Lecture 27
July 14

Cache Memory

Direct cache

- Use least significant bits of word address as "index"
    - Other bits are the "tag"
- If the processor requests a word from RAM
    - Take its index look in that position in the cache
    - Check to see if the tag in the cache matches the tag of the address
    - Match: return value from cache (hit)
    - No match: go to memory & store in cache (miss)
- In the example (8-bit address, 3 bit index) the cache looks like:

Index	Tag	Data
000	10101	…….
001		
010		
011		
…		
111		

- If Tag||Index is the memory address we want, fetch from Data
    - I.e. Memory 10101000 is cached, 11111000 isn't
- Memory address with the same index can't be cached
    - Very bad if you have code & data with the same index
- Fully associative cache
    - We could use the entire address as the tag & cache words anywhere:

Tag	Data
10101	…….

- Gives a lot more freedom when deciding what to cache
- Problem: if the CPU requests a particular word, how do we decide if/where it is in the cache?
    - It would take too long to search each tag for a match

- Solution: store tags m "associative memory"

- Memory with comparison circuitry added
- Each stored tag is compared to the memory address in parallel
  - Each tag has a comparison circuit to do it
- But, the extra circuitry for associative memory is expensive

– Cache replacement
  - With fully associative cache, there is more freedom in deciding what o replace
  - Some possible STRAEGIES:
    - Least recent used (LRU)
    - Random
    - FIFO(first-in first-out)
  - LRU is the best of these
    - but, its difficult to implement
    - with each, update a time stamp
    - when replacing, find the oldest time stamp
  - if used, LRU is approximated
– set –associative code is a cheaper alternative
– start with direct mapped cache
  - instead of one tag/data pair per index, create several
  - e.g. For each tag allow 4 tags:
  -

Index	Tag1	Tag2	Tag3	Tag4	Data
000					…….

  - For each read/replacement we only have to check the corresponding index's tag
    - Doing 4 comparisons is possible
  - 2-way or 4-way set associative cache is common in modern architectures

Cache variations:

– instruction/data cache
  - some architectures use separate caches for instruction & data reads in a single cycle
– several levels of cache
  - some architectures have different levels of memory cache
    - e.g. 64KB of L1 cache that reads in 1 cycle 1MB of L2 cache that reads in 2 cycles

CPU <=> L1 <=> L2 <=> memory