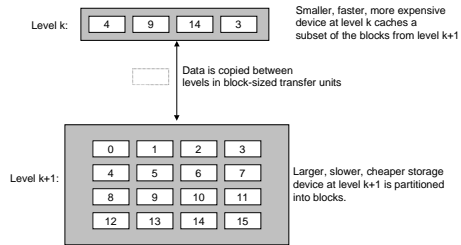


Cache Memory: Issues



- [O'H&Bryant, fig 6.22
- issue: cache must store the memory address (as a 'tag') of the data, as well as the data itself!
- issue: (considering the cache to be organized as an array) which cache entry ('index') holds the data for address X (if any)?
- idea (direct-mapped cache: for a cache of size $C' = 2^c$ entries, all addresses with same a_1 are mapped to the same entry

$$X = \begin{array}{|c|c|} \hline 31 & c-1 \\ \hline a_0 & a_1 \\ \hline \end{array} 0$$

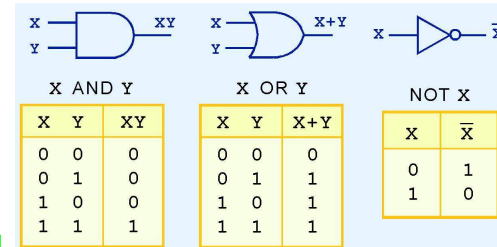
Cache Memory: Details: Associativity and Replacement Policy

- cache hit rates: % of (word) accesses in program when data is in cache
 - needs to be high (e.g. > 95%) for good performance!
- problem: the data for addresses with the same a_1 cannot be in the cache simultaneously!
- solution (K-way set associative caches): each entry can hold a set of K data items (blocks) (typically $K = 1, 2, 4, 8$)
 - addresses with the same a_1 can map into any of the K items in the set
- issue (replacement policy): if all items in the set contain data, what happens when we access a new address X' with the same a_1 ?
 - random: choose any item, and put data and tag for X' there
 - least recently used: choose the item that was *least recently accessed* to replace

Q: which is likely to be better? any downsides?
- cache behavior (hit rates are not always simple to analyze!
 - the Random Cache\$ for Beer Challenge!

The Digital Logic Level: Gates

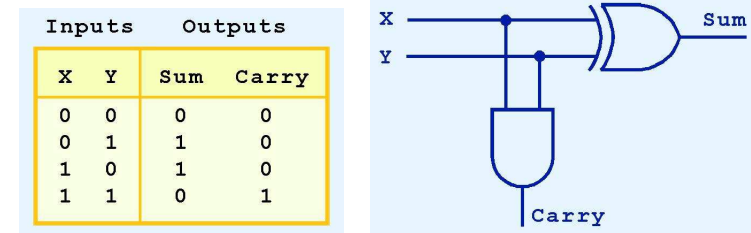
- Ref: [Null&Lobur, sect 2.3–2.4], [Tanenbaum, ch 3]
- digital logic is lowest level of computer organization!
- digital circuit elements (including 1-bit memory cells) are made from gates (in turn, from 1 or more transistors or switches, [Tanenbaum, fig 3.1])
- [Null&Lobur, figs 3.1-2]: AND and OR gates manipulate voltage levels (0- low, 1 - high) according to truth tables (Boolean logic)



- [O'H&Bryant, p 45]: set of colors forming an 8-element boolean algebra

The Digital Logic Level: Half Adders

- from gates, higher-level components such as a half-adder are constructed ([Null&Lobur, figs 3.10,3.11]: Sum = X XOR Y, Carry = X AND Y)



- this can be extended to a full adder ([Null&Lobur, fig 3.13]) (truth table, circuit)
 - how could this be used to implement an integer add circuit?
- discussion point: why is binary the representation used in computers?