Virtual Memory (1)
Page Tables

When and why do we access a page table?

- On every instruction to translate virtual to physical addresses?
Page Tables

When and why do we access a page table?
- On every instruction to translate virtual to physical addresses?

In Blitz, YES, but in real machines NO!

In real machines it is only accessed
- On TLB miss faults to refill the TLB
- During process creation and destruction
- When a process allocates or frees memory?
Translation Lookaside Buffer

Problem: MMU can’t keep up with the CPU if it goes to the page table on every memory access!
Translation Lookaside Buffer

Solution:
- Cache the page table entries in a hardware cache
- Small number of entries (e.g., 64)
- Each entry contains page number and other stuff from page table entry
- Associatively indexed on page number ie. You can do a lookup in a single cycle
Translation Lookaside Buffer
## Hardware Operation of TLB

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>
# Hardware Operation of TLB

## Virtual Address

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused D R W V</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused D R W V</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused D R W V</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused D R W V</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused D R W V</td>
</tr>
</tbody>
</table>

## Physical Address

<table>
<thead>
<tr>
<th>Frame Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1312</td>
<td>0</td>
</tr>
</tbody>
</table>
# Hardware Operation of TLB

## Virtual Address Table

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>

The table shows the mapping of virtual addresses to physical addresses through the TLB. The key is highlighted by the blue line, indicating the process of address translation.
Hardware Operation of TLB

**Virtual Address**

- Page number
- Offset

**Physical Address**

- Frame number
- Offset

**Key**

- Page Number
- Frame Number
- Other

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>
Hardware Operation of TLB

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>

Physical address: 01312

Virtual address: 0123

Frame number, offset

Key

Page number, offset

Other: D R W V
Hardware Operation of TLB

Virtual address:
- Key
- Page Number
- Frame Number
- Other

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>

Physical address:
- Frame number
- Offset

Unused: D R W V
Software Operation of TLB

What if the entry is not in the TLB?
- Go look in the page table in memory
- Find the right entry
- Move it into the TLB
- But which TLB entry should be replaced?
Software Operation of TLB

Hardware TLB refill
- Page tables in specific location and format
- TLB hardware handles its own misses
- Replacement policy fixed by hardware

Software refill
- Hardware generates trap (TLB miss fault)
- Lets the OS deal with the problem
- Page tables become entirely a OS data structure!
- Replacement policy managed in software
Software Operation of TLB

How can we prevent the next process from using the last process’s address mappings?

- **Option 1:** empty the TLB on context switch
  
  *New process will generate faults until its pulls enough of its own entries into the TLB*

- **Option 2:** just clear the “Valid Bit” on context switch
  
  *New process will generate faults until its pulls enough of its own entries into the TLB*

- **Option 3:** the hardware maintains a process id tag on each TLB entry
  
  *Hardware compares this to a process id held in a specific register ... on every translation*
Page Tables

Do we access a page table when a process allocates or frees memory?
Page Table Usage

Do we access a page table when a process allocates or frees memory?
- Not necessarily

Library routines (malloc) can service small requests from a pool of free memory already allocated within a process address space

When these routines run out of space a new page must be allocated and its entry inserted into the page table
- This allocation is requested using a system call
Page Table Design

Page table size depends on
- Page size
- Virtual address length

Memory used for page tables is overhead!
- How can we save space? ... and still find entries quickly?

Three options
- Single-level page tables
- Multi-level page tables
- Inverted page tables
Single-Level Page Tables

20-bits  12-bits
page number  offset

Single-level page table
Single-Level Page Tables

- 20-bits page number
- 12-bits offset

Single-level page table

Frames in memory
Single-Level Page Tables

Problem: requires one page table entry per virtual page!
Single-Level Page Tables

- 20-bits: page number
- 12-bits: offset

32 bit addresses and 4KB pages means $2^{20}$ page table entries per process.
Single-Level Page Tables

64 bit addresses and 4KB pages means $2^{52}$ page table entries per process!
Multi-Level Page Tables

Top-level Page table

2nd-level tables

frames in memory
Multi-Level Page Tables

<table>
<thead>
<tr>
<th></th>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Top-level Page table

2nd-level tables

Frames in memory
Multi-Level Page Tables

- Top-level Page table
- 2nd-level tables
- Frames in memory

<table>
<thead>
<tr>
<th>PT1</th>
<th>PT2</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-bits</td>
<td>10-bits</td>
<td>12-bits</td>
</tr>
</tbody>
</table>
Multi-Level Page Tables

<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>

Top-level Page table

2nd-level tables

frames in memory
Multi-Level Page Tables

<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>

Top-level Page table

2nd-level tables

frames in memory
Multi-Level Page Tables

- 10-bits
- 10-bits
- 12-bits

- PT1
- PT2
- offset

Top-level Page table

2nd-level tables

frames in memory
Multi-Level Page Tables

<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>

Top-level Page table

2nd-level tables

Frames in memory
Multi-Level Page Tables

Ok, but how exactly does this save space?
Multi-Level Page Tables

Ok, but how exactly does this save space?
Not all pages within a virtual address space are allocated
- Not only do they not have a page frame, but that range of virtual addresses is not being used
- So no need to maintain complete information about it
- Some intermediate page tables are empty and not needed

We could also page the page table
- This saves space but slows access ... a lot!
Address Translation Puzzle