Translation Lookaside Buffer (TLB) to cache the page table
Caching the page table in a TLB

TLB caches a few entries of the page table of process 1.

With TLB, we avoid accessing memory twice on each memory reference?

Caching the page table into the TLB

Example:
- 32-pages virtual mem
- 32-entry Page Table
- 4-entries TLB,
- Fully associative
- 1 entry caching granularity

If page table entry is not in TLB:
- TLB miss
- get the entry from the page table (PT walk)
- load it to the TLB.
- may have to replace a valid TLB entry.
Caching the page table into the TLB

Example:
- 4-entries TLB,
- direct mapped
- 1 entry caching granularity

TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Page table

<table>
<thead>
<tr>
<th>Valid</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Physical memory

Disk storage

(swap space)

Example:
- 4-entries TLB,
- direct mapped
- 1 entry caching granularity

Virtual space = 128 pages
Page = 16 words

| Physical memory = 8 pages
| Page = 16 words |
|-----------------|----------------|
| Page 3          | Page 24        |
| Page 67         | Page 88        |
| Page 67         | Page 24        |
| Page 3          | Page 24        |

Ex: Consider references to locations:
00110000 0101 (page 24) ➔ 110 0101
10000111 0111 (page 67) ➔ 101 0111
10110000 0001 (page 88) ➔ TLB miss
00110011 0011 (page 25) ➔ TLB miss + Page fault

Page #   Page offset
Virtual word address

Page #   Page offset
Physical word address

translate

x x x x x x z z z z

y y y z z z z
The Page Table (PT) is very large

Example: If VS = 32-bit address (4GBytes) and memory page size = 4KB
  → VS = 1 million pages, page table = 1 Million entry.
  → if each table entry = 4 bytes  → page table occupies 4MB
  → page table occupies 1024 memory pages (memory page size =4KB)

The PT is too large to be stored in physical memory

The Page Table (PT) is very large

- The PT is stored in 1024 pages of the virtual memory space.
- PT’s currently used pages are brought to memory, and like any other page in the virtual space, the location of a page in memory is recorded in PT
- Note that the 1024 PT entries corresponding to the pages of the PT can fit in the first page of the PT – That page is pinned in memory.

- On a TLB miss, a “PT walker” is invoked to bring the missing PT entry to the TLB.
- The PT address register is replaced by a PT base address register which points to the first page of PT (pinned in memory)
Multi level Page Tables (multi level PT)

- In the example of 4GB VS and 4KB pages, the PT can be stored in 1024 pages
  - Pages of the PT are brought to memory on demand
  - The first page (root) of the PT keeps track of the locations of PT pages in memory.
- This is a “2-level” PT organization – may generalize to a multi-level PT organization
- Memory foot-print = the part of the VS which is actually used (accessed)
  - A large number of pages in the VS are not allocated or used (empty).
  - Hence a large number of entries of the PT are never accessed

Alpha 21264 example (3-levels page tables)
The whole picture

- If cache miss → process stalls (pipeline stalls)
- If TLB miss → process stalls (pipeline stalls)
- If Page fault → process relinquishes the CPU

CPU

Virtual address from lw/sw instructions or from program counter (PC)

Virtual page number

Page offset

Data

Block of a page

Physical address

Physical Memory

Part of the page table

Virtual Address space

The OS is invoked to move a page from disk (where virtual pages reside) to physical memory

TLBs and caches

- Page table walk to get the PT entry into the TLB (hardware)
- If PT indicates that page is not in memory, then service page fault (software – OS)

Note that there cannot be a page fault in case of a TLB hit – there is no reason for the PT entry of a page to be in the TLB if the page is not in memory

Some pages in the VS cannot be accessed if executing in user mode. “Access bits” in the PT entry for these pages are used to impose the appropriate protection.
### 2-Level TLB Organization for Cortex-A8 and Core-i7

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>ARM Cortex-A8</th>
<th>Intel Core i7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual address</td>
<td>32 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>Physical address</td>
<td>32 bits</td>
<td>44 bits</td>
</tr>
<tr>
<td>Page size</td>
<td>Variable: 4, 16, 64 KB, 1, 16 MB</td>
<td>Variable: 4 KB, 2/4 MB</td>
</tr>
<tr>
<td>TLB organization</td>
<td>1 TLB for instructions and 1 TLB for data</td>
<td>1 TLB for instructions and 1 TLB for data per core</td>
</tr>
<tr>
<td></td>
<td>Both TLBs are fully associative, with 32 entries, round robin replacement</td>
<td>Both L1 TLBs are four-way set associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td>TLB misses handled in hardware</td>
<td>L1 I-TLB has 128 entries for small pages, 7 per thread for large pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L1 D-TLB has 64 entries for small pages, 32 for large pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 TLB is four-way set associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 TLB has 512 entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLB misses handled in hardware</td>
</tr>
</tbody>
</table>