LECTURE - 21

Topics for Today

- Hit-time reduction techniques
- Virtual memory
-
- ***Scribe for today?***

Small and Simple Caches

- Keep the cache small
  - Faster
  - Can fit inside processor
  - Trade-off: tags within processor, data outside
- Keep the cache simple
  - Direct-mapped ==> tag comparison can be in parallel with data transmission

Other Techniques

- Faster writes: pipeline writes
  - Split the tag and data storage in cache
  - Pipeline stage-1: tag access and comparison
  - Pipeline stage-2: write data
- Dealing with virtual address --> physical address translation
  - Avoid it (virtually addressed caches)
  - In parallel with cache access (virtually indexed, physically tagged cache)

Virtual Memory

- Another level in the hierarchy
- Uses of virtual memory:
  - *Level of indirection*
    - No program overlays required
    - Easy relocation
  - Sharing and protection

Just like Memory-->Cache in Functionality...

- Cache line
- Cache miss
- Memory --> cache mapping

- Page or segment
- Page fault
- VA --> PA mapping (address translation)

But Quite Different Quantitatively...

Parameter	Memory --> Cache	VM --> Ph. Memory
Hit time	1-2 Cycles	40-100 Cycles
Miss penalty	8-100 Cycles	O(10ms-100ms)
Miss rate	0.5-10%	0.00001-0.001%
Block/Page size	16-128 Bytes	4-64KB
Upper level size	16KB-1MB	O(1GB)

- Page faults handled in software
  - Be very careful about what you discard
  - Lots of time anyway
- VM size determined by ISA
- VM is not quite the hard-disk...

Paging versus Segmentation

Criterion	**Paging**	**Segmentation**
Block size	Uniform (4 to 64KB)	Variable (max: 2^16-2^32, min: 1byte)
Words per address	One	Two
Programmer visible?	No	Perhaps
Block replacement	Easy	Need to find contiguous memory
Memory use inefficiency	Internal fragmentation	External fragmentation
Efficient disk traffic?	Usually yes (for appropriate page size)	Not for small segments

- Other possibilities:
  - Paged segments
  - Choices for page size

The Four Memory Hierarchy Questions

- Where to place a block?
  - Fully associative
- How to find a block in main memory?
  - Page table, or inverted table; cached in TLB
- Which block to replace?
  - LRU, with the help of a *use/reference* bit
- What happens on write?
  - Write-back, write-allocate

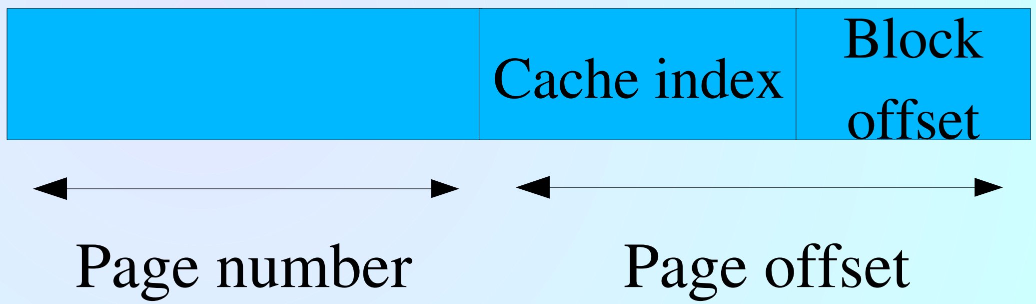
Trade-Offs in Page-Size

- Large page size good for:
  - Smaller page tables
  - Lesser TLB miss rate
  - Efficient disk or network transfer
  - Faster cache hits (how?)
- Smaller page size good for:
  - Efficient use of memory (lesser fragmentation)
  - Faster process startup time

Fast Translation

- Translation Look-aside Buffer (TLB)
  - Small table in hardware
  - Fully associative
  - Fields:
    - The translation, valid bit, use bit, dirty bit, protection bits
- TLB access can be in critical path
  - Pipeline TLB access
  - Overlap cache tag access with translation!

Overlapping Tag Access with Translation



Tag access through index is independent of translation

- Cache index is virtual, but tags are physical
  - This limits cache size potentially
  - Solutions possible:
    - Higher associativity
    - Page colouring (set associativity)
    - Small guessing hardware

Alternate Strategy: Avoid Translation!

- Virtually addressed caches:
  - Cache is accessed using the virtual address
- Advantage: faster hit time
- Disadvantages:
  - Cache has to be flushed on process switch
  - What if two different VAs for the same PA?
    - Synonyms/aliases
  - I/O usually uses PA to access memory/cache

Dealing with Virtually Addressed Caches

- Avoiding cache flush:
  - Include a PID field in cache tag
- Anti-aliasing
  - Page colouring (set associativity)
  - Create "enough" colours (sets) to ensure that cache size <= block-size x number of sets
  - Cache has to be direct-mapped