Virtual Memory Review

- Goal: give illusion of a large memory
- Allow many processes to share single memory
- Strategy
  - Break physical memory up into blocks (*pages*)
  - Page might be in physical memory or on disk.
- Addresses:
  - Generated by lw/sw: *virtual*
  - Actual memory locations: *physical*

Memory access

- Load/store/PC computes a *virtual* address
- Need address translation
  - Convert virtual addr to physical addr
  - Use page table for lookup
  - Check virtual address:
    - If page is in memory, access memory with physical address
      - May also need to check access permissions
    - If page is in not in memory, access disk
      - Page fault
      - Slow – so run another program while it's doing that
  - Do translation in hardware
    - Software translation would be too slow!

Handling a page fault

- Occurs during memory access clock cycle
- Handler must:
  - Find disk address from page table entry
  - Choose physical page to replace
    - if page dirty, write to disk first
  - Read referenced page from disk into physical page

TLB: Translation Lookaside Buffer

- Address translation has a high degree of locality
  - If page accessed once, highly likely to be accessed again soon.
  - So, cache a few frequently used page table entries
- TLB = hardware cache for the page table
  - Make translation faster
  - Small, frequently fully-associative
- TLB entries contain
  - Valid bit
  - Other housekeeping bits
  - Tag = virtual page number
  - Data = Physical page number
- Misses handled in hardware (dedicated FSM) or software (OS code reads page table)

TLB Misses

- TLB miss means one of two things
  - Page is present in memory, need to create the missing mapping in the TLB
  - Page is not present in memory (page fault), need to transfer control to OS to deal with it.
    - Need to generate an exception
    - Copy page table entry to TLB – use appropriate replacement algorithm if you need to evict an entry from TLB.

Optimizations

- Make the common case fast
- Speed up TLB hit + L1 Cache hit
  - Do TLB lookup and cache lookup in parallel
    - Possible if cache index is independent of virtual address translation
  - Have cache indexed by virtual addresses

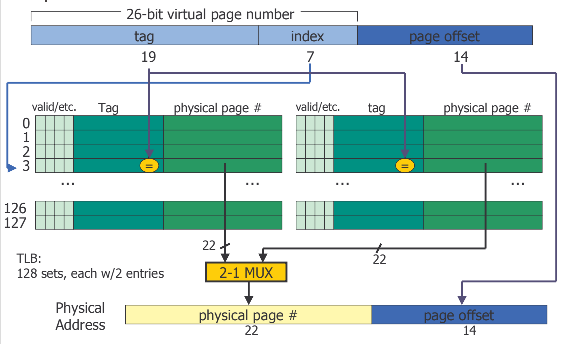
TLB Example - 7.39, 7.40

- Given:
  - 40-bit virtual addr
  - 16KB pages
  - 36-bit physical byte address
  - 2-way set associative TLB with 256 total entries
- Total page table size?
- Memory organization?

Page table/address parameters

- 16KB=2^14, so 16K pages need 14 bits for an offset inside a page.
- The rest of the virtual address is the virtual page index, and it's 40-14 = 26 bits long, for 2^26 page table entries.
- Each entry contains 4 bits for valid/protection/dirty information, and the physical frame number, which is 36-14 = 22 bits long, for a total of 26 bits.
- The total page table size is then 26 bits * 2^26 entries = 208 MB

Memory organization with TLB


26-bit virtual page number
tag
index
page offset
19
7
14
valid/etc.
Tag
physical page #
valid/etc.
tag
physical page #
0
1
2
3
=
=
...
126
127
22
22
TLB:
128 sets, each w/2 entries
2-1 MUX
Physical Address
physical page #
22
page offset
14