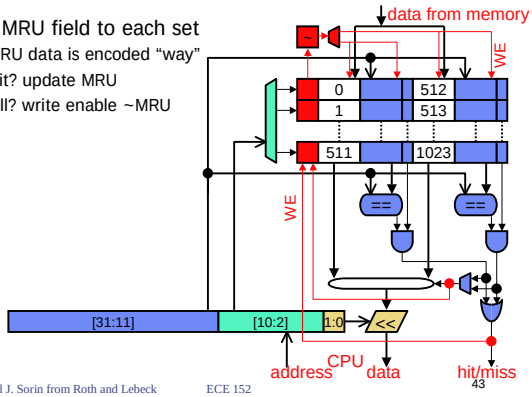
Cache Replacement Policies

- Set-associative caches present a new design choice
  - On cache miss, which block in set to replace (kick out)?
- Some options
  - **Random**
  - **FIFO (first-in first-out)**
    - When is this a good idea?
  - **LRU (least recently used)**
    - Fits with temporal locality, LRU = least likely to be used in future
  - **NMRU (not most recently used)**
    - An easier-to-implement approximation of LRU
    - NMRU=LRU for 2-way set-associative caches
  - **Belady's**: replace block that will be used furthest in future
    - Unachievable optimum (but good for comparisons)
  - Which policy is simulated in previous slide?

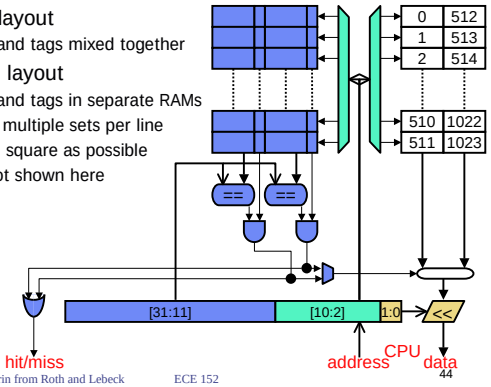
NMRU and Miss Handling

- Add MRU field to each set
  - MRU data is encoded "way"
  - Hit? update MRU
  - Fill? write enable ~MRU

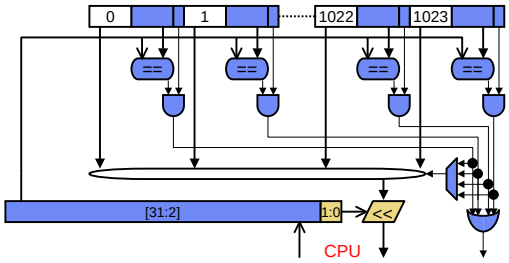


Physical Cache Layout

- Logical layout
  - Data and tags mixed together
- Physical layout
  - Data and tags in separate RAMs
  - Often multiple sets per line
    - As square as possible
    - Not shown here



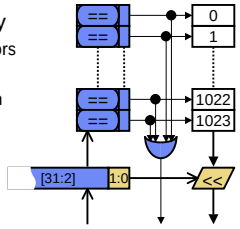
Full-Associativity



- How to implement full (or at least high) associativity?
  - Doing it this way is terribly inefficient
  - 1K matches are unavoidable, but 1K data reads + 1K-to-1 mux?

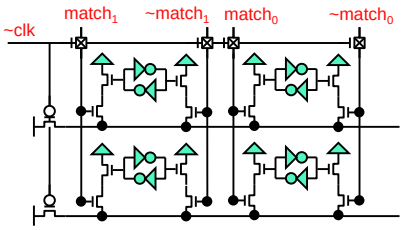
Full-Associativity with CAMs

- **CAM**: content addressable memory
  - Array of words with built-in comparators
  - Matchlines instead of bitlines
  - Output is "one-hot" encoding of match
- FA cache?
  - Tags as CAM
  - Data as RAM



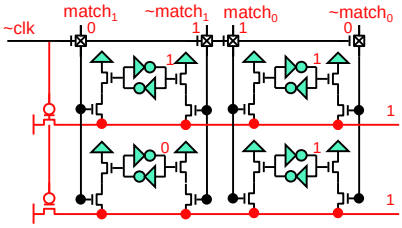
- **Hardware is not software**
  - Example I: parallel computation with carry speculate adder
  - Example II: parallel search with CAM
    - No such thing as software CAM

CAM Circuit



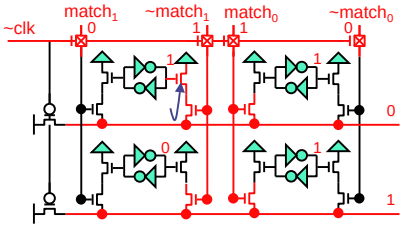
- **Matchlines** (correspond to bitlines in SRAM): inputs
- **Wordlines**: outputs
- Two phase match
  - Phase I: clk=1, pre-charge wordlines to 1
  - Phase II: clk=0, enable matchlines, non-matched bits dis-charge wordlines

CAM Circuit In Action



- Phase I: clk=1
  - Pre-charge wordlines to 1

CAM Circuit In Action



Looking for match with 01

- Phase I: clk=0
  - Enable matchlines (notice, match bits are flipped)
  - Any non-matching bit discharges entire wordline
    - Implicitly ANDs all bit matches (NORs all bit non-matches)
  - Similar technique for doing a fast OR for hit detection

CAM Upshot

- CAMs are effective but expensive
  - Matchlines are very expensive (for nasty circuit-level reasons)
  - CAMs are used but only for 16 or 32 way (max) associativity
    - See an example soon
  - Not for 1024-way associativity
    - No good way of doing something like that
    + No real need for it either

Analyzing Cache Misses: 3C Model

- Divide cache misses into three categories
  - **Compulsory (cold)**: never seen this address before
    - Easy to identify
  - **Capacity**: miss caused because cache is too small
    - Consecutive accesses to block separated by accesses to at least N other distinct blocks where N is number of frames in cache
  - **Conflict**: miss caused because cache associativity is too low
    - All other misses

Cache Performance Simulation

- Parameters: 8-bit addresses, 32B cache, 4B blocks
  - Initial contents : 0000, 0010, 0020, 0030, 0100, 0110, 0120, 0130
  - Initial blocks accessed in increasing order

Cache contents	Address	Outcome
0000, 0010, 0020, 0030, 0100, 0110, 0120, 0130	3020	Miss (compulsory)
0000, 0010, **3020**, 0030, 0100, 0110, 0120, 0130	3030	Miss (compulsory)
0000, 0010, 3020, **3030**, 0100, 0110, 0120, 0130	2100	Miss (compulsory)
0000, 0010, 3020, 3030, **2100**, 0110, 0120, 0130	0012	Hit
0000, 0010, 3020, 3030, 2100, 0110, 0120, 0130	0020	Miss (capacity)
0000, 0010, **0020**, 3030, 2100, 0110, 0120, 0130	0030	Miss (capacity)
0000, 0010, 0020, **0030**, 2100, 0110, 0120, 0130	0110	Hit
0000, 0010, 0020, 0030, 2100, 0110, 0120, 0130	0100	Miss (capacity)
0000, 1010, 0020, 0030, **0100**, 0110, 0120, 0130	2100	Miss (conflict)
1000, 1010, 0020, 0030, **2100**, 0110, 0120, 0130	3020	Miss (capacity)

ABC

- **Associativity** (increase)
  + Decreases conflict misses
  – Increases t_{hit}
- **Block size** (increase)
  – Increases conflict misses
  + Decreases compulsory misses
  ± Increases or decreases capacity misses
  • No effect on t_{hit}
- **Capacity** (increase)
  + Decreases capacity misses
  – Increases t_{hit}