Principle of Locality: Memory Hierarchies

- Text and data are not accessed randomly
- Temporal locality
  - Recently accessed items will be accessed in the near future (e.g., code in loops, top of stack)
- Spatial locality
  - Items at addresses close to the addresses of recently accessed items will be accessed in the near future (sequential code, elements of arrays)
- Leads to memory hierarchy at two main interface levels:
  - Processor - Main memory -> Introduction of caches
  - Main memory - Secondary memory -> Virtual memory (paging systems)

Processor - Main Memory Hierarchy

- Registers: Those visible to ISA + those renamed by hardware
- (Hierarchy of) Caches: plus their enhancements
  - Write buffers, victim caches etc…
- TLB's and their management
- Virtual memory system (O.S. level) and hardware assists (page tables)
- Inclusion of information (or space to gather information) level per level
  - Almost always true

Questions that Arise at Each Level

- What is the unit of information transferred from level to level ?
  - Word (byte, double word) to/from a register
  - Block (line) to/from cache
  - Page table entry + misc. bits to/from TLB
  - Page to/from disk
- When is the unit of information transferred from one level to a lower level in the hierarchy?
  - Generally, on demand (cache miss, page fault)
  - Sometimes earlier (prefetching)

Questions that Arise at Each Level (c'ed)

- Where in the hierarchy is that unit of information placed?
  - For registers, directed by ISA and/or register renaming method
  - For caches, in general in L1
    - Possibility of hinting to another level (Itanium) or of bypassing the cache entirely, or to put in special buffers
- How do we find if a unit of info is in a given level of the hierarchy?
  - Depends on mapping;
  - Use of hardware (for caches/TLB) and software structures (page tables)

Questions that Arise at Each Level (c'ed)

- What happens if there is no room for the item we bring in?
  - Replacement algorithm; depends on organization
- What happens when we change the contents of the info?
  - i.e., what happens on a write?

Caches (on-chip, off-chip)

- Caches consist of a set of entries where each entry has:
  - line (or block) of data: information contents
  - tag: allows to recognize if the block is there
  - status bits: valid, dirty, state for multiprocessors etc.
- Cache Geometries
  - Capacity (or size) of a cache: number of lines * line size, i.e., the cache metadata (tag + status bits) is not counted
  - Associativity
  - Line size

Cache Organizations

- Direct-mapped
- Set-associative
- Fully-associative

Cache Hit or Cache Miss?

- How to detect if a memory address (a byte address) has a valid image in the cache:
- Address is decomposed in 3 fields:
  - *line offset* or *displacement* (depends on line size)
  - *index* (depends on number of sets and set-associativity)
  - *tag* (the remainder of the address)
- The tag array has a width equal to *tag*

Hit Detection

tag	index	displ.

Example: cache capacity C, line size b

Direct mapped: displ = $\log_2 b$; index = $\log_2(C/b)$; tag = 32 -index - displ

N -way S.A: displ = $\log_2 b$; index = $\log_2(C/bN)$; tag = 32 -index - displ

So what does it mean to have 3-way (N=3) set-associativity?

Replacement Algorithm

- None for direct-mapped
- Random or LRU or pseudo-LRU for set-associative caches
  - Not an important factor for performance for low associativity. Can become important for large associativity and large caches