CSE 30321 – Lecture 17 – In Class Handout

Part A: Example

- In the slides, I noted what would happen if Fetch/Memory took 100 ns
- Let's look at a slightly more optimistic case…

- The CPU has a 1 GHz clock rate
- The L1 cache access time is 1 ns
    - (The L1 cache is a faster level of memory hierarchy)
- 90% of the time we find data in the L1 cache
- The Main Memory access time is 75 ns
    - Thus, if we miss in the L1 cache, we pay a 75 ns penalty
- 1/3 of all instructions are loads and stores
- The base CPI of this machine is 1 (without considering caching)

What is the impact on CPI?

- First, how many instructions reference memory:
    - 1 reference for fetch
    - 0.33 for load/store
    - Thus, there are 1.33 memory references/instruction
- 90% of the time we get an L1 hit – i.e. we find data in L1
- 10% of the time, we have to spend 75 ns
    - 0.1 x 75 x 1.33

Thus, the new CPI is:

= 1 + (0.1 x 75 x 1.33)
= 10.975!

The take away:

- Even with a 90%, 1 CC hit rate, the performance impact can be fairly severe
- We need to be better

Part B: Average Memory Access Time

Therefore, the Average Memory Access Time is given by:

AMAT = Hit Time + (1 – Hit Time) x Miss Penalty

In the previous example: 1 + (1-0.9) x 75 ns → 1 ns + 7.5 ns → 8.5 CCs

Part C: Caches and their structures

Cache Blocks:

- As mentioned, a "block" is the smallest amount of "stuff" (data) that can be brought into a cache
  - Generally blocks are between 16-128 bytes of data
- Question:
  - In MIPS, datawords are just 4 bytes of data.
  - Why bring in 16-128 bytes of data?
- Answer:
  - Locality
  - (In other words, the idea is that because we referenced a particular data word or instruction encoding, we'll probably reference other stuff by that same data/instruction soon … so just bring it closer to the datapath right away.)
- Therefore, a cache organization might look something like this:

…..

Block 0	Word 0	…	Word N
Block 1	Word 0	…	Word N
Block 2	Word 0	…	Word N
Block 3	Word 0	…	Word N
Block 4	Word 0	…	Word N

…..

Part D: Where does a block go in the cache?

- If a cache is an array of blocks, how do we choose where a block goes?
  - There are 3 ways to decide

1. Direct Mapping

- As an example, let's say that we have 8 blocks in our cache and the address that we want to load data from is 12.
- We can use the mod function to select where this block goes.
  - E.g. 12 % 8 = Block 4
- Similarly
  - 120 % 8 = Block 0
- What if we get the sequence of memory addresses: 12, 20, 12, 20, 12, 20, 12, 20 …
  - Both map to Block 4!
  - We have to replace a block with each reference
    - (And with this sequence, we would never find the data in the cache)

2. Fully Associative Mapping

- If we have a cache with 8 blocks, the block can go *anywhere*
  - E.g. it could be placed at Block 0, 1, 2, 3, 4, 5, 6, or 7
- The net effect:
  - We could potentially eliminate conflicts like you just saw above

  - However, the search time will realistically increase significantly

3. **Set Associative Mapping**

- This involves different *sets* of *blocks*
- See the picture below:

Location	Data	**Set**
0		**0**
1		
2		**1**
3		
4		**2**
5		
6		**3**
7		

- The basic idea is that a block maps to a set – and then can be placed anywhere within that set.
  - Thus, you get some of the speed of a direct mapped cache (e.g. its easier to find where a block maps too), but could eliminate some of the conflicts associated with a direct mapped cache.
- Thus, if we have a request for the data at address 12, we would do a mod function with the ***number of sets***
  - E.g. 12 % 4 = Set 0
  - The block could then be placed anywhere within Set 0
    - E.g. at Location 0 or Location 1

Part E: How do you find a block?

- The previous discussion focused on how where you place a block in a cache.
- Another question to consider is how you find data associated with a given block.

As an example, let's assume that we have the instruction: lw $5, 0($2)

- How do we find the data associated with "0($2)" in a cache?
- Well, in MIPS, we use 0($2) to calculate a 32-bit physical address
- We'll start with that – and divide the physical address up into 3 different fields
  - Note that the procedure to be discussed applies even if the address is not 32 bits; we could just as easily discuss an N-bit physical address.

Bit 31		Bit 0
Tag	Index	Offset

A very important thing to understand: How to use/interpret each field!

Let's start with the Index:

- The index bits are used to pick which block (for a direct mapped cache) or which set (for a set-associative cache) an address will map to

00	Block / Set 0
01	Block / Set 1
10	Block / Set 2
11	Block / Set 3

- For example, if there are 2 index bits, then there are 4 blocks or 4 sets in the cache that a physical address may map to

- Another example:
  - If a cache has 1024 blocks in it, how many bits of index are needed to address each block?
    - $2^{10} = 1024$; therefore 10 bits
  - If a cache has 1024 blocks in it and is *8-way set associative*, how many bits of index are needed?
    - Note that 8-way set associative means that there are 8 blocks associated with a given set
    - However, note that the question explicitly states that there are only 1024 TOTAL blocks in the cache
    - 2^{10} blocks / 2^3 blocks / set = 2^7 sets. Therefore 7 bits of index are needed.

<u>Let's look at the offset next:</u>

- The offset bits are used to find the right word in a block
  - Remember, even though an instruction encoding or data word may be 4 bytes long, blocks usually contain anywhere from 16-128 bytes!
- The number of bits that comprise the offset depends on:
  - If there is more than 1 word / block
  - To what level a word can be addressed
    - Remember, MIPS is byte addressable
- Example:
  - If data is addressed to the word:
    - If there is just 1 word / block, 0 bits of offset are needed
    - If there are 2 words / block, 1 bit of offset is needed
    - If there are 4 words / block, 2 bits of offset are needed
    - If there are 8 words / block, 3 bits of offset are needed
    - Etc., etc.
- But what if there are 2 words per block and data is byte addressable?

Word 1				**Word 2**			
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte

- If each byte can be addressed, how many bits of offset are needed?
  - Answer: 3
    - There are 8 byte and $8 = 2^3$ … so 3 bits are needed.

<u>The remaining bits form the tag:</u>

- The tag helps us to ensure that we're looking at the right entry.

Note that:

- **The least significant bits of the physical address form the offset**
- **The next N bits of the physical address form the index**
- **The last / most significant bits of the physical address form the tag**

Part F: Example

- Assume we have lw $8, 0($2)
  - 0($2) turns out to be physical address: AA BB CC DD (in hex)
- The first place we would look to find the data associated with address AA BB CC DD is in the cache
- Let's assume our cache is:
  - Direct mapped
  - There are 16 words / block
  - Data is addressed to the word
  - There are 4096 blocks

How many bits of offset are needed?

- 4. $2^4 = 16$.
  - We need to pick one of the 16 words in a block

How many bits of index are needed?

- We need to be able to select 1 of 4096 blocks
- $2^{12} = 4096$
- Therefore 12 bits of index are needed.

The rest of the bits form the tag.

- Therefore there are 32 - 4 - 12 = 16 bits of tag

For this physical address we would have:

Tag	Index	Offset
AA BB	CC D	D

CCD = 1100 * 1100 * 1101 = 3277_{10} th entry (or block)
D = 1101 = 13th word in that block