup page mapping
- Maintenance of page table and gotchas
- Predates VirtualMemory library

Glossary

- Physical Memory
 - Directly addressable without hardware support
- Virtual Memory
 - Directly addressable only with hardware support
- Block Address Translation
 - Address translation provide by BAT registers.
- Page Fault
 - An exception that occurs when no mapping exists between virtual page and physical page
- DSI/ISI (Data Storage/Instruction Storage Interrupt)
 - Exceptions caused by attempts to access invalid effective addresses

Glossary (cont)

- PTE: Page Table Entry
 - 64-bit data structure mapping a physical page to virtual one
- PTEG: PTE Group
 - A group of 8 PTEs ('valid' or 'invalid'), any one of which is a potential candidate to store a mapping
- MMU: Memory Management Unit
 - Chip which handles translating effective addresses to physical ones
- TLB: Translation Lookaside Buffer
 - MMU cache which holds recently used PTEs
- Working set:
 - Set of pages a process is currently using

32-bit PowerPC Virtual Memory

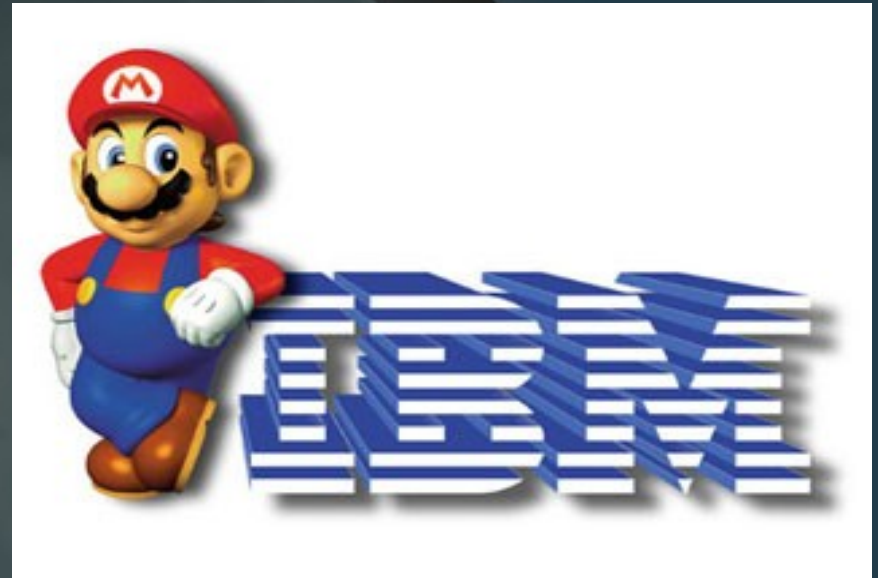
- Virtual address spaces are setup using Block Address Translation (BAT), Segment Registers (SR), or both
- BAT registers allow translation of 256MB blocks of contiguous address space at a time
 - 0x80000D80 -> 0x0D80
 - 0xC0000D80 -> 0x0D80 (uncached)
- Segment Registers allow page level mapping, as well as protection and cache controls (and IO segments)

Paging

- 4k page size
- The page table is maintained by client code, in main memory, and cached by the TLB
- Use of the Segment Registers requires setup and maintenance of page table and creating DSI/ISI exception handlers

Nintendo Gamecube

- PPC 750CXe derivative
- Partitioned Memory
 - 24MB “Fast” 1TRAM
 - 16MB ARAM (PC133?)
- ARAM not directly addressable, must DMA
- Flipper, DVD controller expect physical addresses



- Dolphin OS exclusively uses BAT

The Problem

- GC has tiny amount of main memory
- You thought better texture and mesh formats would help!
- To compete with other platforms, need to increase general purpose memory



The solution

- Lovely ARAM
- Create new address space with 2MB of fast ram backed by 12MB of ARAM
- Require paging to be setup using SRs
- Could explicitly DMA data in/out
 - Requires extensive code modification
 - Tough to maintain
 - Insufficiently hardcore

Initialization

- SR_n
- SDR1
- MSR

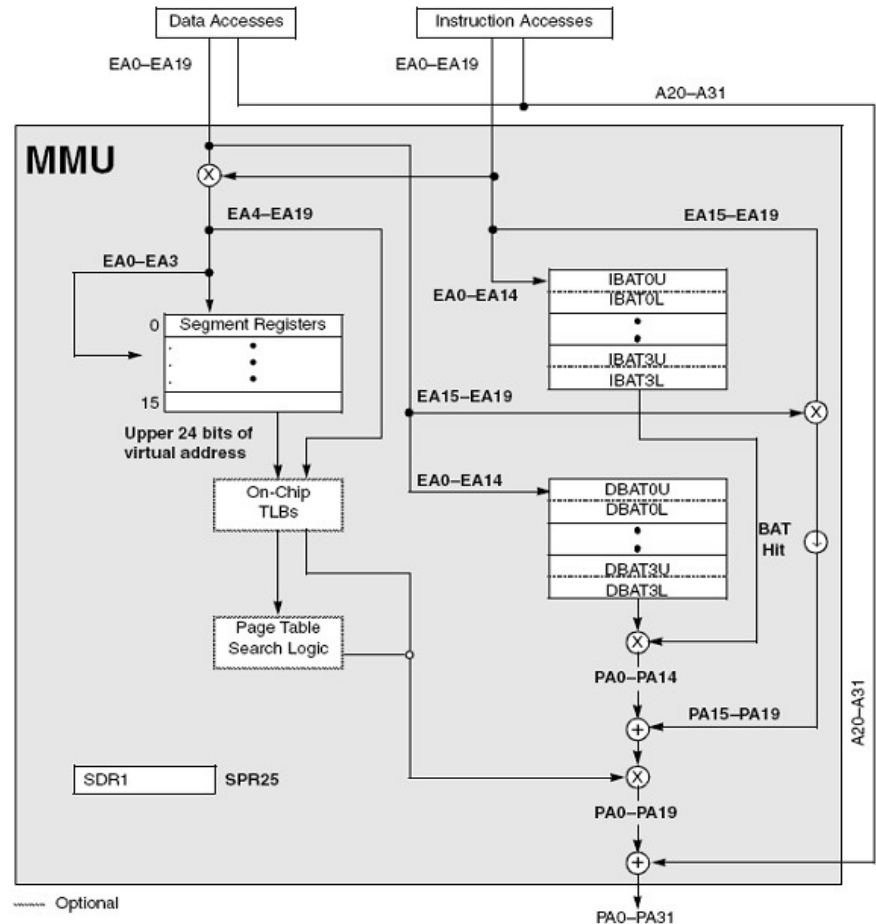
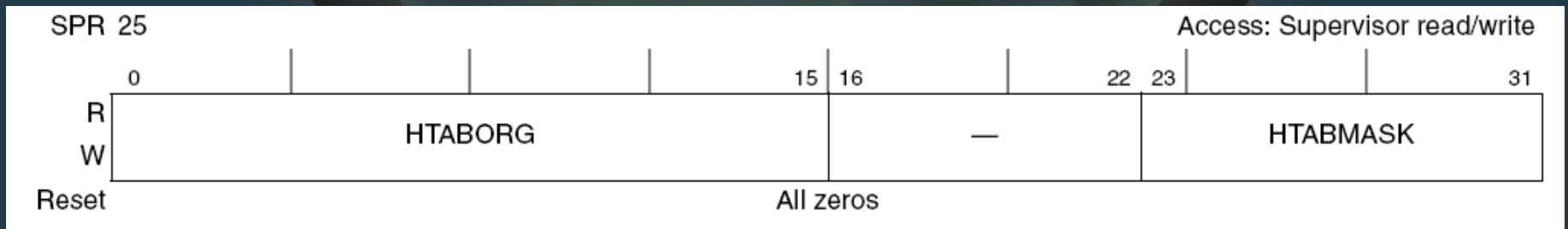


Figure 7-1. MMU Conceptual Block Diagram

- Size and location are set via SDR1
- HTAMASK defines the mask for the page table address
 - $htamask = (tbl_size / 64k) - 1$
- HTABORG is the top 16 bits of 32bit physical address
- Must reside on address boundary divisible by size



```
vm_tbl = memalign(tbl_size, tbl_size);
mtdsdr1 PHYSICAL(vm_tbl) | htamask;
```



Page Table Usage

- When searching for a mapping, the MMU hashes portions of the effective address and VSID from the appropriate SR to get a hash. Page Table is indexed using hash to get primary PTEG.
- The 8 entries of the primary PTEG are searched first. If no match is found, secondary PTEG is searched using secondary hash key.

Oh Noz, Codz

```
n = (ea >> 28) & 0xF
vsid = SRn & 0x7ffff
page_index = (ea >> 12) & 0xffff
primary hash = vsid ^ page_index
secondary hash = ~primary hash
```

```
base = sdr1 & 0xFFFF0000
index = ((hash & (htamask << 10)) |
        (hash & 0x3FF)) << 6
PTEG = base + index
```

Generate primary and secondary PTEG

Search primary PTEG[0-7], secondary PTEG[0-7]

- A hit occurs when all of these are true:
 - $\text{PTE}[\text{VSID}] == \text{SR}[\text{VSID}]$ // correct segment register
 - $\text{PTE}[\text{API}] == \text{EA}[\text{API}]$ // correct page index
 - $\text{PTE}[\text{V}]$ // PTE is valid
 - Primary/secondary hash bit matches
- If no mapping occurs we have a page fault, and our DSI or ISI handler is invoked
- Happens at least once the first time an effective address is accessed.
- Handler must duplicate search performed by MMU to put PTE in right slot

Data Storage Interrupt

- DSISR is set to exception status
 - 0x40000000 for page table miss
 - It's possible to get a DSI for non virtual memory paging situations
 - Protection status
 - Direct Store segments
 - Not covering any of this
- DAR is the faulting VM address
- SRR0 is the faulting instruction
- Search Page Table duplicating the MMU logic (primary/secondary hash & PTEGS)

Unused PTE found

- Allocate physical page of fast ram (RPN)
- New PTE:
 - Set RPN and API (from effective address)
 - Set VSID (from SR)
 - Set valid bit
 - Set Primary/Secondary hash bit
 - Ignore R & C bits for now
- tlbie and sync

No free PTE found

- Have to select a used PTE, delete it, and flush its page
- Each effective page has an ARAM page associated with it
- Changed pages get DMAed back to ARAM
 - Used simple mapping
 - EA & 0xFFF000 -> ARAM page
- Delete PTE by setting $PTE[V] = 0$ and tlbie
- Try again

Free PTE but no free physical page

- The typical case
- Working set for games is **always** bigger than physical pages
- Maintenance of R&C bits and page replacement policy govern performance
- Maintenance of R&C bits and page replacement policy govern performance
- Basically handled same as “No free PTE found”



rfi

- Return after DMA is handled and PTE set
- Control returns to instruction in SRR0
 - We want to retry the faulting instruction, so don't touch it
- tlbie to invalidate the TLB's conception of a PTE
- eieio
- tlbsync(?)
- sync

Instruction Storage Interrupt

- Similar to DSI, except instead of DAR we use SRR0
- Invalidate icache: icbi
- sync: isync

Machine State Register (MSR)

- MSR[IR] and MSR[DR] = 1
- mtmsr
- Needed to “turn on” address translation
- You can handle just faulting instructions, or just data if you want.
- Reset this from your exception handler if you want to use virtual addresses in your DSI handler, but you run the risk of an implicit branch and changing your code stream! Don't cross the code streams!

Performance and Maintenance

- Working set is *always* *always* bigger than available fast ram
 - Worst case exception incurs two 4k DMA transfers, sync costs, at least ~500 cycles, L1 cache eviction
- Data locality is important
 - Hard to enforce on cross platform game
 - No one else on team understands your woes
 - Did I mention the PS2 has an 8k direct mapped I-cache
- Store changed pages
 - Commit pages back to ARAM, clear C bit
 - Cleans pollution due to fresh data reads
- Still need to reduce size of working set



Page replacement

- Used a modified version of Tanenbaum's hierarchy of pages (NRU)
 - Class 0: not referenced, not changed
 - Class 1: not referenced, changed
 - Class 2: referenced, not changed
 - Class 3: referenced, changed
- Need to maintain age information to reward locality
- Requires that the PTE[R] bit is cleared
 - Typically each frame

WIMG

- Use $I=1$ to turn off caching behavior
- Use $W=1$ to change write-back to write-through
- Use $G=1$ to turn off speculative prefetching
- Default $W=0, I=0, G=0$ is set for maximal performance

So... what's that new address space useful for?

- Plug into dlmalloc
 - Keeps bookkeeping info in VM blocks
 - Block info interleaved with heap means more page transfers
- Still not totally hands off
 - Disparity in performance between fast RAM/ARAM meant tweaking to find right data to put in new address space
 - Could have saved a lot of effort by optimizing for data access
 - Added heuristics to mashing/packing pipeline to effectively use ARAM (“aramitize”)
- Complementary async system for texture data
 - Operated independently of vm system
 - Stream texture image data from ARAM
- Sufficient for SM2

Eventually needed more

- On USM, the need for more memory compelled us to put ELF in ARAM
- Maintained custom linker script to relocate hand picked code sections to another address space (0x70000000)
 - Old address space (0xF0000000) outside the 28-bit branch relative space of normal ELF (0x80000000)
- Had to fixup broken SN linker in offline tool
 - Didn't recognize different Load/Memory addresses
- Rewrote debugging stub in order to debug executable
- Performance critical code compiled with optimizations and put exclusively in fast ram, all other code put in ARAM and compiled for minimal executable size
- Finally the bloated ELF was a competitive advantage for the Gamecube!

Conclusion

- Data locality is important
- Don't write bloated code
- Freely available PowerPC docs are really good (Motorola 32-bit Programming Environment)
- USA! USA! USA!

