Announcements

- Reading for this lecture:
  - Mano section 12-5 to 12-7

Questions about Cache Memory

- How big is the cache?
- How many caches?
  - first level caches are typically 8 to 64k (often on the processor chip)
    - due to propagation delays
      - in 1 ns electricity only travels about a foot
      - current clock speed on available machines is < 3ns
  - second level caches can be over a megabyte
  - can have separate cache for instructions and data
- What to do if there is no free space in the cache?
  - need to remove something (called replacement or eviction)
  - FIFO, LRU, and random are all used

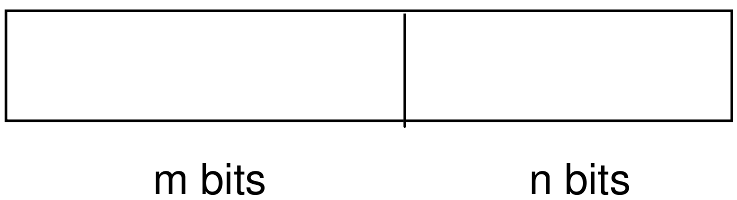
Finding data in the Cache 1

- Associative mapping
  - a table with two values per item (address and value)
  - hardware looks in all locations at once
  - very fast, but expensive

Address #1	Value #1
Address #2	Value #2
Address #3	Value #3
Address #n	Value #n

Finding data in the Cache 2

- Direct mapped
  - use part of the address to decide where in the cache to look
  - can use normal SRAM memory for the cache
  - can get *conflicts* since the same item in the cache is used for several locations



m bits

2^2 lines

Tag #1	Value #1
Tag #2	Value #2
Tag #3	Value #3
Tag #2^n	Value #2^n

Finding data in the Cache 3

- Set Associative
  - combines direct and associated mappings
  - each item has a set of <addr, value> tuples
    - use associate lookup within the item
  - each address maps to one item

2^n lines

Tag #1	Value #1	Tag #1	Value #1
Tag #2	Value #2	Tag #2	Value #2
Tag #3	Value #3	Tag #3	Value #3
Tag #2^n	Value #2^n	Tag #2^n	Value #2^n

Two-way set associative cache

Writing to Cached Memory

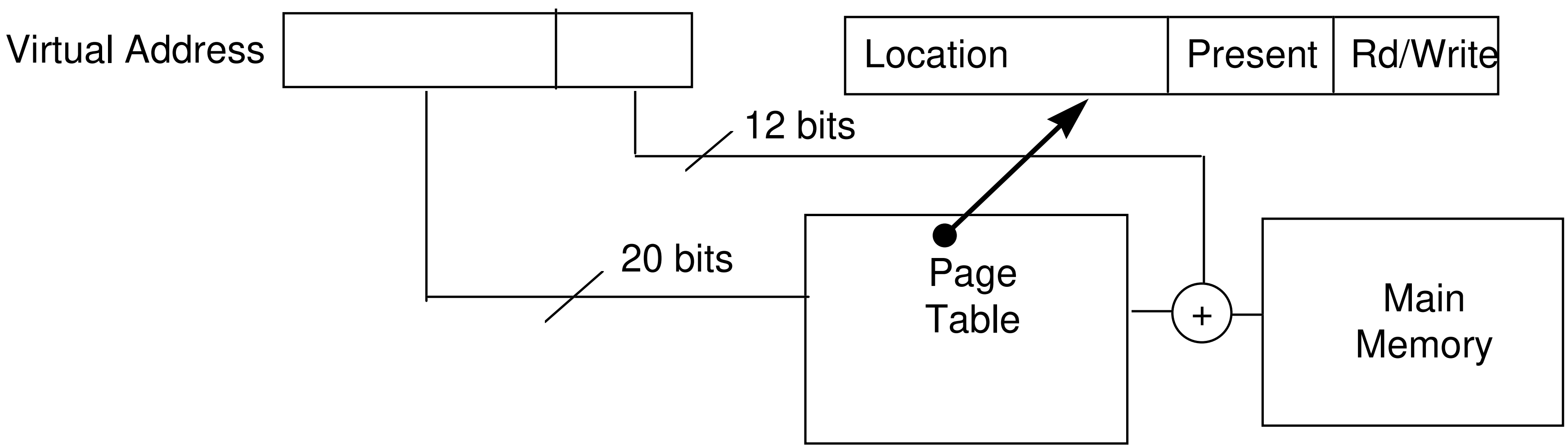
- What happens when memory is written?
- Can write to memory directly
  - called a write-through cache
- Can write to memory only when data leaves cache
  - called write-back cache
- What about when multiple devices (processors, disks, etc.) try to use the same memory?
  - need to develop ways to keep the memory consistent

Managing Memory

- Main memory is big, but what if we run out
  - use virtual memory
  - keep part of memory on disk
    - bigger than main memory
    - slower than main memory
- Want to have several program in memory at once
  - keeps processor busy while one process waits for I/O
  - need to protect processes from each other
  - have several tasks running at once
    - compiler, editor, debugger
    - word processing, spreadsheet, drawing program
- Use *virtual addresses*
  - look like normal addresses
  - hardware translates them to *physical addresses*

Virtual Addressing

- Divide physical memory into fixed sized chunks called pages
  - typical pages are 512 bytes to 64k bytes
- Have a table to map virtual pages to physical pages
- Consider a 32 bit addresses
  - 4096 byte pages (12 bits for the page)
  - 20 bits for the page number



Advantages of Virtual Addressing

- Can assign non-contiguous regions of physical memory to programs
- A program can only gain access to its mapped pages
- Can have more virtual pages than the size of physical memory
- Every program can start at (virtual) address 0

Problems with Page Tables

- One page table can get very big
  - 2^{20} entries (for most programs, most items are empty)
  - solution use a hierarchy of page tables