19. Translation Lookaside Buffers

Operating System: Three Easy Pieces
Part of the chip’s memory-management unit (MMU).

A hardware cache of **popular** virtual-to-physical address translation.
TLB Basic Algorithms

1: VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2: (Success, TlbEntry) = TLB_Lookup(VPN)
3: if(Success == True){ // TLB Hit
4:   if(CanAccess(TlbEntry.ProtectBit) == True){
5:     offset = VirtualAddress & OFFSET_MASK
6:     PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:     AccessMemory(PhysAddr)
8:   }else RaiseException(PROTECTION_ERROR)

- (1 lines) extract the virtual page number (VPN).
- (2 lines) check if the TLB holds the translation for this VPN.
- (5-8 lines) extract the page frame number from the relevant TLB entry, and form the desired physical address and access memory.
11: }else{ //TLB Miss
12:     PTEAddr = PTBR + (VPN * sizeof(PTE))
13:     PTE = AccessMemory(PTEAddr)
14:     if (PTE.Valid == False) {
15:         RaiseException(SEGMENTATION_FAULT)
16:     }else if (CanAccess(PTE.ProtectBits) == False) {
17:         RaiseException(PROTECTION_FAULT)
18:     }else{
19:         TLB_Insert( VPN , PTE.PFN , PTE.ProtectBits)
20:         RetryInstruction()}
21: }
22:}

- (11-13 lines) Accesses the page table to find the translation.
- (13-17 lines) Check if PTE is ok
- (19 lines) updates the TLB with the translation.
Example: Accessing An Array

- How a TLB can improve its performance.

```
0:    int sum = 0 ;
1:    for( i=0; i<10; i++){
2:        sum+=a[i];
3:    }
```

The TLB improves performance due to spatial locality

3 misses and 7 hits.
Thus TLB hit rate is 70%.
Locality

- **Temporal Locality**
  - An instruction or data item that has been recently accessed will likely be re-accessed soon in the future.

- **Spatial Locality**
  - If a program accesses memory at address \( x \), it will likely soon access memory near \( x \).

```
Virtual Memory

Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
...
Page n
```

```
1\textsuperscript{st} "sum" access is page1.
2\textsuperscript{nd} "sum" access is also page1.
```

```
Virtual Memory

Page 1
Page 2
Page 3
Page 4
Page 5
...
Page n
```

```
1\textsuperscript{st} "array" access is page1.
N\textsuperscript{nd} "array" access is near by page1.
```
Who Handles The TLB Miss?

- Hardware handle the TLB miss entirely on **CISC**.
  - The hardware has to know exactly where the page tables are located in memory.
  - The hardware would “walk” the page table (TLB walker), find the correct page-table entry and **extract** the desired translation, **update** and **retry** instruction. This is the previous algorithm.
  - **hardware-managed TLB.**
RISC have what is known as a software-managed TLB.

- On a TLB miss, the hardware raises exception (trap handler).
  - Trap handler is code within the OS that is written with the express purpose of handling TLB miss.
TLB Control Flow algorithm (OS Handled)

1: VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2: (Success, TlbEntry) = TLB_Lookup(VPN)
3: if (Success == True){ // TLB Hit
4:     if (CanAccess(TlbEntry.ProtectBits) == True){
5:         Offset = VirtualAddress & OFFSET_MASK
6:         PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:         Register = AccessMemory(PhysAddr)
8:     } else{
9:         RaiseException(PROTECTION_FAULT)}
10: } else { // TLB Miss
11:     RaiseException(TLB_MISS)
TLB entry

- TLB is managed by **Full Associative** method.
  - A typical TLB might have 32 to 128 entries.
  - Hardware search the entire TLB in parallel to find the desired translation (i.e. a cache without index bit, i.e. tag=VPN)
  - other bits: valid bits, protection bits, address-space identifier, dirty bit

![Typical TLB entry look like this](image)

Translation Lookaside Buffers
TLB Issue: Context Switching

Process A

access VPN10

Virtual Memory

Page 0
Page 1
Page 2
...
Page n

Virtual Memory

Page 0
Page 1
Page 2
...
Page n

Insert TLB Entry

TLB Table

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Translation Lookaside Buffers
TLB Issue: Context Switching

Process A

Process B

Context Switching

Virtual Memory

TLB Table

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>170</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Insert TLB Entry

AOS@UC

Translation Lookaside Buffers
TLB Issue: Context Switching

Process A

Page 0
Page 1
Page 2
...
Page n

Virtual Memory

Process B

Page 0
Page 1
Page 2
...
Page n

Virtual Memory

TLB Table

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>170</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Can’t Distinguish which entry is meant for which process
To Solve Problem

- Provide an address space identifier (ASID) field in the TLB.

![Diagram of TLB Table]

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>1</td>
<td>rwx</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>170</td>
<td>1</td>
<td>rwx</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Two processes share a page.

- Process 1 is sharing physical page 101 with Process 2.
- P1 maps this page into the 10th page of its address space.
- P2 maps this page to the 50th page of its address space.

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>rwx</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>101</td>
<td>1</td>
<td>rwx</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sharing of pages is useful as it reduces the number of physical pages in use.
TLB Replacement Policy

- LRU (Least Recently Used)
  - Evict an entry that has not recently been used.
  - Take advantage of *locality* in the memory-reference stream.

Reference Row:

```
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1
```

Page Frame:

```
7 7 7 2
0 0 0
3 1 1
```

```
2 4 4 4 0
0 0 3 3
3 2 2 2
```

```
1 1
3 0
2 2
```

Total 11 TLB miss
A Real TLB Entry

All 64 bits of this TLB entry (example of MIPS R4000)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-bit VPN</td>
<td>The rest reserved for the kernel.</td>
</tr>
<tr>
<td>24-bit PFN</td>
<td>Systems can support with up to 64GB of main memory (2^{24} \times 4KB pages).</td>
</tr>
<tr>
<td>Global bit(G)</td>
<td>Used for pages that are globally-shared among processes (ignore ASID).</td>
</tr>
<tr>
<td>ASID</td>
<td>OS can use to distinguish between address spaces.</td>
</tr>
<tr>
<td>Coherence bit(C)</td>
<td>determine how a page is cached by the hardware.</td>
</tr>
<tr>
<td>Dirty bit(D)</td>
<td>marking when the page has been written.</td>
</tr>
<tr>
<td>Valid bit(V)</td>
<td>tells the hardware if there is a valid translation present in the entry.</td>
</tr>
</tbody>
</table>

ISA provides the mechanisms to control the TLB content:
TLBP (probe), TLBR (read), TLBWI (replace index), TLBWRI (replace random)
Applications with very large memory footprint might be heavily impacted by TLB size (i.e. the locality of the code/data is too large)

- Data/Instruction Separate TLBs
- Multiple level TLB
- Mega pages
- No TLB ➔ no pages for a fraction of the address space

TLB is an itchy issue for virtualization

- Complex hardware support required to minimize performance overheads
- Unikernels
Recommended Reading (beyond class)

- Unikernels
  - [http://unikernel.org](http://unikernel.org)

- Specialized single-address-space OS
  - One application
  - Convert the OS back in a set of library routines
  - Designed to be run in virtual environments

- Improved security and performance over conventional OS
  - Smaller attack surface
  - No virtual memory overhead
  - No VMM redirections in syscalls
This lecture slide set is used in AOS course at University of Cantabria. Was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea Arpaci-Dusseau (at University of Wisconsin)