Virtual page number

Page table

Physical page or disk address

Valid

0

1

1

1

1

0

1

1

0

1

Physical memory

Disk storage

Virtual Address

VPN

Page Offset

Translation

PPN

Page Offset

Physical Address

30 29 28 ... 14 13 12 11 10 9 ... 2 1 0

19

12

15
Replacement and writes

• **Replacement:** To reduce page fault rate, prefer least-recently used (LRU) replacement
  – Reference bit (aka use bit) in PTE set to 1 on access to page
  – Periodically cleared to 0 by OS
  – A page with reference bit = 0 has not been used recently

• **Writes:** Disk writes take millions of cycles
  – Block at once, not individual locations
  – Write through is impractical; use write-back
  – Dirty bit in PTE set when page is written
  – Write a physical page and reloading a different virtual page is called **swapping**
Fast translation using TLB

• Use a translation lookaside buffer (TLB)
  – Small cache of most recent translations
  – Reduces number of memory accesses required for most loads/stores from two to one

• TLB
  – Small: accessed in < 1 cycle
  – Typically 16 - 512 entries
  – Fully associative
  – > 99 % hit rates typical
  – Reduces # of memory accesses for most loads and stores from 2 to 1
Example: Two-entry TLB

Virtual Page Number | Physical Page Number
---|---
0x00002 | 0x0000

Virtual Page Number | Physical Page Number
---|---
0x7FFFD | 0x0000

Virtual Page Number | Physical Page Number
---|---
0x00002 | 0x7FFF

Hit

Virtual Address: 0x00002
Page Offset: 47C

Entry 1:
Virtual Page Number: 0x7FFFD
Physical Page Number: 0x0000

Entry 0:
Virtual Page Number: 0x00002
Physical Page Number: 0x7FFF

TLB

Hit1

Physical Address: 0x7FFF 47C
TLB misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler

- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction
Memory protection

- Multiple programs (*processes*) run at once
- Each process has its own page table – special register holds the starting address of the table
- Each process can use entire virtual address space without worrying about where other programs are
- A process can only access physical pages mapped in its page table – can’t overwrite memory from another process

- **Modifications to TLB:**
  - Invalidate the TLB every time there is a context switch
  - Or augment TLB with process id
1. TLB/Cache: physical tag, physical index

- Virtual address: 20 bits
- Page offset: 12 bits
- 4 KB page size

Translation in TLB:
- 1 cycle

Cache:
- 16 KB cache size
- 4 bytes / block
- Direct mapped
- 4096 blocks

- 1 cycle
- Hit: data
- Tag

S. Reda EN164 Sp ’11
2. TLB/Cache: virtual tag, virtual index

- TLB is only accessed in case of miss
- Complications due to aliasing: Different virtual addresses for shared physical address
3. TLB/Cache: physical tag, virtual index

- Virtual address: 20 bits
- Page offset: 12 bits
- 4 KB page size
- 4 KB cache size
- 4 bytes / block
- Direct mapped
- 1024 blocks
- First design attempt

1 cycle

Translate TLB

1 cycle

Data

Hit
3. TLB/Cache: physical tag, virtual index

- TLB can now be accessed in parallel with cache
- Constrain cache design: *fundamental* cache size (# sets * block size = page size).
- Only way to increase cache size is by increasing associativity
Virtual memory summary

• Virtual memory increases **capacity and essential for multitasking OS**
• A subset of virtual pages are located in physical memory
• A **page table** maps virtual pages to physical pages – this is called address translation
• A **TLB** speeds up address translation
• Using different page tables for different programs provides **memory protection**
Pentium Pro memory hierarchy

- **Address Size:** 32 bits
- **VM Page Size:** 4 KB, 4 MB
- **TLB organization:** separate I and D TLBs (I-TLB: 32 entries, D-TLB: 64 entries) 4-way set associative LRU approximated hardware handles miss

- **L1 Cache:** 8 KB, separate I, D 4-way set associative LRU 32 byte block write back

- **L2 Cache:** 256 or 512 KB
Memory subsystem summary

- Fast memories are small, large memories are slow
  - We really want fast, large memories
  - Caching gives this illusion

- Principle of locality
  - Programs use a small part of their memory space frequently

- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk

- Virtual memory