Modern Virtual Memory Systems

Mengjia Yan
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Based on slides from Daniel Sanchez
Recap: Virtual Memory Systems

*Illusion of a large, private, uniform store*

Protection & Privacy
- several users, each with their private address space and one or more shared address spaces
  - page table ≡ name space

Demand Paging
- Provides the ability to run programs larger than the primary memory
- Hides differences in machine configurations

*The price is address translation on each memory reference*
Recap: Hierarchical Page Table

Virtual Address

Root of the Current Page Table

(Processor Register)

Level 1 Page Table

Level 2 Page Tables

Data Pages

page in primary memory

page in secondary memory

PTE of a nonexistent page
Recap: Translation Lookaside Buffers

Address translation is very expensive!
- In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*
- TLB hit \(\Rightarrow\) *Single-cycle Translation*
- TLB miss \(\Rightarrow\) *Page Table Walk to refill*

![Diagram of address translation process]

- Virtual address
  - VPN
  - Offset
- Physical address
  - PPN
  - Offset

\(\text{(VPN} = \text{virtual page number)}\)
\(\text{(PPN} = \text{physical page number)}\)
Recap: Translation Lookaside Buffers

Address translation is very expensive!
• In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*

TLB hit $\Rightarrow$ Single-cycle Translation
TLB miss $\Rightarrow$ Page Table Walk to refill

```
V R W D    tag        PPN
VPN offset
virtual address
hit?
physical address
PPN offset
```

(VPN = virtual page number)

(PPN = physical page number)
Recap: TLB Designs

- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages ➔ more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-16 way set-associative
- Random or FIFO replacement policy
- No process information in TLB?
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = \[ 64 \text{ entries} \times 4 \text{ KB} = 256 \text{ KB (if contiguous)} \]
Variable-Sized Page Support

Virtual Address

31  22  21  12  11  0

p1  p2  offset

10-bit  10-bit
L1 index  L2 index

Root of the Current Page Table

(Processor Register)

Level 1 Page Table

p1

Level 2 Page Tables

Offset

Data Pages

Page in primary memory
Large page in primary memory
Page in secondary memory
PTE of a nonexistent page
Variable-Sized Page Support

Virtual Address

```
31 22 21 12 11 0
```
```
p1  p2  offset
```

10-bit 10-bit
L1 index L2 index

Root of the Current Page Table

(Processor Register)

Level 1 Page Table

Level 2 Page Tables

Data Pages

- page in primary memory
- large page in primary memory
- page in secondary memory
- PTE of a nonexistent page
Variable-Size Page TLB

- Chicken-and-egg problem

```
virtual address

xxxxx...xxxxx 000000110 110010111000
```

```
<table>
<thead>
<tr>
<th>V</th>
<th>R</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tag

<table>
<thead>
<tr>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

L

```
Variable-Size Page TLB

- Chicken-and-egg problem
Variable-Size Page TLB

- Chicken-and-egg problem

### Example

<table>
<thead>
<tr>
<th>virtual address</th>
<th>2MB page VPN</th>
<th>2MB page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxxxxxxx......xxxxxxx</td>
<td>000000110</td>
<td>110010111000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4KB page VPN</th>
<th>4KB page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th>R</th>
<th>W</th>
<th>D</th>
<th>Tag</th>
<th>PPN</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Variable-Size Page TLB

- Chicken-and-egg problem

<table>
<thead>
<tr>
<th>V</th>
<th>R</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tag

<table>
<thead>
<tr>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

hit?

2MB page VPN

<table>
<thead>
<tr>
<th>VPN</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx</td>
<td>00000110</td>
</tr>
<tr>
<td>xxxx</td>
<td>110010111000</td>
</tr>
</tbody>
</table>

4KB page VPN

<table>
<thead>
<tr>
<th>VPN</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2MB page offset

<table>
<thead>
<tr>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000110</td>
</tr>
<tr>
<td>110010111000</td>
</tr>
</tbody>
</table>

4KB page offset

<table>
<thead>
<tr>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Variable-Size Page TLB

- Chicken-and-egg problem

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>Physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxxxxx...xxx</td>
<td>xxxxxxxxx110010111000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2MB page VPN</th>
<th>2MB page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxxxxx...xxx</td>
<td>000000110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4KB page VPN</th>
<th>4KB page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxxxxxxxx</td>
<td>110010111000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VRWD</th>
<th>Tag</th>
<th>PPN</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

hit?
Variable-Size Page TLB

- Chicken-and-egg problem
- Q: how about set-associative TLBs?

```
<table>
<thead>
<tr>
<th>VRWD</th>
<th>Tag</th>
<th>PPN</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>W</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>
```

```
virtual address

2MB page VPN | 2MB page offset
xxxxxxxxx......xxxxx | 000000110 | 110010111000

4KB page VPN | 4KB page offset
xxxxxxx | 110010111000

physical address

xxxxxxxxx......xxxxx | xxxxxxxxxx | 110010111000
```
Variable-Size Page TLB

- Chicken-and-egg problem
- Q: how about set-associative TLBs?
- Some systems use separate TLBs for different page sizes.

Virtual address

2MB page VPN

VRWD

Tag

PPN

L

hit?

Physical address

2MB page offset

4KB page VPN

4KB page offset

2MB page VPN

4KB page VPN

VRRWW

110010111000

000000110

xxxxxxx......xxxxxxx

110010111000

physical address

virtual address

VRRWW

110010111000

000000110

xxxxxxx......xxxxxxx

110010111000

2MB page VPN

4KB page VPN

VRRWW

110010111000

000000110

xxxxxxx......xxxxxxx

110010111000

2MB page offset

4KB page offset
Handling a TLB Miss

- Software (MIPS, Alpha)
- Hardware (SPARC v8, x86, PowerPC)
Handling a TLB Miss

• Software (MIPS, Alpha)
  • TLB miss causes an exception and the operating system walks the page tables and reloads TLB.
  • A privileged “untranslated” addressing mode used for walk

• Hardware (SPARC v8, x86, PowerPC)
Handling a TLB Miss

- **Software (MIPS, Alpha)**
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB.
  - *A privileged “untranslated” addressing mode used for walk*

- **Hardware (SPARC v8, x86, PowerPC)**
  - A memory management unit (MMU) walks the page tables and reloads the TLB.
  - If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction.
Handling a TLB Miss

- **Software (MIPS, Alpha)**
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB.
  - A privileged “untranslated” addressing mode used for walk

- **Hardware (SPARC v8, x86, PowerPC)**
  - A memory management unit (MMU) walks the page tables and reloads the TLB
  - If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

**Q: Pros and cons of software and hardware approaches?**
Hierarchical Page Table Walk: SPARC v8

Virtual Address

Context Table Register

Context Table

root ptr

L1 Table

PTE

L2 Table

PTE

L3 Table

PTE

Physical Address

PPN

Offset

MMU does this table walk in hardware on a TLB miss
Address Translation: putting it all together

Virtual Address

TLB Lookup

- hardware
- hardware or software
- software
Address Translation: putting it all together

Virtual Address

TLB Lookup

Protection Check

hit

hardware

hardware or software

software
Address Translation: putting it all together

Virtual Address

TLB Lookup

Protection Check

Physical Address (to cache)

hit

permitted

hardware

hardware or software

software

February 20, 2020
Address Translation: 
*putting it all together*

Virtual Address

- TLB Lookup
  - hit
  - hardware
  - hardware or software
  - software

Protection Check

- denied
- permitted

Protection Fault

Physical Address (to cache)

SEGFAULT
Address Translation: putting it all together

Virtual Address

- TLB Lookup
  - hit
  - miss

Page Table Walk

Protection Check
  - denied
  - permitted

Protection Fault
  - Physical Address (to cache)
  - SEGFAULT
Address Translation: putting it all together

- **Virtual Address**
  - TLB Lookup
    - miss
    - hit
    - **Page Table Walk**
      - the page is \( \in \text{memory} \)
      - Update TLB
    - **Protection Check**
      - permitted
      - denied
      - **Protection Fault**
        - SEGFAULT
        - Physical Address (to cache)
Address Translation: putting it all together

Virtual Address

TLB Lookup

Virtual Address

TLB Lookup

Page Table Walk

Page Table Walk

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault (OS loads page)

Update TLB

Page Table Walk

Protection Fault
Address Translation: putting it all together

Virtual Address

TLB Lookup

Virtual Address → TLB Lookup

TLB Lookup:
- hit
- miss

Page Table Walk:
- the page is in memory
- the page is not in memory

Page Fault (OS loads page)

Update TLB

Protection Check:
- denied
- permitted

Protection Fault

Physical Address (to cache)

SEGFAULT

Where?

February 20, 2020

MIT 6.823 Spring 2020
Topics

• Interrupts
• Speeding up the common case:
  – TLB & Cache organization
• Speeding up page table walks
• Modern Usage

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Page Fault
(OS loads page)

Update TLB

Protection Fault

Where?

SEGFAULT
**Interrupts:** altering the normal flow of control

An *external or internal event* that needs to be processed by another (system) program. The event is usually unexpected or rare from program’s point of view.
Interrupts: altering the normal flow of control

An *external or internal event* that needs to be processed by another (system) program. The event is usually unexpected or rare from program’s point of view.
Causes of Interrupts

Interrupt: an event that requests the attention of the processor

- Asynchronous: an external event
  - input/output device service-request
  - timer expiration
  - power disruptions, hardware failure

- Synchronous: an internal event (a.k.a exception)
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception
  - misaligned memory access
  - virtual memory exceptions:
    - page faults, TLB misses, protection violations
  - traps: system calls, e.g., jumps into kernel
Asynchronous Interrupts: invoking the interrupt handler

- An I/O device requests attention by asserting one of the *prioritized interrupt request lines*
- Tricky: interrupted thread cannot anticipate/prepare for this control transfer
- When the processor decides to process the interrupt
Asynchronous Interrupts: invoking the interrupt handler

- An I/O device requests attention by asserting one of the *prioritized interrupt request lines*
- Tricky: interrupted thread cannot anticipate/prepare for this control transfer
- When the processor decides to process the interrupt
  - **Precise interrupt**: It stops the current program at instruction $I_i$, completing all the instructions up to $I_{i-1}$
  - It saves the PC of instruction $I_i$ in a special register (EPC)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode
Interrupt Handler

• Needs to read a status register that indicates the cause of the interrupt

• Uses a special indirect jump instruction RFE (return-from-exception) that
  – enables interrupts
  – restores the processor to the user mode
  – restores hardware status and control state

• Nested interrupts
Interrupt Handler

• Needs to read a *status register* that indicates the cause of the interrupt

• Uses a special indirect jump instruction RFE (*return-from-exception*) that
  - enables interrupts
  - restores the processor to the user mode
  - restores hardware status and control state

• Nested interrupts
  - Saves EPC before enabling interrupts
  - need an instruction to move EPC into GPRs
  - need a way to mask further interrupts at least until EPC can be saved
Synchronous Interrupts

- A synchronous interrupt (exception) is caused by a particular instruction

- In general, the instruction cannot be completed and needs to be restarted after the exception has been handled
  - With pipelining, requires undoing the effect of one or more partially executed instructions

- In case of a trap (system call), the instruction is considered to have been completed
  - A special jump instruction involving a change to privileged kernel mode
Topics

• Interrupts

• Speeding up the common case:
  – TLB & Cache organization

• Speeding up page table walks

• Modern Usage
Address Translation in CPU
Address Translation in CPU
Address Translation in CPU

TLB miss? Page Fault? Protection violation?

TLB miss? Page Fault? Protection violation?
Address Translation in CPU

- TLB miss: a hardware or software mechanism
- Page fault or protection violation: software handler
TLB miss? Page Fault? Protection violation?

- TLB miss: a *hardware* or *software* mechanism
- Page fault or protection violation: software handler
- The common case: TLB lookup before every cache access
Address Translation in CPU

• TLB miss: a hardware or software mechanism
• Page fault or protection violation: software handler
• The common case: TLB lookup before every cache access

• Need mechanisms to cope with the additional latency of TLB:
Address Translation in CPU

- TLB miss: a *hardware* or *software* mechanism
- Page fault or protection violation: software handler
- The common case: TLB lookup before every cache access

- Need mechanisms to cope with the additional latency of TLB:
  - slow down the clock
  - pipeline the TLB and cache access
  - virtual-address caches
  - parallel TLB/cache access
Virtual-Address Caches

Pros and cons:
Virtual-Address Caches

Pros and cons:

Alternative: place the cache before the TLB
Virtual-Address Caches

Pros and cons:
- one-step process in case of a hit (+)

Alternative: place the cache before the TLB
Virtual-Address Caches

Pros and cons:

- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)

Alternative: place the cache before the TLB
Virtual-Address Caches

Pros and cons:
- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- aliasing problems due to the sharing of pages (-)

Alternative: place the cache before the TLB
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page

General Solution: *Disallow aliases to coexist in cache*
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page

General Solution: *Disallow aliases to coexist in cache*
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Disallow aliases to coexist in cache*
Aliasing in Virtual-Address Caches

Two virtual pages share one physical page

General Solution: **Disallow aliases to coexist in cache**

Software (i.e., OS) solution for direct-mapped cache

VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)
Concurrent Access to TLB & Cache

VA

VPN

TLB

Page Offset

k

PA

PPN

Direct-map Cache

$2^L$ blocks

$2^b$-byte block

Data
Concurrent Access to TLB & Cache

Virtual Index

Direct-map Cache
$2^L$ blocks
$2^b$-byte block

Data
Concurrent Access to TLB & Cache

Index L is available without consulting the TLB
⇒ cache and TLB accesses can begin simultaneously
Concurrent Access to TLB & Cache

Index L is available without consulting the TLB
⇒ cache and TLB accesses can begin simultaneously
Concurrent Access to TLB & Cache

Index L is available without consulting the TLB
⇒ *cache and TLB accesses can begin simultaneously*
Tag comparison is made after both accesses are completed
Concurrent Access to TLB & Cache

Index L is available without consulting the TLB
⇒ *cache and TLB accesses can begin simultaneously*

Tag comparison is made after both accesses are completed
Index L is available without consulting the TLB
⇒ cache and TLB accesses can begin simultaneously
Tag comparison is made after both accesses are completed

When does this work? \( L + b < k \) \( L + b = k \) \( L + b > k \)
Index L is available without consulting the TLB

⇒ cache and TLB accesses can begin simultaneously

Tag comparison is made after both accesses are completed

When does this work? \( L + b < k \) ✓  \( L + b = k \) __  \( L + b > k \) __
Concurrent Access to TLB & Cache

Index L is available without consulting the TLB
⇒ cache and TLB accesses can begin simultaneously
Tag comparison is made after both accesses are completed

When does this work? \( L + b < k \) ✓  \( L + b = k \) ✓  \( L + b > k \)
Concurrent Access to TLB & Cache

Index L is available without consulting the TLB
⇒ cache and TLB accesses can begin simultaneously
Tag comparison is made after both accesses are completed

When does this work? L + b < k ✅ L + b = k ✅ L + b > k ✗
Concurrent Access to TLB & Large L1
The problem with L1 > Page size
Concurrent Access to TLB & Large L1
The problem with L1 > Page size

TLB

Virtual Index

L1 cache

VIPT

Direct-map

VA

VPN

b

TLB

Page Offset

k

PA

PPN

b

Tag

hit?
Concurrent Access to TLB & Large L1
The problem with L1 > Page size
Can $VA_1$ and $VA_2$ both map to $PA$?
Concurrent Access to TLB & Large L1
The problem with L1 > Page size

Can VA₁ and VA₂ both map to PA? Yes
Virtual-Index Physical-Tag Caches: Associative Organization

After the PPN is known, $2^a$ physical tags are compared

Is this scheme realistic?
A solution via Second-Level Cache

A solution via Second-Level Cache

Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is “inclusive” of both Instruction and Data caches
Anti-Aliasing Using L2: *MIPS R10000*

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)
Anti-Aliasing Using L2: *MIPS R10000*

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)

```plaintext

<table>
<thead>
<tr>
<th>VA</th>
<th>VPN</th>
<th>Page Offset b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>into L2 tag</td>
</tr>
</tbody>
</table>

Virtual Index

L1 cache

**V IPT**

Direct-map

<table>
<thead>
<tr>
<th>VA1</th>
<th>PPNa</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L2 cache

(PIPT)

<table>
<thead>
<tr>
<th>PA</th>
<th>a1</th>
<th>Data</th>
</tr>
</thead>
</table>
```
Anti-Aliasing Using L2: *MIPS R10000*

Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2).

After VA2 is resolved to PA, collision is detected in L2. Collision →
Anti-Aliasing Using L2: \textit{MIPS R10000}

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 \neq VA2)
- After VA2 is resolved to PA, collision is detected in L2. Collision \( \Rightarrow \) Field \( a \) is different.
Anti-Aliasing Using L2: *MIPS R10000*

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 \(\neq\) VA2)
- After VA2 is resolved to PA, collision is detected in L2. Collision → Field a is different.
- VA1 will be purged from L1, and VA2 will be loaded \(\Rightarrow\) *no aliasing!*

**Diagram:**
- VA and PA are mapped through VPN and Page Offset to TLB, which resolves to L1 cache.
- L1 cache maps to L2 cache via VIPT.
- When VA1 and VA2 map to the same PA, and VA1 is in L1, collision is detected.
- VA1 is purged from L1, and VA2 is loaded.

**Note:** Field a is different for L2 cache (PIPT) mapping.
Anti-Aliasing Using L2: *MIPS R10000*

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)
- After VA2 is resolved to PA, collision is detected in L2. Collision \(\rightarrow\) Field a is different.
- VA1 will be purged from L1, and VA2 will be loaded \(\Rightarrow\) no aliasing!
Anti-Aliasing Using L2: *MIPS R10000*

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 \( \neq \) VA2)
- After VA2 is resolved to PA, collision is detected in L2. Collision \( \Rightarrow \) **Field a is different.**
- VA1 will be purged from L1, and VA2 will be loaded \( \Rightarrow \) *no aliasing!*

### Diagram Description

- VA (Virtual Address) flows through the TLB (Translation Lookaside Buffer) to match the Page Offset into L2 tag.
- L1 cache with VIPT (Virtual to Physical Translation) and Direct-map structure.
- L2 cache (PIPT) with pages mapping to different physical Page Numbers (PN).
Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2).

After VA2 is resolved to PA, collision is detected in L2. Collision $\Rightarrow$ Field a is different.

VA1 will be purged from L1, and VA2 will be loaded $\Rightarrow$ no aliasing!
Physically addressed L2 can also be used to avoid aliases in virtually addressed L1
Topics

• Interrupts

• Speeding up the common case:
  – TLB & Cache organization

• Speeding up page table walks

• Modern Usage
Page Fault Handler

• When the referenced page is not in DRAM:
  – The missing page is located (or created)
  – It is brought in from disk, and page table is updated
    *Another job may be run on the CPU while the first job waits for the requested page to be read from disk*
  – If no free pages are left, a page is swapped out
    *Pseudo-LRU replacement policy*
Page Fault Handler

• When the referenced page is not in DRAM:
  – The missing page is located (or created)
  – It is brought in from disk, and page table is updated
    *Another job may be run on the CPU while the first job waits for the requested page to be read from disk*
  – If no free pages are left, a page is swapped out
    *Pseudo-LRU replacement policy*

• Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS
  – Untranslated addressing mode is essential to allow kernel to access page tables
Translation for Page Tables

- Can references to page tables cause TLB misses?
- Can this go on forever?
Translation for Page Tables

- Can references to page tables cause TLB misses?
- Can this go on forever?

A program that traverses the page table needs a “no translation” addressing mode.
A PTE in primary memory contains primary or secondary memory addresses

A PTE in secondary memory contains only secondary memory addresses

⇒ a page of a PT can be swapped out only if none of its PTE’s point to pages in the primary memory

Why?________________________________________
Swapping a Page of a Page Table

A PTE in primary memory contains primary or secondary memory addresses

A PTE in secondary memory contains only secondary memory addresses

⇒ a page of a PT can be swapped out only if none of its PTE’s point to pages in the primary memory

Why?

Pointed-to pages become inaccessible (page fault due to swapped-out PT page)

May cause deadlock!
Atlas Revisited

- One PAR for each physical page

- PAR’s contain the VPN’s of the pages *resident in primary memory*

- Advantage:

- Disadvantage?
Atlas Revisited

- One PAR for each physical page

- PAR’s contain the VPN’s of the pages *resident in primary memory*

- **Advantage:**
  - The size is proportional to the size of the primary memory

- **Disadvantage?**
Atlas Revisited

- One PAR for each physical page
- PAR’s contain the VPN’s of the pages *resident in primary memory*
  - Advantage:
    - The size is proportional to the size of the primary memory
- Disadvantage?
  
  *Must check all PARs!*
Hashed Page Table: Approximating Associative Addressing

Virtual Address

VPN

Offset

PID

hash

Base of Table

PA of PTE

Page Table

Primary Memory

VPN PID PPN

VPN PID DPN

VPN PID

VPN PID
Hashed Page Table: Approximating Associative Addressing

- Hashed Page Table is typically 2 to 3 times larger than the number of PPNs to reduce collision probability
Hashed Page Table: Approximating Associative Addressing

- Hashed Page Table is typically 2 to 3 times larger than the number of PPNs to reduce collision probability.
- It can also contain DPNs for some non-resident pages (not common).
- If a translation cannot be resolved in this table then the software consults a data structure that has an entry for every existing page.
Virtual Memory Use Today - 1

- Desktop/server/cellphone processors have full demand-paged virtual memory
  - Portability between machines with different memory sizes
  - Protection between multiple users or multiple tasks
  - Share small physical memory among active tasks
  - Simplifies implementation of some OS features

- Vector supercomputers and GPUs have translation and protection but not demand paging
  (Older Crays: base&bound, Japanese & Cray X1: pages)
  - Don’t waste expensive processor time thrashing to disk (make jobs fit in memory)
  - Mostly run in batch mode (run set of jobs that fits in memory)
  - Difficult to implement restartable vector instructions
Virtual Memory Use Today - 2

- Most embedded processors and DSPs provide physical addressing only
  - Can’t afford area/speed/power budget for virtual memory support
  - Often there is no secondary storage to swap to!
  - Programs custom-written for particular memory configuration in product
  - Difficult to implement restartable instructions for exposed architectures
Next lecture: Pipelining!
Global System Address Space

- Level A maps users’ address spaces into the global space providing privacy, protection, sharing etc.
- Level B provides demand paging for the large global system address space
- Level A and Level B translations may be kept in separate TLB’s
Hashed Page Table Walk:
PowerPC Two-level, Segmented Addressing

64-bit user VA

<table>
<thead>
<tr>
<th>Seg ID</th>
<th>Page</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35</td>
<td>51</td>
</tr>
</tbody>
</table>

Hashed Segment Table

<table>
<thead>
<tr>
<th>Global Seg ID</th>
<th>Page</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>51</td>
<td>67</td>
</tr>
</tbody>
</table>

Hashed Page Table

<table>
<thead>
<tr>
<th>Hashed Segment Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash&lt;sub&gt;S&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hashed Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash&lt;sub&gt;P&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

PA of Segment Table

per process

PA of Page Table

system-wide

[ IBM numbers bits with MSB=0 ]

40-bit PA

<table>
<thead>
<tr>
<th>PPN</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>27</td>
</tr>
</tbody>
</table>

80-bit System VA

Global Seg ID

<table>
<thead>
<tr>
<th>0</th>
<th>51</th>
<th>67</th>
<th>79</th>
</tr>
</thead>
</table>

[ IBM numbers bits with MSB=0 ]
Power PC: Hashed Page Table

- Each hash table slot has 8 PTEs \(<\text{VPN}, \text{PPN}>\) that are searched sequentially.
- If the first hash slot fails, an alternate hash function is used to look in another slot.
  
  *All these steps are done in hardware!*

- Hashed Table is typically 2 to 3 times larger than the number of physical pages.
- The full backup Page Table is a software data structure.