Cache Review

This lesson is a review of caches. Beginning with the structure of the cache itself, including set associative and direct mapped caches. Then the lesson discusses replacement policies, specifically the LRU policy. The final portion of the lesson covers write policies; write back, write through,write allocate, and no-write allocate.

Locality Principle

Things that happened recently are likely to happen again.
An example of locality is: it rained 3 times today, so it is likely to rain again today.

Memory References

Temporal Locality: If an address has been accessed recently, it will likely be accessed again.
Spatial Locality: If an address has been accessed, it is likely addresses close to it will be accessed.

Locality and Data Accesses video is wrong one - notes when video is correct

Cache Lookups

A cache is a small memory for fast data lookup, to save the time that would be spent going to main memory.
Cache needs to be Fast, so it must be small.
Since it is small, not all the data will fit, so when memory access is required one of two things will happen:

- -a Cache hit - the data from the memory location is in the cache
- -a Cache miss - the data from the memory location is not in the cache

Cache hits return the data fast, cache misses are slow because they have to go to memory. So caches should be designed to have as few misses as possible.

Cache Performance

The properties of a good cache are:

1. A short AMAT

    AMAT = Hit Time + Miss Rate * Miss Penalty

    To achieve a low AMAT:

    A low hit time is required - which means the cache must be small and fast.

    A low miss rate is required - which means the cache must be large or smart.

The miss penalty is usually tens or hundreds of processor cycle times.

In a well designed cache:

hit time < miss time

miss time > miss penalty
hit rate > miss rate
hit rate is almost 1

Cache Size in Real Processors
Modern processors have several caches
L1 (level 1) ::
size = 16k - 64k bytes
hit rate = 90%
hit time = 1 - 3 cycles

Cache Organization
Two basic criteria for a cache are:
Determining a hit and a miss -
hit = the data at the requested memory location is stored in the cache
miss = the data at the requested memory location is NOT stored in the cache
Determining what to kick out of the cache-
the cache is not large enough to hold all the data required by the program, so it must throw out data to make room for new data.

Block Size = the amount of data stored for each memory address. The block size should at least be as large as the single largest access that can be done in the cache. Usually 32 - 128 bytes work well for a block size.

Cache Block Start Address
Where should blocks begin in memory?
Blocks should not overlap - otherwise more than one copy of data will be stored in the cache.
Blocks should start at consistent locations - this reduces cache complexity.
Therefore: blocks should be aligned to match the cache size

Blocks in Cache and Memory
Memory has blocks of data and caches have lines of data. The line size equals the block size.
In 2k byte cache the following block sizes are NOT good:
1 byte -- does not exploit spatial locality and word accesses will need to access multiple lines
48 byte -- this is not a power of 2, which makes the cache more complicated
1k byte - this is too big, only 2 lines will fit in the cache

Choose a block size that:
is a Power of 2
exploits spatial locality
is small enough to have a significant number fit in the cache.

Block Offset and Block Number
The address is divided into

block offset
block number

Cache Tags
Tags determine which block is to be accessed. The tag is the most significant bits of the address.
The tag always:

contains at least one bit from the block number

Valid Bit
The valid bit is stored in the cache, one bit for each cache line. The valid bit stores information about the validity of the data. If valid = 0, the data is not valid and is considered not a hit. If valid = 1, the data is valid and is considered a hit.

Types of Caches

Fully Associative - any block can be placed in any line in the cache
Direct Mapped - a block can only go to a specific line in the cache
Set-Associative Cache - A block can be placed in N number of lines in the cache

Direct Mapped Cache
More than one memory location will map to the same cache line. The least significant bits of the block number are used to determine the cache line, these bits are called the block index. These index bits do *not* need to be stored with the cache line - a cache line whose index is K can only store blocks whose index is K. But we do need to store the remaining bits of the block number - the tag bits.

Upside and Downside of Direct Mapped Caches
Upsides:
Only one line is checked for a hit or miss, so it is fast.
It is also cheaper because only one line is checked.
It is energy efficient.

Downsides:
The block must go in only one line. This can lead to conflicts in the cache, which increases the miss rate.

Set Associative Caches

N-way set associative = the cache is divided into sets of N lines each.

2 Way set associative has sets of 2 lines, N= 2.

Offset, Index, Tag for Set-Associative

The address is divided into:

Offset: which portion of the block is to be accessed, the number of bits used for this are determined by the block size.

Index: which set is to be accessed, determined by the number of sets in the cache

Tag: the unique portion of the address, used to identify the correct address to be accessed.

Fully Associative Cache

Any block can map to any line.

The address is divided into:

Offset: bits used to determine which part of the line is being accessed.

Tag: Used to identify the line to be accessed.

Direct Mapped and Fully Associative

Direct Mapped = 1-way Set associative

Fully Associative = N-way Set Associative

Cache Replacement

When the set is full and there is a miss, a block needs to be kicked out of the cache to make room for the new one.

Methods for choosing which block to kick out of the cache:

Random - pick one at random to replace

FIFO - kick out oldest block

LRU - least recently used, the block that has not been accessed the longest is the one to replace.

NMRU- not most recently used. Only track the most recently used block, then randomly kick out one of the other blocks.

Implementing LRU

LRU exploits locality.

The cache now has the data, tag, valid bit, and LRU bits.

The LRU bits are valued from 0 to N. The LRU block = 0, and the MRU block = N.

When replacing a block, the least recently used block (LRU = 0) is replaced and the LRU is changed to equal the highest value (N). All other LRU bits are decremented by one. This will designate a new LRU block.
Implementing LRU's methods are hardware intensive.

Write Policy

Write allocate - the block is written to the cache
No-write allocate - the block is not written to the cache.
Write-through - the memory is updated when the cache is updated. This is an unpopular method.
Write-back - only write to the memory when the block is replaced in the cache.

Usually write-allocate is paired with write-back.

Write-Back Cache

If a block is modified, it needs to be written to memory at replacement, if not modified, then the block does not need to be written.
A dirty bit is used to track if when the block is written.
Dirty = 0 : the block was not modified
Dirty = 1: the block was modified and needs to be written to memory when replaced.