Two-Level Page Tables

Each process needs its own address space!

Choosing page size

- A large page size
  - keeps page table small.
  - reduces cache miss times, if accesses have locality
  - reduces start-up overhead in moving data from disk to memory
  - means fewer TLB misses
- but also
  - wastes memory (internal fragmentation)
  - increases the time to start up a program
### VM and Disk: Page replacement policy

**Page Table**

<table>
<thead>
<tr>
<th>dirty</th>
<th>used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</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>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Set of all pages in Memory**

**Head pointer**

Place pages on free list if used bit is still clear. Schedule pages with dirty bit set to be written to disk.

**Tail pointer**

Clear the used bit in the page table.

**Hardware Architect’s role:** support setting dirty and used bits

- **Free list:** Support setting dirty and used bits
- **Tail pointer:** Clear the used bit in the page table
- **Head pointer:** Place pages on free list if used bit is still clear. Schedule pages with dirty bit set to be written to disk.

### MIPS Address Translation: How does it work?

**“Virtual Addresses”**

- CPU
- Data
- Translation Look-Aside Buffer (TLB)
- Memory

**“Physical Addresses”**

- Physical and virtual pages must be the same size!

**Translation Look-Aside Buffer (TLB)**

A small fully-associative cache of mappings from virtual to physical addresses

- TLB also contains protection bits for virtual address

**Fast common case:** Virtual address is in TLB, process has permission to read/write it.

### The TLB caches page table entries

**Virtual Address**

- **V page no.**
- **offset**

**Physical Address**

- **V=0** pages either reside on disk or have not yet been allocated.
- **OS handles V=0 “Page fault”**

**Physical frame address**

Even a cache hit requires TLB translation first!

### Common Organization

- **CPU**
- **TLB**
- **L1 Cache**
- **Write Buffer**
- **L2 Cache**
- **Memory bus**

**MIPS** handles TLB misses in software (random replacement). Other machines use hardware.
Can TLB and caching be overlapped?

<table>
<thead>
<tr>
<th>Virtual Page Number</th>
<th>Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Byte Select</td>
</tr>
<tr>
<td>Cache Tags</td>
<td>Valid</td>
</tr>
<tr>
<td>Cache Tag</td>
<td>Hit</td>
</tr>
<tr>
<td>Cache Block</td>
<td></td>
</tr>
<tr>
<td>Cache Block</td>
<td></td>
</tr>
<tr>
<td>Cache Block</td>
<td></td>
</tr>
<tr>
<td>Data out</td>
<td></td>
</tr>
</tbody>
</table>

This works, but ...

Q. What is the downside?

A. Inflexibility. Size of cache limited by page size.

Use virtual addresses for cache?

“Virtual Addresses”

A0-A31

CPU

Virtual Cache

D0-D31

“A0-A31”

“Physical Addresses”

A0-A31

Main Memory

D0-D31

Virtual Translation Look-Aside Buffer (TLB)

Only use TLB on a cache miss!

Downside: a subtle, fatal problem. What is it?

A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a nightmare.

Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation.

This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache.

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

This bit is changed by VA translation, but is needed for cache lookup.

Solutions:

- go to 8K byte page sizes;
- go to 2 way set associative cache;
- or SW guarantee VA[13]=PA[13]

Paging vs. segmentation – Fig. C.21

<table>
<thead>
<tr>
<th>Page</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words per address</td>
<td>One</td>
</tr>
<tr>
<td>Programmer visible?</td>
<td>Invisible to app programmer</td>
</tr>
<tr>
<td>Replacing a block</td>
<td>Trivial (all blocks same size)</td>
</tr>
<tr>
<td>Memory use inefficiency</td>
<td>Internal fragmentation (within page)</td>
</tr>
<tr>
<td>Efficient disk traffic</td>
<td>Yes (can adjust page size)</td>
</tr>
</tbody>
</table>
Summary #1/3: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation

- The optimal choice is a compromise
  - depends on access characteristics
    » workload
    » use (I-cache, D-cache, TLB)
  - depends on technology / cost

- Simplicity often wins

Summary #2/3: Caches

- The Principle of Locality:
  - Program accesses a relatively small portion of the address space in any short interval of time.
    » Temporal Locality: Locality in Time
    » Spatial Locality: Locality in Space

- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!

- Write Policy: Write Through vs. Write Back

- Today CPU time is a function of (ops, cache misses) vs. just f(ops): affects Compilers, Data structures, and Algorithms

Summary #3/3: TLB, Virtual Memory

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!
- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- Today VM allows many processes to share single memory without having to swap all processes to disk: today VM protection is maybe even more important than memory hierarchy benefits, but computers still insecure