Translation Buffers (TLB’s)

• To perform virtual to physical address translation we need to look-up a page table
• Since page table is in memory, need to access memory
  – Much too time consuming; 50 cycles or more per memory reference
• Hence we need to cache the page tables
• To that effect special purpose caches named translation buffers
  – Also named Translation Lookaside Buffers (TLBs)
TLB organization

• TLB organized as caches
• Therefore for each entry in the TLB we’ll have
  – a tag to check that it is the right entry
  – data which instead to be the contents of memory locations, like in a cache, will be a page table entry (PTE)
• TLB’s are smaller than caches
  – 32 to 128 entries
  – from fully associative to direct-mapped
  – there can be an instruction TLB, a data TLB and also distinct TLB’s for user and system address spaces
TLB organization

Virtual page number → Offset

tag → Index

Copy of PTE

v d prot

Physical frame number
From virtual address to memory location (highly abstracted; revisited)
Address translation

• At each memory reference the hardware searches the TLB for the translation
  – TLB hit and valid PTE the physical address is passed to the cache
  – TLB miss, either hardware or software (depends on implementation) searches page table in memory.
    • If PTE is valid, contents of the PTE loaded in the TLB and back to step above
• In hardware the TLB miss takes 10-100 cycles
• In software takes up to 100 -1000 cycles
• In either case, no context-switch
  – Context-switch takes more cycles than a TLB miss
• If PTE is invalid, we have a page fault (even on a TLB hit)
TLB Management

• TLB’s organized as caches
  – If small could be fully associative
  – Current trend: larger (about 128 entries); separate TLB’s for instruction and data; Some part of the TLB reserved for system
  – TLB’s are write-back. The only thing that can change is dirty bit + any other information needed for page replacement algorithm (cf. CSE 451)

• MIPS 3000 TLB (old)
  – 64 entries: fully associative. “Random” replacement; 8 entries used by system
  – On TLB miss, we have a trap; software takes over but no context-switch
TLB management (ct’d)

• At context-switch, the virtual page translations in the TLB are not valid for the new task
  – Invalid the TLB (set all valid bits to 0)
  – Or append a Process ID (PID) number to the tag in the TLB. When a new task takes over, the O.S. creates a new PID.
  – PID are recycled and entries corresponding to “old PID” are invalidated.
Paging systems -- Hardware/software interactions

• Page tables
  – Managed by the O.S.
  – Address of the start of the page table for a given process is found in a special register which is part of the state of the process
  – The O.S. has its own page table
  – The O.S. knows where the pages are stored on disk

• Page fault
  – When a program attempts to access a location which is part of a page that is not in main memory, we have a page fault
Page fault detection (simplified)

• Page fault is an *exception*
• Detected by the hardware (invalid bit in PTE either in TLB or page table)
• To resolve a page fault takes millions of cycles (disk I/O)
  – The program that has a page fault must be interrupted
• A page fault occurs in the middle of an instruction
  – In order to restart the program later, the state of the program must be saved and instructions must be restartable (precise exceptions)
• State consists of all registers, including PC and special registers (such as the one giving the start of the page table address)
Page fault handler (simplified)

- Page fault exceptions are cleared by an O.S. program called the page fault handler which will
  - Grab a physical frame from a free list maintained by the O.S.
  - Find out where the faulting page resides on disk
  - Initiate a read for that page
  - Choose a frame to free (if needed), i.e., run a replacement algorithm
  - If the replaced frame is dirty, initiate a write of that frame to disk
  - Context-switch, i.e., give the CPU to a task ready to proceed
Completion of page fault

• When the faulting page has been read from disk (a few ms later)
  – The disk controller will raise an *interrupt* (another form of exception)
  – The O.S. will take over (context-switch) and modify the PTE (in particular, make it valid)
  – The program that had the page fault is put on the queue of tasks ready to be run
  – Context-switch to the program that was running before the interrupt occurred
## Two extremes in the memory hierarchy

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>L1</th>
<th>PAGING SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>block (page) size</td>
<td>16-64 bytes</td>
<td>4K-8K (also 64K)</td>
</tr>
<tr>
<td>miss (fault) time</td>
<td>10-100 cycles</td>
<td>Millions of cycles</td>
</tr>
<tr>
<td></td>
<td>(20-1000 ns)</td>
<td>(3-20 ms)</td>
</tr>
<tr>
<td>miss (fault) rate</td>
<td>1-10%</td>
<td>0.00001-0.001%</td>
</tr>
<tr>
<td>memory size</td>
<td>4K-64K Bytes</td>
<td>Gigabytes</td>
</tr>
<tr>
<td></td>
<td>(impl. depend.)</td>
<td>(depends on ISA)</td>
</tr>
</tbody>
</table>
Other extreme differences

• Mapping: Restricted (L1) vs. General (Paging)
  – Hardware assist for virtual address translation (TLB)

• Miss handler
  – Hardware only for caches
  – Software only for paging system (context-switch)
  – Hardware and/or software for TLB

• Replacement algorithm
  – Not that important for caches
  – Very important for paging system

• Write policy
  – Always write back for paging systems
Some optimizations

• Speed-up of the most common case (TLB hit + L1 Cache hit)
  – Do TLB look-up and cache look-up in parallel
    • possible if cache index independent of virtual address translation
      (good only for small caches)
  – Have cache indexed by virtual addresses but with physical tags
  – Have cache indexed by virtual addresses but with virtual tags
    • these last two solutions have additional problems referred to as synonyms