**Cache Controller Design**

**Daniel Komac**

**CPE 186 Spring 2016**

**Professor Vadhva**

**April 14, 2016**

**Table of Contents:**

1. Main Memory
2. Cache
3. Cache sets 2K bytes each
4. 4 way set associative
5. Block Size 64LW(526bytes)
6. 4 Lines per block
7. Line size 16LW(64 bytes)
8. LRU policy
9. 8-bit and 16-bit non-cacheable
10. I/O transfers are non-cacheable
11. Read cache miss LW acces
    1. LW 0 – LW12
       1. Transfer will start from the address accessed till the end of the line boundary and then wrap around to the address
       2. The first data is transferred to the CPU at the same time to the cache.
    2. LW 13 – 14 non-cacheable, however the requested data is transferred to the CPU
    3. LW 15
       1. Non-cacheable, data transferred to the CPU
       2. After CPU receives this LW, entire next consecutive line is transferred to the cache.
12. Cache Write
    1. Cache Write miss is non-cacheable
    2. Cache write hit- write back policy
13. Bus Snooping circuits – MESI protocol
14. Memory pages 100 to 200 are non-cacheable
15. Flush circuitry to invalidate the entire Cache
16. **Main Memory**

The Main memory is the area in which all data is being held. Our permitted memory size is 4 Mbytes which in Long word translate to 1 M LW (Mega Long Word). Now we can splice this memory into 2000 pages which is considering the caches. Now this memory is 4 way set associative so 4 sets must be incorporated which means each set will handle 500bytes in this page. That means that there are 4 rows, each a set with 500 bytes of information.



Main memory is also connected to the processors, four processors total, each with an equivalent cache size of 2K bytes(512 LW)



**2. Cache Size**

The cache is designed in the same manner, several pages per cache this is approximately 8 Kbytes of data. Each cache is designed to be 2 Kbytes of data because of 4 way set associative rule. The pages of caches hold 64 LW (256 bytes) of info. The cache is grouped in with processor and is used to make memory access much more available.

Both connected to each other to allow quick transfer of data.

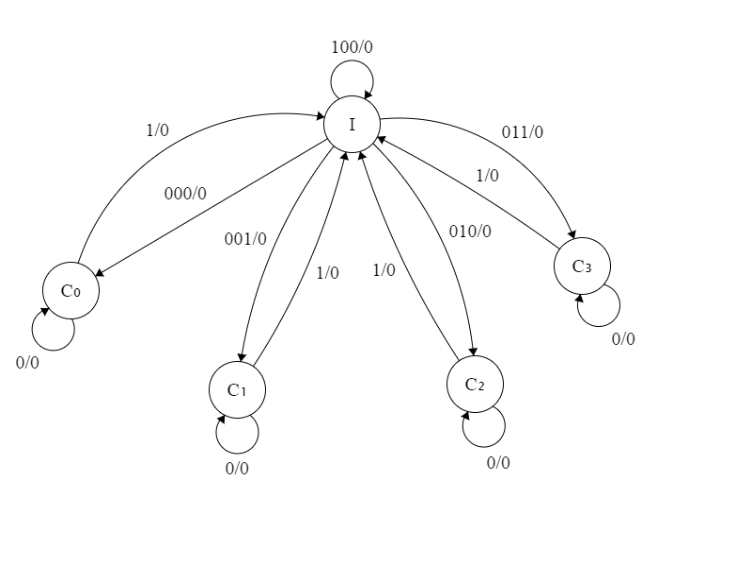


Cache is able to be very useful at the expense of being expensive. Cache is also granted according to who needs access to the bus.

**3. 4 Way set Associative:**

Four way set association is the process in which the memory of the cache is split amongst four specified sets. These sets will give the cache the ability to take a fourth of the page and assign each individual cache a segment. The memory segments are split into 4 lines, the Block size correlates to how each cache will work. These splits allow for the separate storage of memory but storing memory like this requires a comparator which will check each set to make sure there aren’t duplicate information.



The cache being split into different segments allows for separation of materials and avoiding redundant information in specific sets. Having this set up uses more resources but allows for an efficient means of storing info. The cache line that the bus/ cache controller uses contains a tag along with a verification giving the bus the access to the specified set which would be tagged. The tag allows a value to be hold of the specific set allowing for read write capabilities and other capabilities.

**4. 2Kbytes Set Cache**

Each set being 2k allows for Set cache to operate as an entire entity of cache. The total Cache size is 8Kbytes which means that all 4 Caches can split this page of Cache equally and retain all the same capabilities. When cache is segmented into sets there are additional resources needed and precautions taken.

Each cache is split into this amount of memory because it is one fourth of the cache, being able to split data into smaller pages, a 4 way comparator is still needed and the more caches we can implement the closer the system will become to fully associative.

Each 2k slice of Cache will contain a unique tag so they can be uniquely addressed and they will have the same amount of memory as other caches as so the memory can readily be used for easy transactions from cache to cache, or sharing of data between any other processes that need access to the cache.

The reasoning behind split memory is for easier management at the expense of more materials, the lookup penalties do increase simply because there are more entries that have to be managed in order to assure that the right information is being attained. 

Allowing for unique memory addresses gives more secure uses of holding data. Especially since cache is rapidly updated over and over as processes happen.

**5. Block Size**

The block size for the cache will be 64 LW which is roughly 256 bytes. The long word consists of 32 bits and uses all available bits for information relevant for the cache being operated, I.E. tags, data, and relative address. In this cache controller it will need a set field, 2 bits, an address, and data. All able to be organized with in the lines of each section of the cache.



**6. 4 Lines per Block:**

Within each block of memory that is being held in the cache it must be segmented into 4 lines for 4 way set associativity. We do this by assigning special bit amounts to each unique line. Also needing comparators to check to make sure there are no information that is being shared or is going to throw an invalid flag within the MESI protocol.

This allows us to identify chucks with bit sets, as labeled on the side of each line. These lines will contain unique information and allow for quick transportation between data at the cost of some extra materials. It is also worth noting that the more associativity cache blocks gain the closer it becomes to fully associative.

**7. Line Size 16 LW:**

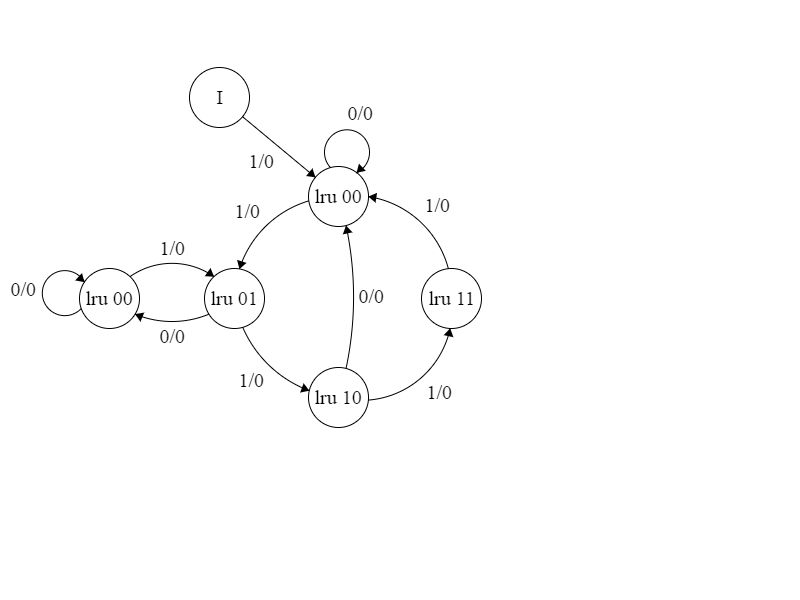
Each address line within the line will hold 4 bytes which is equivalent to 16 bits. All the information will be stored with a generic address, i.e. line address, set number or unique id, and data. These are usually held in the order of the upper bits being the address while the lower bits hold the data. This means that we’ll be able to hold information in a four way associative cache diagram.

With this structure we are able to hold memory in a readily available fashion with the assignment of very simple tags using the same address.

**8. Cache Replacement Policy – LRU**

This replacement policy allows for the cache to prioritize memory in order of importance this can be easy using our tag window in our line to determine the priority of the information in the cache. We may set the tag to 00 when initialized and then increment anytime a variable is added when it’s not in the, any value that is reused has its LRU value changed to 00 as most recent. If the LRU is swapped constantly between the two most common variables, do not increment any other caches, except the most recent ones being used, I.E 00 and 01 would swap variables while all others stay the same. Along with that there needs to be an exception for caches containing variables of type 11, since they are least frequently used the cache is allowed to erase any data with more relevant data that is important to more recent processes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| X\_2 | X\_1 | X\_0 | I\_0 |  | X\_2n | X\_1n | X\_0n |
| 0 | 0 | 0 | 0 |  | X | X | X |
| 0 | 0 | 0 | 1 |  | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |  | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |  | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |  | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |  | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |  | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |  | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |  | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |  | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |  | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |  |  | | |
| 1 | 1 | 0 | 0 |  |
| 1 | 1 | 0 | 1 |  |
| 1 | 1 | 1 | 0 |  |
| 1 | 1 | 1 | 1 |  |



**9. 8 and 16 bit requests are non-cacheable:**

Since 8 and 16 bit requests are non-cacheable they will be written in the LW 13-14 of every block. Allowing the instruction to not be cached as it would cause an error in storage but allowing for the request to still be processed as it will be sent to the CPU directly instead of both the cache and the processor and being held by the cache.



**10. I/O transfers are non-cacheable:**

For this we use the same methodology of implementing non-cacheable information into one of the subsequent long words into the cache. Rather than being cached it will be processed directly into the CPU also relaying any information needed by the cache to successfully output/execute what the I/O is asking of the device. This is achievable by using cache longword 15 which is un-cacheable and transfers subsequent lines of data from the cache to the CPU.

**11. Read Cache miss Long Word access:**

When the memory that is trying to be located within the cache is not found at the declared location it will continue to cycle through all cacheable entries looking for the correct data within the cache and wrapping around back to the origin if it is not found. Misses will happen when data is not found or when trying to cache a non-cacheable operation. For our first 13 long words will be allowed to be readable but the next 3 lines are non-cacheable and return data or transfer the data trying to be obtained directly to the processorAny cache read/write miss will cause a lot of time to be passed. The parameters being passed will need to specify an address that does not exist or give information to a specific long word in the block that is non-cacheable to be used instead of a regular cacheable .

If there are specific requests for non-cacheable segments that allow for data transfer then that will be defined by the long word being defined. Long word 15 will return data of the next consecutive line after being called.

**12. Cache Write:**

Cache write miss will not be stored into the cache and will translate over to the processor where the information will be used or abandoned. The information will stay out of main memory but may cause some errors. Cache hit will result in the MESI protocol being triggered and cause a modified bit flag to occur which means main memory and the cache both share the information. This information only stays in the cache until instructed to move back to main memory after processing. MESI protocol will maintain order amongst the caches during write hit/miss. Using this write back method will use less bandwidth and everything occurs at the speed of the cache.



**13. MESI Protocol:**

This will enable the caches to talk amongst themselves along the bus using snooping circuitry which allows for them to share information and validate any information going to or from the caches. This snooping will ensure that there are no duplicate strands of information or to ensure main memory does not get overridden by invalid data. This protocol will also give access to a safer system with onboard comparators.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X\_1 | X\_0 | I\_1 | I\_0 |  | X\_n1 | X\_n0 |
| 0 | 0 | 0 | 0 |  | 1 | 1 |
| 0 | 0 | 0 | 1 |  | 1 | 0 |
| 0 | 0 | 1 | 0 |  | 0 | 0 |
| 0 | 0 | 1 | 1 |  | X | X |
| 0 | 1 | 0 | 0 |  | 1 | 1 |
| 0 | 1 | 0 | 1 |  | 1 | 0 |
| 0 | 1 | 1 | 0 |  | 0 | 1 |
| 0 | 1 | 1 | 1 |  | 0 | 0 |
| 1 | 0 | 0 | 0 |  | 1 | 1 |
| 1 | 0 | 0 | 1 |  | 0 | 0 |
| 1 | 0 | 1 | 0 |  | 1 | 0 |
| 1 | 0 | 1 | 1 |  | X | X |
| 1 | 1 | 0 | 0 |  | 1 | 0 |
| 1 | 1 | 0 | 1 |  | 0 | 1 |
| 1 | 1 | 1 | 0 |  | 0 | 0 |
| 1 | 1 | 1 | 1 |  | X | X |

**14. Pages 100-200 un-cacheable:**

These pages cannot be used for cache and are assumed to be something more important and have a higher priority than any other page in memory, therefore caches will not be able to attain these pages and are not permitted to change any information from these pages.

Doing this wil allow for main memory to have dedicated pages to valuable information that should not be accessed by any process that the processor will be executed as to not compromise the rest of the programs that are running or any other executions that might consider these to be very essential to the well-being of the structure.

**15. Flush circuitry to invalidate the entire Cache:**

A circuit will allow for the flushing of entire cache blocks when declared, this can be obtained with a process that resets all bits in a block with irrelevant information such as all 0’s as to restart the cache and obtain a clean slate for caches. This can be done at the end of all uses of the cache or at start up to ensure that everything is clean before any other operations or executions are made. This allows for the invalidation of any selected cache or other information that needs to terminate execution or anything else.

