Advanced Compilers and Architectures

ARM MMU Overview

Mattias Holm, M.Sc.
holm@liacs.nl
Outline

- Memory Management Overview
- ARM Overview
- ARM MMU Specifics
VM and MMU
Memory Management

• Needed to protect applications from each other.

• Necessary if an application requests more memory than is physically available.
Memory Management

• Two Major Technologies:
  • Segmentation
  • Paging
Segmentation

• Processor tracks start and length attributes for memory segments using registers.

• Each segment has access attributes:
  • Read, write, execute et.c.

• Violation of attributes results in a segment violation or segfault (UNIX SIGSEGV).
Paging

- Introduced to allow memory to be swapped out to disk.
- Memory divided into pages of fixed size (usually 4 KiB).
- Memory pages specified using page tables.
- Pages have access attributes like segments.
- Has mostly replaced segmentation.
Paging

• Page Table Pointers (PTPs) identify page tables.

• Page Table Entries (PTEs) map virtual to physical address and track page attributes.

• Translation Lookaside Buffer (TLB) caches PTEs for quick access.

• If an entry is not in the TLB, memory system will do a page table walk.
Paging

- Table Walk

  - Processor performs a load or store to an address that is not in the TLB (assume address is 0x10201234).

  - Processor uses the page table pointer (stored in a special register) to find the page table.
Paging

- Table Walk

<table>
<thead>
<tr>
<th>Root PTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x 10 20 1234</td>
</tr>
</tbody>
</table>
Paging

- Table Walk

- Processor extracts the high bits of the virtual address and loads PTP from L1 table.

Root PTP

0x 10 20 1234
Paging

- Table Walk

- Processor extracts the high bits of the virtual address and loads PTP from L1 table.
Paging

- Table Walk

- Processor extracts the high bits of the virtual address and loads PTP from L1 table.
Paging

- Table Walk
  
  Processor extracts the high bits of the virtual address and loads PTP from L1 table.
Paging

- Table Walk

- Processor extracts the high bits of the virtual address and loads PTP from L1 table.
Paging

- **Table Walk**
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
Paging

- Table Walk

  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.

  - Processor uses mid bits to load the L2 PTE.
Paging

- **Table Walk**
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.
Paging

- **Table Walk**
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.
Paging

- Table Walk
  - Processor extracts the high bits of the virtual address and loads PTP from L1 table.
  - Processor uses mid bits to load the L2 PTE.
  - Use the PTE and lower bits to compute the physical address.
Page Allocation

• How do we allocate virtual memory?
• How do we allocate physical memory?
Allocating virtual addresses

- Each process has its own VM table.
- Process associated with a sorted linked list that track the allocated VM blocks.
- Can place allocated page list in a balancing tree for faster search.
Allocating physical addresses

• Kernel needs to track which physical pages has been allocated.

• Needs a list of free pages.

• Linux has lists of $2^n$ sized blocks that are free. If a block of power $a$ is requested, but does not exist, split block of power $a+l$. 
Allocating space for the VM structures

- Cannot use the kernel’s generic VM code to allocate space for VM structures (*turtles all the way down*).

- Break the chain of recursion by special casing the VM structure allocation.

- Steal one physical page and use this for storing VM structures, and to describe itself.
ARM
ARM Overview

• ARM Developed by Acorn Computers Ltd in the UK, ARM 1 released 1985.

• Acorn + Apple worked on new chip design for the Newton PDA, ARM 6 released in 1994.

• Processors are licensed, not manufactured.
ARM Overview

- Used in around 98% of all mobile phones.
- Has around 90% of the embedded processor market.
- Very power efficient.
## ARM Overview

<table>
<thead>
<tr>
<th>Family</th>
<th>Architecture</th>
<th>Core</th>
<th>Chips</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM1</td>
<td>ARMv1</td>
<td>ARM1</td>
<td>ARM1</td>
</tr>
<tr>
<td>ARM6</td>
<td>ARMv3</td>
<td>ARM60, ARM600, ARM610</td>
<td>ARM60, ARM600, ARM610</td>
</tr>
<tr>
<td>Cortex-A</td>
<td>ARMv7-A</td>
<td>Cortex-A8, Cortex-A9</td>
<td>OMAP3xxx, Apple A4</td>
</tr>
</tbody>
</table>
ARM Characteristics

• 32 bit architecture
• Load-store architecture (RISC)
• Normally little-endian, but may vary between processors
• 16 GPRs (r15 = pc)
• Multiple ISAs (ARM, Thumb, Jazelle)
Modes and Registers

- ARM processor banks registers, depending on mode.
- USR: User applications
- SYS: System mode, with access to USR registers.
- SVC, ABT, UND, IRQ: banks r13-r14
- FIQ: banks r8-r14

<table>
<thead>
<tr>
<th></th>
<th>USR</th>
<th>SYS</th>
<th>SVC</th>
<th>ABT</th>
<th>UND</th>
<th>IRQ</th>
<th>FIQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r8_fiq</td>
<td></td>
</tr>
<tr>
<td>r9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r9_fiq</td>
<td></td>
</tr>
<tr>
<td>r10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r10_fiq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r11_fiq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r12_fiq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r13 (sp)</td>
<td>r13_svc</td>
<td>r13_abt</td>
<td>r13_und</td>
<td>r13_irq</td>
<td>r13_fiq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r14 (lr)</td>
<td>r14_svc</td>
<td>r14_abt</td>
<td>r14_und</td>
<td>r14_irq</td>
<td>r14_fiq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r15 (pc)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cpsr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>spsr_svc</th>
<th>spsr_abt</th>
<th>spsr_und</th>
<th>spsr_irq</th>
<th>spsr_fiq</th>
</tr>
</thead>
</table>

Thursday, February 24, 2011
ARMv7

- ARMv7 comes in 3 variants:
  - ARMv7-A with MMU (paging).
  - ARMv7-R for hard realtime applications with MPU (segmentation).
  - ARMv7-M micro-controller version, no memory protection.
ARMv7-A/R

- VMSA - Virtual Memory System Architecture
- PMSA - Protected Memory System Architecture
- Our kernel runs on ARMv7-A (Cortex-A8)
ARMv7-A MMU
ARMv7-A MMU

- Control using coprocessor 15 and mrc + mcr instructions.
- 2 level page tables.
- Page sizes: 4 KiB, 64 KiB, 1 MiB and 16 MiB
- Permissions for supervisor and user (read / write).
- No-Execute (NX) bit
# ARMv7-A MMU

## Level I Table Entries

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault</td>
<td>Page table base address, bits [31:10]</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>Section base address, PA[31:20]</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>Supersection base address PA[31:24]</td>
<td>00</td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Domain</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>Domain</td>
<td>XN C B</td>
</tr>
</tbody>
</table>

**Legend:**
- **S**: Supervisor
- **N**: Normal
- **B**: Supervisor
- **Z**: Normal
- **IM**: Instruction Memory
- **MP**: Data Memory
- **TEX**: Exception"
ARMv7-A MMU

Level 2 Table Entries

<table>
<thead>
<tr>
<th>Fault</th>
<th>IGNORE</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large page</td>
<td>Large page base address, PA[31:16]</td>
<td>X N</td>
<td>TE X</td>
</tr>
<tr>
<td>Small page</td>
<td>Small page base address, PA[31:12]</td>
<td>n G</td>
<td>S</td>
</tr>
</tbody>
</table>
ARMv7-A MMU

- More page attributes:
  - Global
  - Memory region attributes
    - Cacheable, Bufferable, TEX
  - Shareable
  - Domain
## ARMv7-A MMU

<table>
<thead>
<tr>
<th>TEX[2:0]</th>
<th>C</th>
<th>B</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>Strongly ordered</td>
</tr>
<tr>
<td>000</td>
<td>0</td>
<td>1</td>
<td>Shareable device</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>0</td>
<td>Outer and inner write-through, no write-allocate</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>1</td>
<td>Outer and inner write-back, no write-allocate</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>0</td>
<td>Outer and inner non-cacheable</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
<td>Outer and inner write-back, write-allocate</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>0</td>
<td>Non-shareable device</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>011</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>1BB</td>
<td>A</td>
<td>A</td>
<td>Cacheable memory; outer = AA, inner = BB</td>
</tr>
<tr>
<td>AA/BB</td>
<td>Attribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>Non-cacheable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Write-back, write-allocate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Write-through, no write-allocate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Write-back, no write-allocate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ARMv7-A MMU

• TEX remapping can be used to change TEX, C and B bits to an attribute index.

• Useful for operating systems to define a set of logical memory types using PRRR and NMRR registers.

  • TEX[0]:C:B = 0 → Device memory
  • TEX[0]:C:B = 1 → Normal memory
ARMv7-A MMU

- Large pages do not decrease table sizes.
- Sections and super-sections reduce the need for L2 table blocks and the penalty for walking the full table.
- Large pages, sections and super-sections increase the memory covered by the TLB.
ARMv7-A MMU

- Two root pointers, with configurable address coverage.
- TTBR0: Recommended for user applications (non-global)
- TTBR1: Recommended for system (global)
ARMv7-A TLBs

- The TLB caches the PTEs.
- PTE in TLB is if it is non global bound to an ASID which must be synced to the root PTP.
- No TLB flush necessary on context switch as ASID will change.
Q&A