3D2 Microprocessor Systems II – Lab 8 – David O’Leary

1. **Briefly describe why software developers should care about caches.**

Software developers should care about caches for a few reasons. Firstly, they can greatly reduce the power consumption usually needed to keep data close to the CPU. They also ensure that the instructions and data that a processor requires are as near as possible to the processor to enhance speed performance. Not using caches will greatly increase the time needed for a program to run.

1. **Briefly describe the difference between a write-through and a write-back cache.**

In a write-through cache, the write is done to the cache and the backing store synchronously, whereas in a write-back cache, writing is only done to the cache and an edited cache block is written to the backing store just before it is replaced. Due to this, write-through is slower in writing data, since it has to be written to two places, than in write-back. The main con of write-back is that if a power failure occurs before data is written to storage, the data can be lost.

1. **Describe and contrast the differences between *Full-Associative* / *Direct-Mapped* and *N-Way Set-Associative* cache organizations.**

In Full-Associative cache organisations there is only one set in the cache, whereas in N-way Set-Associative cache organisations and Direct-Mapped cache organisations there can be one or more sets in the cache.

In Full-Associative cache organisations each main memory location can be in any entry in the cache, whereas in Direct-Mapped cache organisations each location can only go in one entry in the cache and in N-way Set-Associative cache organisations, each main memory location can go in one of N locations in the cache.

Full-Associative caches are essentially the opposite of Direct-Mapped caches. Full-associative caches require the most System-on-chip (SOC) area in order to be implemented, whilst Direct-Mapped caches require the least. N-way Set-Associative caches are, in a sense, midway between the other two types in terms of both performance and SOC area implementation.

1. **Caches are an important and scarce resource – describe how they are managed for maximum benefit.**

Since caches fill up over time despite new references still needing to be looked after, cache entries are emptied over time. To accommodate for this, a cache content replacement/eviction policy and a policy to handle misses are put into place. A cache miss is when references are either never stored in the cache, stored and then evicted. In the case of a multiprocessor system, coherence-handling policies are also required.

A cache replacement/eviction policy is based on either spatial or temporal locality, i.e. working under the assumption that applications either execute code sequentially in memory and thus pre-fetching data during a cache miss, or, working under the assumption most apps will repeatedly access the same blocks of code (loops etc.) and thus keeping the most recently accessed data in the cache.

Cache miss policies are either write-back or write-through, respectively meaning that writes are only sent to memory when cache is evicted or that all copies of data in the cache are uploaded immediately.

These policies help to improve the efficiency of caching as much as possible while still storing important data.

1. **Using the same example configuration that is given in W09L04, write a short for-loop in ARM assembly (that is different from the example given in the lecture notes) and illustrate how long it might take to execute the code:**

* **In the presence of an initially cold cache (no instructions/data are cached)**
* **In the presence of an initially warm cache (all instructions/data are cached)**
* **With the cache entirely disabled**

For loop:

movs r0, #3

loop ldr r2, [r1]

add r2, #3

str r2, [r1]

subs r0, #1

bne loop

* Since un-cached instructions normally take 10ns to run, with LDR (if un-cached) and STR instructions taking an extra 10ns, the length of time taken to execute this code would be 10+20+10+20+10+10+1+1+11+1+1+1+1+11+1+1= 100 ns in an initially cold cache.
* In an initially warm cache, this would be much faster, with each instruction taking 1ns and LDR (if loaded from un-cached) and STR still taking an additional 10ns. It would take 1+3\*(1+1+11+1+1) = 46ns
* With the cache entirely disabled, the code would take much longer, taking a total of 10+3\*(20 + 10 + 20 + 10 + 10) = 220 ns