The Page Table is allocated in the virtual memory

- Each process has its own page table stored in memory starting at a specific address indicated in the page address register.
- The page table and page address register are part of the process context (along with the PC, stack pointer, registers …)
- A memory reference (if hits in main memory) requires two memory operations???
- A page fault (main memory miss) requires a disk operation.

Multiple processes share the physical memory
Multiple processes share the physical memory

Caching the page table in a Translation Lookaside Buffer
Caching the page table in a TLB

Making Address Translation Fast

With the TLB, we avoid accessing memory twice for each memory reference.

What if we get a TLB miss (page table entry is not in TLB)?
  • Get the entry from the page table (from memory) and load it to the TLB.
  • may have to replace a TLB entry (the LRU)
Example

Page table (128 entries)

Virtual word address

Physical word address

Physical Page #

valid

Page #

Page offset

Example: Consider references to locations:

- 0011000 0101 (page 24)
- 1000011 0111 (page 67)
- 1011000 0001 (page 88)
- 0011001 0011 (page 25)

Physical memory = 8 pages
Page = 16 words

Ex: Consider references to locations:

- 0011000 0101 (page 24)
- 1000011 0111 (page 67)
- 1011000 0001 (page 88)
- 0011001 0011 (page 25)

Note: The page table is stored in pages within the virtual space. Hence the page table contains entries for its own pages.

Page Tables can be very large

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

The physical memory may contain only a few pages of the page table.
Moreover, a fraction of the page table entries that are in memory are cached in TLB.
Multi level Page Tables (multi level PT)

Example: If VS = 32-bit address (byte address) and page size = 4KB
→ VS = 1 million pages and page table = 1 Million entry.
→ if each table entry = 4 bytes → each page can hold 1024 entries of the PT
→ page table occupies 4MB = 1024 pages

A level 1 PT occupies one page and contains one entry for each of the 1024 pages of a level 2 PT

Notes:
• A large number of pages in the VS are not used (empty).
• Hence a large number of entries of the PT are never accessed
• Memory foot-print = the part of the VS which is actually used (accessed)
• The level 1 PT is always kept in physical memory and is pointed to by a “base register”
• Pages of the level 2 PT are brought to physical memory on demand.

Alpha 21264 example (3-levels page tables)
TLBs and caches

Virtual address

TLB

Physical address

cache

In this example, the TLB is fully associative.

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, 512 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>
The whole picture

CPU (pipeline) stalls if:
- TLB miss (but no page fault)
- Cache miss

CPU

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

Virtual page
number

Page offset

CPU

Physical
address

Block of
a page

Physical Memory

Part of the
Page table

Page table walker

Bring page table
entry to the TLB

Page fault handler

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

Page fault

Page table

Physical address

TLBs and caches

Virtual address

TLB access

Yes

TLB hit

No

TLB miss exception

Yes

No

Virtual page number

Page offset

Physical address

Cache

To memory

Physical address

TLB

No

Try to read data from cache

Write?

Yes

Write hit

No

Write access bit on?

No

Write protection exception

Yes

Try to write data to cache

Cache miss

No

Cache hit

Yes

Cache miss stall
while read block

Yes

Cache hit

No

Cache miss stall
while read blocks

No

Yes

Write data into cache, update the dirty bit(s). Get
the data and the address into the valid buffer.

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 is the page
is not in memory

Assumes a
write through
cache