ECE 531 – Advanced Operating Systems
Lecture 23

Vince Weaver

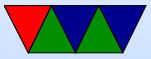
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

31 October 2025

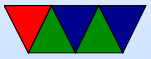
Announcements

- Don't forget HW#6



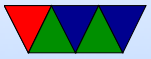
TLB Maintanence

- In theory the page-walker will automatically update the TLB entries for you
- If you update a page's info (physical mapping, permissions, etc) you need to invalidate the pages in the TLB
- Sometimes you can flush individual mappings, sometimes you have to flush entire TLB
- If your TLB has ASID mappings you need to manage those too, as there might be more processes than ASIDs



available

- You might need memory barriers when updating the TLB too, otherwise on OoO processors the loads might bypass/run ahead of the changes to modify the TLB

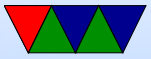


ARM 1176 (ARMv6) MMU

We'll discuss the setup and use of the MMU in the ARM1176 processor.

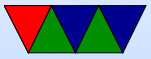
Note that later processors found in other Pis are not necessarily compatible with this.

The appendix to these slides have some notes on ARMv7 MMU



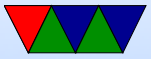
ARM 1176 (ARMv6) MMU

- See Chapter 6 of the ARM1176 Technical Reference Manual `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf`
- This blog post is also very useful `https://www.snaums.de/informatik/mmu-on-raspberry-pi.html`
- The ARM1176 MMU has a backward-compatibility mode to ARMv5 and ARMv4; this is often used for its simplicity

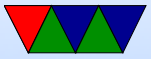


ARM 1176 TLB Summary

- micro-TLB
  - two 10-entry TLBs (one instruction, one data)
  - Fully associative
  - 1-cycle access, ASID
- Unified TLB
  - Accessed when micro-TLB misses
  - 64 entries, two-way
  - lockdown region of eight entries (for secure code?)
- you can mark entries as a global mapping, or associated

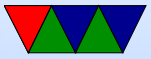


with a specific application space identifier to eliminate the requirement for TLB flushes on most context switches



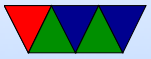
ARM 1176 – Page Tables

- hardware page table walks
- page sizes: 4KB, 64KB, 1MB, and 16MB
- you can specify access permissions for 64KB large pages and 4KB small pages separately (Subpages)
- access permissions extended to enable Privileged read-only and Privileged or User read-only modes to be simultaneously supported
- separate Secure and Non-secure entries and page tables
- Enable MMU by writing to special register in CP15



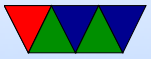
ARM 1176 – Caches

- Caches can only be enabled if VM enabled (why? because needed to mark memory as non-cachable)
- L1 instruction cache (32kB?)
- L1 data cache (32kB?). Aliasing if VM enabled (bits 12:13). Need to do page coloring, only use 4kB pages, or limit cache size to 16KB
- TCM (tightly coupled memory) small area up to 32kb that can map to arbitrary page location. Fast, cache speeds. Useful for FIQ handlers in real time systems.



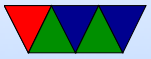
ARM 1176 – Mutexes

- It turns out the ARM LDREX/STREX instructions uses for mutexes also only work if VM enabled



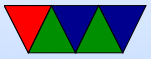
ARM 1176 MMU – Low Level Interfaces

- We'll be considering the ARMv6 legacy mode where the pagetables are treated like ARMv4/ARMv5
- VMSAv6 – virtual memory system architecture



ARM 1176 MMU – CP15

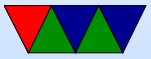
- ARM has notion of "co-processor" registers. The MMU is coprocessor 15
- Use special `mcr` (move co-processor register) to configure it
- `mcr p15,0,r0,c2,c0,0` – p15 = coprocessor 15, r0 is the src register to copy from, c0,c2 are sub-registers



ARM 1176 MMU – TTBR0 register

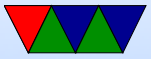
- The Translation Table Base Register (TTBR) registers point to page tables
- In compat mode when n==0 (?) TTBR0 points to page tables in the backward compatible fashion
- This is set with CP15/register 2

```
mcr p15,0,r0,c2,c0,0
```

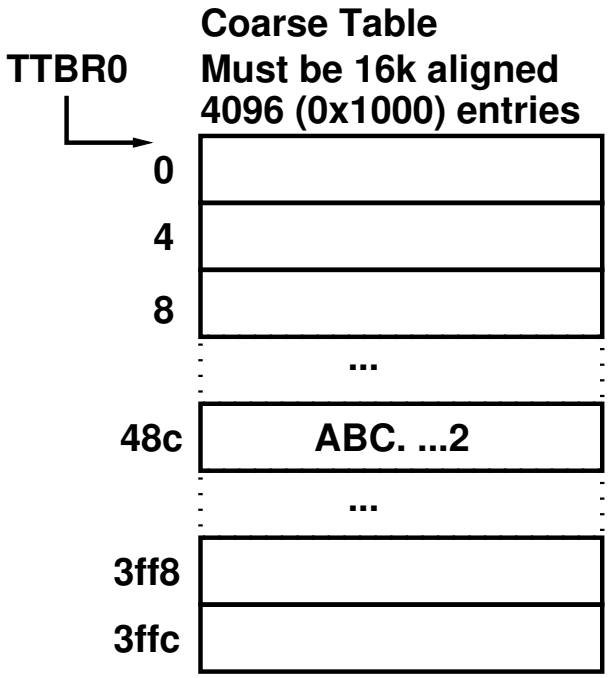


ARM 1176 MMU – Coarse Pages

- Memory split up into 1MB sections
- There are 4096 of these in a 4GB address space
  table with mapping is 16k in size
- Must be allocated in a 16k-aligned section of memory (bottom 14 bits must be zero)
  - How would you allocate a chunk of memory like that?
- These coarse pages can be joined up into super-sections (16MB groups)

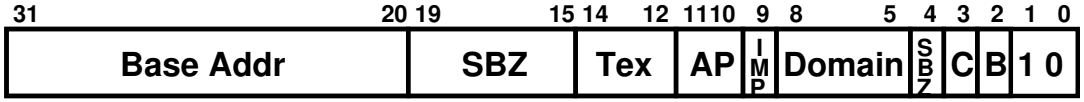


Coarse Pages Diagram

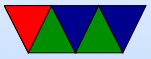


Example:
load from virtual address 0x12345678

first 12 bits = 0x123
multiply by 4 get 0x48c
look at offset 0x123 (0x48c) bytes into TTBR0
translation to physical addres is 0xABC
so the physical address loaded from us 0xABC45678

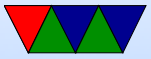


Coarse descriptor format

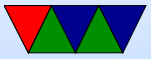


Coarse Pages Descriptor Info

- Base address – has virtual to physical mapping for this 1MB region
- SBZ – should be zero
- Tex – fancy complex caching stuff, too complex to describe here. Can also be turned over to OS for own use
- AP – access permissions
  - 00 – no read / no write, always fault (invalid)
  - 01 – read/write for privileged (kernel)

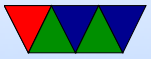


  - 10 – read/write privileged, r/o userspace
  - 11 – read/write everyone
- Domain – can set up 16 domains in a different domain register
  - 00 – no access
  - 01 – client
  - 10 – UNPREDICTABLE
  - 11 – manager (not checked) (usual default)
- C – cachable (when might you disable this? mmio?)
- B – buffered (slow writes allowed)

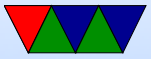


Setting up 1:1 Coarse Map

- Setup region aligned 16k
- Setup 1:1 virtual/physical mapping
  - Virt and Phys base are same each entry
  - If no physical RAM there, invalid
  - If kernel there, mark r/w privileged only
  - If mmio region, mark non-cachable
- Invalidate the dcache `mcr 0,c7,c7,0`
- Invalidate the TLB `mcr 0,c8,c7,0`
- Data barrier DSB `mcr 0,c7,c10,4`

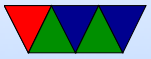


- Set all domains to 11, 0xffffffff to `mcr 0,c3,c0,0`
- Point the TTBR0 to the coarse table `mcr 0,c2,c0,0`



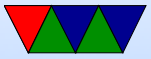
Actually Enable MMU

- Write to `mcr 0,c1,c0,0`
- Config bits:
  - 0: M = Enable MMU
  - 2: C = Enable L1 data cache
  - 11: Z = Enable Branch Predictor
  - 12: I = Enable L1 Instruction cache
  - 22: U = allow unaligned memory access



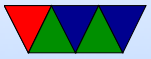
What happens to the code executing when MMU switched on?

- If we have 1:1 mapping doesn't matter, as the virt and phys the same
- If otherwise were turning it on we'd have to be careful the mapping is set up right so that the next instruction loaded at the now-translated address is one we want

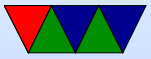


What if we want more traditional 2-level pagetables?

- Still set up coarse (first level) page table
- If the coarse entries end in 01 it means it points to a fine (second level) page table
- The top bits 31:12 used to point to a 4k "fine" table that holds 1024 entries
- The "fine table" has translation to the final memory location
- The "fine table" has four AP values, the 4k table can

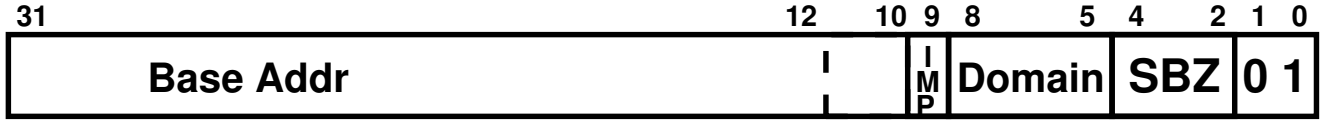


have individual permissions on 1k chunks

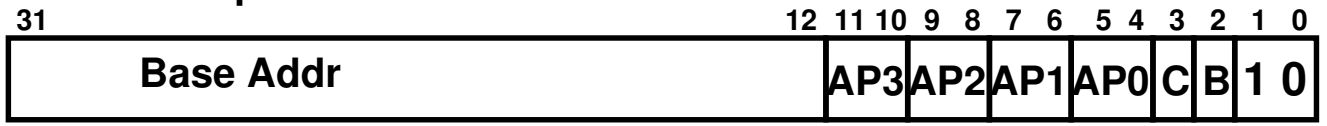


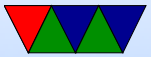
Fine Translation Diagram

Coarse descriptor format



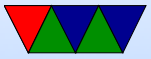
Small Descriptor format



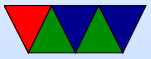


If we get page fault, how does OS know what address caused the fault?

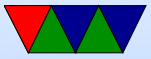
- Data Fault Status Register `mcr 0,c5,c0,0` says what kind of fault
- Data Fault Address Register `mcr 0,c6,c0,0` says what address is trying to be used
- Can use this to see if it should be valid or not
  - If is, then allocate descriptor and update the pages
  - If not, segfault, possibly print the source of the error for debugging)



- Instruction Fault Register `mcr 0,c5,c0,1` if an instruction fetch caused the page fault

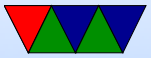


Appendix



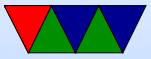
ARMv7 Virtual Memory

- ARM virtual memory is *really* complicated
- It's a lot more complicated than x86
- TODO, write this up better



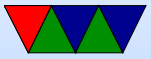
ARMv7 Page Tables

- ARMv7 supports two pagetables, one for kernel-type thing that's fixed and always there, one for process that you can swap in/out
- Pagetable has lots of fields
  - Address (31-20)
  - NS - not secure
  - nG - not global
  - S - shared
  - AP[2:0] access permissions (kernel r/w, kernel r/o,



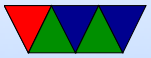
anyone r/w, anyone r/o, no access)

- TEX - caching (cacheable, no cache)
- Domain - up to 16 domains with different checking permissions



ARMv7 VM on our OS

- Our OS we set up a 1:1 Virtual to Physical "Section" Mapping with 1MB pages
- Set up a pagetable, 4k table that is 14-bit aligned
- Setup pagetable, point to it
- Setup domains (want 0x55555555 not 0xffffffff or won't check)
- Flush TLB/Caches?
- Enable MMU



ARMv7 Caches

- Caches are small, fast memories that mirror parts of DRAM for speed
- Important for performance, really hard to set up on ARMv7