CAM
Content Addressable Memory

For TAG look-up
in a
Fully-Associative Cache
Fully Associative Cache
Tag search Using CAM

• CAM: Content Addressable Memory
  – Input: Data
  – Output: Address
    It searches for the content (input data) and provides the address of the location with that content

• Tag search using CAM
  – Input: Tag portion of the block address requested
  – Output: One hot coded or encoded binary address of where the tag is found => i.e. the address of the data in the data RAM
Tag search using CAM

- At most one match signal is high
- Match signal determines the correct line in DATA RAM
- One hot encoded address for Data RAM
Interfacing with Data RAM

- Use binary encoding to generate address from Tag RAM.
- Use that address to fetch data from Data RAM.
- \((\text{Hit} == 1) \Rightarrow \text{Address is valid}\)
- \((\text{Hit} == 0) \Rightarrow \text{Address is invalid}\)
Updating CAM

- CAM should be updated on a cache miss
- Use Write Enable Signal
- Input: Location to write
- Input: Tag to be written

Note: Address to write can be one of the empty locations address if the cache is not full. If the cache is full, the “Victim” address is chosen using either RANDOM replacement algorithm or LRU (Least Recently Used) replacement algorithm. In a Write-back cache, if the victim block is dirty, it should first be copied to the MM. Then Data is written into the Data RAM while Tag (along with Valid-bit == 1) is written into the CAM.
Empty address

• CAM constantly generates Empty information and also an empty address for use by CCU if it is going to bring a new (missing) block into cache and store the corresponding Tag into the CAM.
• If multiple locations of the CAM are empty, the address of the lowest addressed empty location can be produced. This is to be deterministic instead of being random in choosing an empty address as logic to make it random is more expensive.
When you bring a new block you not only deposit the TAG in the CAM but you validate it by setting the Valid bit to 1. If we are always writing a “1” into the valid bit can we avoid stating it explicitly through a pin and save a pin? Well we write a zero in the valid bit if we are flushing out a block for various reasons (1. replacing the dirty block in preparation to bring a new block into cache, 2. in a multicore system to be covered later, if the cache in another core requests the block as he wants to write to it, we need to give away the block and invalidate it in this core’s cache). So we need the input pin for Valid bit.
A CAM for Fully-Associative Mapping only?

• A CAM is certainly used in fully associative mapping (in TLBs, in routers, etc. but not in cache as cache are too big for fully-associative mapping).

• But a CAM can also be used whenever the degree of set associativity is quite high (say 16 or more) where so many shallow TAG RAMs do not make sense. See Q#4.3 from the ee457_MT_Spring2017.
Q#4.3 of ee457_MT_Spring2017

- Processor: 80486 (32-bit address, 32-bit data, byte-addressable processor)
- Cache size: 1KB (256 32-bit words in 4 byte-wide banks each 256x8)
- Block size = 4 words => Block frames = 64
- DoSA (Degree of Set Associativity) = 32
- Number of sets = 2
- This is almost fully associative. 32 segregations (with 32 2-location TAG RAMs and 32 8-location DATA RAMs) does not make sense.
- So we choose to handle with 2 CAMs and 2 DATA RAMs as shown on next page. Each CAM searches the block frames in one set.
4.3 Set-Associative Mapping with DoSA (Degree of Set Associativity) = 32 using CAMs to do TAG search in each set.

# of sets: ______
# of total such CAM+Data RAM combinations: ______
Q#4.3 of ee457_MT_Spring2017

Solution diagram

4.3 Set-Associative Mapping with DoSA (Degree of Set Associativity) = 32 using CAMs to do TAG search in each set.

<table>
<thead>
<tr>
<th>TAG</th>
</tr>
</thead>
</table>

- # of sets: 2
- # of total such CAM+Data RAM combinations: 2
Explanation of the “i” and the “j” in the figure

The “i” (either a “0” or a “1” based on the set #)

If the processor wants to read or write to the cache, the CCU would first ascertain that the block is present in the specific set (\(I = 0\) or \(1\) based on the set bit in the address from the CPU) and then allow the processor to read from or write to the data ram of that set.

The “j” (either a “0” or a “1” based on the set #)

When there is a cache miss, the cache control unit (CCU) will bring the block from MM and will deposit in either set 0 or set 1 based on the set index bit of the address of the block. Within the set, it will choose an empty location if available, or a victim location chosen by LRU or Random replacement algorithm, if it is full.
Q#4.3 of ee457_MT_Spring2017

Solution diagram replicated for the two sets (set 0 and set 1)

Set 0 (i=0, j=0)

Set 1

Set 0 (i=1, j=1)