Part 3 - Instruction Cache and Memory [25 marks]

Once you have the data cache and data memory working properly, use the same concepts to add an instruction cache and an instruction memory to your system. Note that you will not be using the hardcoded instruction array in your test bench (with #2 time units artificial reading latency) anymore.

Following are the parameters to be used when designing the instruction cache:

- Instruction ***word-size*** is 4Bytes (32 bits), as defined by the ISA.
- ***Cache size*** is 128Bytes
- Use a ***block-size*** of 16 Bytes (∴ cache can hold eight instruction blocks)
- Use ***direct-mapped*** block placement
- You need to store the corresponding ***address tag*** along with every instruction block
- You need to store a ***valid bit*** with every instruction block. At the beginning, all cache entries are empty, therefore invalid.
- Dirty bit and write policies are not needed, as the CPU does not write to the instruction memory. Evicted blocks can simply be discarded.

Your instruction words are 4 Bytes wide. The instruction memory should be able to hold 256 instruction words, making the total size of instruction memory 1024 Bytes. The CPU accesses a single instruction word at a time using a 10-bit word address (from the program counter) where the two least significant bits are zeros.

The instruction cache should split the address into *Tag*, *Index* and *Offset* sections appropriately. Finding the correct cache entry and extracting stored data block, tag and valid bits should be done based on the *Index*. Include an artificial indexing latency of #1 time unit when extracting the stored values. Then perform *Tag* comparison and validation to determine whether the access is a hit or a miss. Include an artificial latency of #0.9 time units for the tag comparison.

Read hits should be handled asynchronously similar to the data cache, with the only differences being the size of the address and word-size. Include an artificial latency of #1 time unit for selecting the requested instruction word from the block (which can happen in parallel to the tag comparison).

In the event of a miss, the CPU must be stalled using a `BUSYWAIT` signal. Cache controller should assert the memory `READ` control signal as soon as the miss is detected and start fetching the missing 16-Byte block from the instruction memory on the next positive clock edge.

Note that you are given a separate instruction memory module for part 3, which deals with blocks of 16 Bytes. A 6-bit block-address should be used when the cache is accessing the instruction memory. The fetching will take a long time (16×5 = 80 cycles), so the cache must wait (holding the relevant signals stable) until the memory de-asserts its `BUSYWAIT` signal to retrieve the instruction block.

On the positive clock edge this memory read completes, the cache controller should write the fetched block into the indexed cache entry and update the tag and valid bit accordingly. Include an artificial latency of #1 time unit in this writing operation. Then the original read access can be served by the asynchronous logic, where the status of the hit signal will change after further #1.9 time units, and consequently de-assert the `BUSYWAIT` signal of the cache at the next positive clock edge while sending the requested instruction word to the CPU.

Therefore, the total instruction miss penalty adds up 81 CPU cycles.

1. Implement the instruction cache module as specified and connect to the CPU via the testbench, along with the instruction memory module. Include **a lot of comments**.

2. Test your system thoroughly with several software programs. Programs can be hardcoded inside the instruction memory module or loaded from a file.

3. Submit a compressed file ***groupXX_lab6_part3.zip*** containing your Verilog files with all the modules in your design, testbench and screenshots of timing diagrams clearly showing signals related to instruction cache and CPU control.

Note that any form of plagiarism will result in zero marks for the entire lab.

Have fun coding. May the force be with you!