MICROPROCESSOR SYSTEMS

LAB #08

Ujjayant Kadian

22330954

Questions

Q1. Briefly describe why software developers should care about caches.

Ans.

Software Developers should care about caches as it can significantly improve the performance of the code. Since, cache memory is located near to the processor, they are fast though small and store the frequently accessed instructions or data. Thus, the processor can access these pieces of information faster than accessing it from the main memory which is slower. Using this concept, software developers can write their code in such a way that it leads to faster execution and better resource utilization. This can also be seen from Lab#07, where calculation of PI using Wallis’s product was much faster when the code used the cache.

Q2. Briefly describe the difference between a write-through and a write-back cache.

Ans.

The difference between a write-through and a write-back cache is as follows:

|  |  |
| --- | --- |
| Write-through | Write-back |
| In write-through caching, data is written to both the cache and the main memory simultaneously. This ensures that the data stored in cache and main memory are consistent. However, since the data is written simultaneously in both the memories the performance is slow. | In write-back caching, data is written to the cache memory only at first and then written to main memory if it is required by it or cache memory requires more space. Thus, the performance is faster than write-through caching. However, it is difficult to maintain consistency in data in both these memories and requires complex recording mechanism. Inconsistencies may arise due to system failure or power outrage. |

Q3. Describe and contrast the differences between Full-Associative / Direct-Mapped and N-Way Set-Associative cache organizations.

Ans.

Full Associative Cache Organization:

In this organization, every block of the main memory can be mapped to **any** of the smallest portion of data that can be stored into a cache (called cache line). This mapping is done by comparing tag bits of main memory and cache lines and then mapped when a match between these bits is found. One cache line stores one block of the main memory. Thus, there would not be a case of cache conflict (where two blocks of main memory map to the same cache line). However, this organisation requires complex hardware to fulfil this and hence, is expensive.

Direct-Mapped Cache Organization:

In this method each block of the main memory is mapped to **only one** cache line. Thus, like full associative cache organization, one cache line stores one block of the main memory but that cache line is fixed unlike in Full Associative organization. To map memory blocks to cache line, the memory address is divided into three parts – tag bits, index bits and offset bits. Tag bits are used to uniquely identify the memory block, index bit is used to determine to which cache line to block should go to and offset bits are used to determine the location of data block within the cache line. Since, the mapping mechanism is not that difficult to implement, the hardware is not that complex, and cases of cache conflicts can occur.

N-Way Set Associative Cache Organization:

This organization is the combination of Direct-Mapped and Full Associated cache organizations. In this mechanism, the cache memory is divided into **different sets and each set contain multiple cache lines**. Thus, each set can store many main memory blocks and each block is mapped to **any of the cache line within a set**. The index bits of the memory address are used to translate blocks to a set of cache lines. Tag bits are used to uniquely identify a memory block and offset bits is used to locate the position of data within the cache line. The N in this organization denotes the number of cache lines that can exist in a set. Thus, this organization reduces the possibility of cache conflicts (as N increase, probability of having cache conflict decreases) and reduces the complexity of the hardware.

Q4. Caches are an important and scarce resource – describe how they are managed for maximum benefit.

Ans.

Management of Cache for maximum benefit:

1. Location and Size of cache memory: Having a larger cache can hold more data and thus reducing the number of cache misses (when the processor requests to retrieve data from cache but that data is not available in cache). Having a cache closer to the processor reduces the access time.
2. Replacement Policies: If a cache miss occurs, there should be some mechanism which determines which existing data to replace. Some of these mechanisms include Least Recently Used (LRU) – it replaces the data/cache line, which was least recently used, Least Frequently Used (LFU) – it replace the cache line which was used the least.
3. Concept of Pre-Fetching: It predicts the data which is most likely to be called by the processor and thus loads it into cache. It reduces the access time and cache misses.
4. Write Polices: Cache can opt for different write policies as explained in Q2.
5. Coherency: It is important for the cache to have coherent/consistent copies of shared data. There are several coherency policies that helps to maintain such consistency by invalidating or updating copies of data in the caches whenever a write occurs. Two most common such mechanisms are –

Bus Snooping: The cache controllers watch the main address bus to see if there are any accesses to areas of memory that are presently being cached.

Directory Based Protocols: Protocols for maintaining the consistency between all the caches that use a common directory to store the shared data for caching.

Q5. Using the same example configuration that is given in W09L01, 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

Ans.

For Loop in ARM that multiplies the number stored in r1 by 2, 4 times:

loop:

MOV r0, #0

MOV r3, #2

loop\_start:

LDR r2, [r1]

CMP r0, #4

BEQ loop\_end

MUL r2, r2, r3

STR r2, [r1]

ADD r0, r0, #1

B loop\_start

loop\_end:

Time taken in different cases:

1. In the presence of an initially cold cache (no instructions/data are cached):

As cache is initially cold, time taken would be as follows:

10ns + 10ns (first two mov instructions)

+

10ns (First load instruction)

+

10ns (CMP instruction)

+

10ns (MUL instruction)

+

10ns (STR instruction)

+

10ns (ADD instruction)

+

10ns (B instruction)

`NOW: all the instructions are cached, therefore-`

+

4 \* 3 \* 1ns = 12ns (there are 4 instructions in every iteration – except LDR and STR instructions and the loop will go on three more times)

Total Time Taken = 92ns

1. In the presence of an initially warm cache (all instructions/data are cached):

As cache is initially warm (all instructions are cached), time taken would be as follows:

1ns + 1ns (first two mov instructions)

+

0ns (First load instruction)

+

1ns (CMP instruction)

+

1ns (MUL instruction)

+

0ns (STR instruction)

+

1ns (ADD instruction)

+

1ns (B instruction)

+

4 \* 3 \* 1ns = 12ns (there are 4 instructions in every iteration – except LDR and STR instructions and the loop will go on three more times)

Total Time Taken = 18ns

1. With the cache entirely disabled:

10ns + 10ns (first two mov instructions)

+

10ns (First load instruction)

+

10ns (CMP instruction)

+

10ns (MUL instruction)

+

10ns (STR instruction)

+

10ns (ADD instruction)

+

10ns (B instruction)

+

(4 + 2) \* 3 \* 10ns = 180ns (there are 4 instructions in every iteration + one LDR and one STR instructions and the loop will go on three more times)

Total Time Taken = 260ns

NOTE:

This is under the assumption that –

• A cached instruction typically executes in one ns, whereas an uncached instruction typically requires ten nanoseconds.

• If the instruction is a memory load (LDR) instruction, it will require an additional 10 ns if the location from which it is to be loaded is not already cached, but not otherwise.

• If the instruction is a memory store (STR) instruction, write-through caching is allowed, so it always takes an additional 10 ns to write the data to the location.