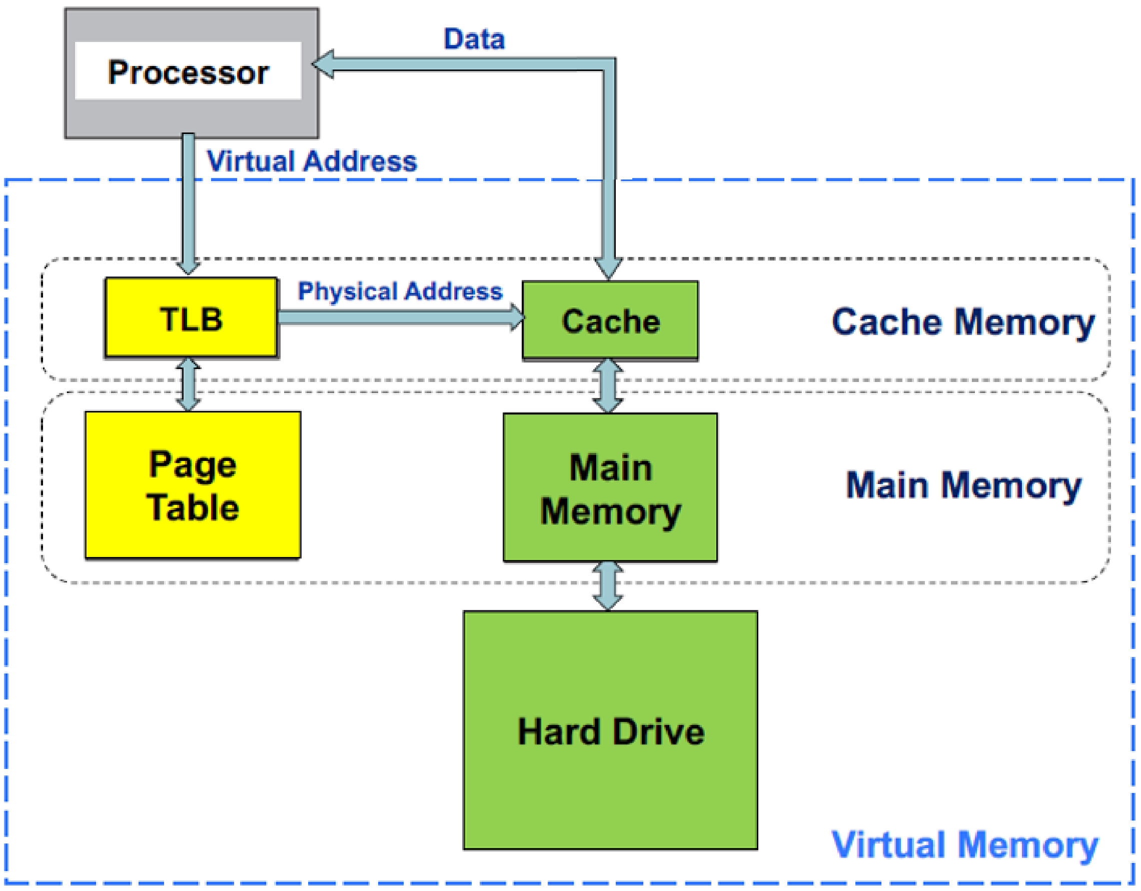
Ve370 Project 3

Project Description

Cache memory is the top of the memory hierarchy that interacts with the CPU directly. Communications
between CPU and the rest of the memory hierarchy are typically through cache. Simplified interfaces
between the CPU and cache and between cache and the main memory are shown in the following structural diagram.



Assume the following properties of the memory:

- Byte addressable
- Size of the main memory: 1024 bytes
- Size of the cache: 64 bytes
- Size of a block: 2 words
- **Size of a page: 2 blocks**
- Cache associativity: 2-way associative
- Write technique: write back
- Cache replacement policy: Least Recently Used (LRU)
- Size of TLB: 32 bytes
- TLB associativity: full associative
- TLB Write technique: write back

- TLB replacement policy: LRU
- **Virtual address: 12 bits**
- **Physical address: 10 bits**

When CPU needs to access a data/instruction in the memory, it sends a virtual address to the **TLB**.

If there is a hit in TLB, TLB then sends the corresponding physical memory address to the cache memory. If there is a TLB miss, TLB then should then get the physical address from the page table, then send the corresponding physical address to the cache memory.

If there is a hit in cache memory, then CPU reads/writes data directly from the cache memory. Else, cache goes to the main memory for the missing block, by sending the address to main memory, then CPU should resend the same address to try again.

For simplicity purpose, assume the main memory always has hit and latency of the main memory access is not considered. You can also design the page table as you want, as long as the value of virtual page address is not equivalent to the physical page address. For example, you can design the page table that 0x000 is mapped into 0x001, and 0x001 is mapped into 0x000.

You can also design the page table and testcase so that there are no page faults.

Model the cache memory and main memory in Verilog HDL. Write a testbench to act like a CPU to provide a sequence of addresses for reading or writing. Pre-load the main memory with randomly generated data by your team. Simulate the functions of the memory hierarchy with an appropriate Verilog simulator.

What you need to implement:

1. A "processor" module that gives out virtual addresses
2. TLB
3. Cache
4. Page Table
5. Main Memory

Notes:

1. Clock signal is not needed.
2. You do not need to implement a FSM.
3. You are free to add some new features, say, using a buffer, or adding a new signal, if you think it is necessary.
4. It is likely that we will just use the provided test case.
5. To show the change of hit_miss, you may add a latency to get data from main memory and write it to the cache.
6. For output of main memory, only showing the part whose value has been changed during the simulation process is enough.
7. You are free to create your own design, but it should be reasonable, and you need to give a clear explanation to us during demonstration.

DELIVERABLES

Written report is OPTIONAL for this project. The entire project including all Verilog modules must be
submitted in a zipped folder and clearly indicate which Verilog simulator has been used. Simulations must be separated and clearly indicated. All Verilog modules must be clearly commented to receive full marks. Screen shots and explanations of simulation results can be submitted in a PDF file if you think it will be helpful.
This is a group assignment. One submission for each group is needed. Your work must be submitted
electronically to Canvas before the specified due date.

GRADING POLICY

Correctness, completeness, clarity of the program: 80%
Source code: 20%
Late submission will result in 0 point for the source code part and deduction of 5% per day in the other part until all 100% is deducted.

Due data

The due date for submission on Canvas is 2nd Aug 11:59 pm.