Cache Controller Implementation with Write-Through Policy

Objectives:

The objective of this project is to implement a simple cache system and integrate it with the RISC-V processor you previously implemented.

Introduction:

In this project, we will work on implementing a simple caching system for the RISC-V processor. For simplicity, we will integrate the caching system with the **single-cycle implementation**. Additionally, we assume the following:

- Only data memory will be cached. The instruction memory will not be affected.
- We will have only one level of caching.
- The main memory module is assumed to have a capacity of 4 Kbytes (word addressable using 10 bits or byte addressable using 12 bits)
- Main memory access (for read or write) takes 4 clock cycles
- The data cache geometry is (512, 16, 1). This means that the total cache capacity is 512 bytes, that each cache block is 16 bytes (implying that the cache has 32 blocks in total), and that the cache uses direct mapping.
- The cache uses write-through and write-around policies for write hit and write miss handling and no write buffers exist. This implies that all SW instructions need to stall the processor.
- LW instructions will only stall the processor in case of a miss.

In order to build the caching system, we are going to replace the data memory in the single cycle implementation with a data memory system module which includes a cache memory module, a cache controller module, and the data memory module. The new memory system module is shown in Figure 1. The memory system has an extra control signal called stall. The stall signal is asserted when the memory system needs to temporarily stop the processor. The stall signal will remain asserted until the processor is allowed to resume normal execution.

The cache controller encapsulates the array of tags and valid bits and uses the index and tag parts of the requested memory address to decide whether there is a hit or a miss. It is also responsible for generating the stall control signal in addition to controlling both the cache module and the memory module as explained in the 4 scenarios below:

- The processor requests a read operation (executing a LW instruction) and the cache controller decides that it is a hit. In this case, there is no stall necessary and the data is read from the cache module.
- The processor requests a read operation (again executing a LW instruction) and the cache controller decides that it is a miss. In this case, the stall signal is asserted and the data is read from the data memory module which provides 1 block (16 bytes or 128 bits) of data to the data cache. When this data is available, the data memory module asserts a ready

signal that the cache controller uses to ask the data cache to fill the corresponding block with the data coming from the memory and to deassert the stall signal.

- The processor requests a write operation (executing a SW instruction) and the cache controller decides that it is a hit. In this case, the word to be stored has to be written both in the cache memory and in the data memory (due to the write-through policy). So the cache controller asserts the stall signal until the memory confirms that it finished writing via its ready signal. Simultaneously the cache controller asks the cache memory to update the value.
- The processor requests a write operation (again executing a SW instruction) and the cache controller decides that it is a miss. In this case, the word to be stored is written in the data memory only (due to the write-around policy); however, in this case too, the cache controller asserts the stall signal until the memory finishes the storing.

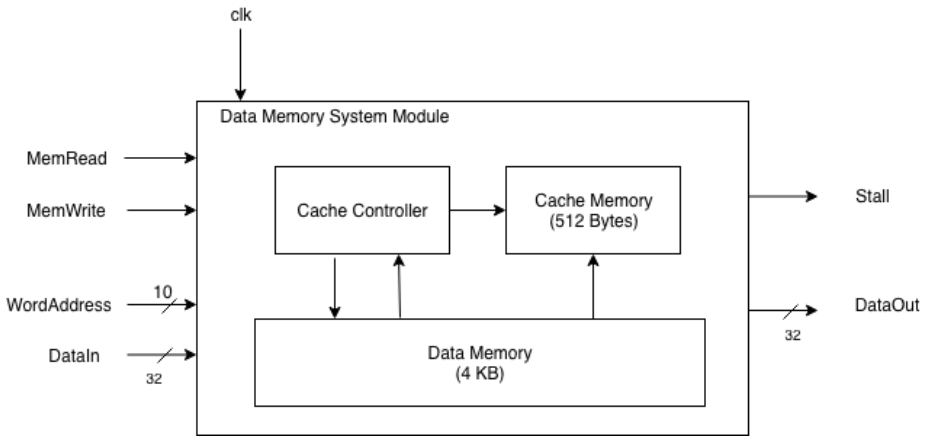

Figure 1 Data Memory System

Note that in order to decide whether the access is a hit or a miss, the cache controller has to be provided with the index and the tag of the address being accessed and it also needs to have access to the array of tags and the array of valid bits corresponding to the cache blocks in the cache module.

The **`valid_bits`** array needs to be initialized to zeros to indicate a cold cache start. This can be done using reset signal. Both arrays (valid and tag) need to be updated when a new block is cached (at the rising edge when the fill signal is asserted).

To function properly, the cache controller needs to implement a finite state machine working on the opposite edge of the clock from the PC. The finite state machine can have 3 states: idle, reading, and writing. Initially the controller is in Idle state and it moves to either the reading or the writing state if there is a read miss or any kind of write respectively. Once the corresponding operation is finished, the controller reverts back to its original idle state. These states will help you assert the stall signal at the right timing.

Deliverables

- Integrated code implementing the above system.
- Testbench running different cases of LW/SW programs
- A drawing of the finite state machine implemented.

Appendix

Caches are used to speed up data fetching from main memory or higher level memory. Because they are smaller in size they can fit very close to CPU, however they can't accommodate all the data in the main memory. So extra bits are added beside data in the cache to help mapping the main memory large address space.

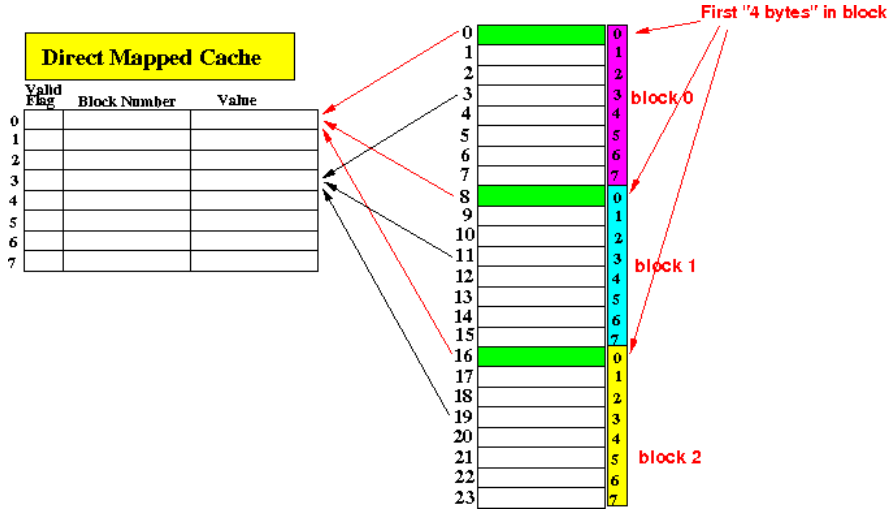

Figure 1 Direct Mapped Cache

Cache controller maps data block (line) from memory to cache. There are several ways to do the mapping, we will use direct mapping. Direct mapping indicates that for every block(line) in the main memory, there is only one place for it in the cache as seen in fig. 1.

So how the cache actually works? When a data requested from the cache, it checks the valid bit, if valid, then it retrieves the data from cache, else it has to raise a hit miss and start loading the missed cache block from the main memory. The rest of the processor should be waiting till the cache block is transferred from main memory to cache.

Let's take an example, assume we have organization like the drawing in fig. 1 where the block is 32 bit (one word) and we have 8 blocks in the cache (word addressable). Initially, cache is empty so all valid bits are zero. If we are trying to read from address 11 for the first time. The cache would see that it doesn't have the value and it must raise an interrupt. This interruption should stop incrementing pc and other processor related activities till the memory is loaded.

The cache controller will then copy the content of the block starting at 11 into block 3 into the cache. The controller uses the address bits to determine the correct block. This is done by splitting the address into three components {Tag, index, offset}. The index indicates which block in the cache memory we are

trying to access, while the offset indicates which word inside the cache we are trying to access. In case of invalid bit, then we have a cache miss. If the valid bit =1 then we must make sure that the address tag and the block tag are the same. If not, then we have a miss and we must copy the data from the main memory to the cache.

To understand more about caching you can check MIT course module here or Emory module.