Computer Organization 2017

HOMEWORK IV

Overview

The goal of this homework is to help you understand how a cache works and how to use Verilog hardware description language (Verilog HDL) to model CPU with a cache. In this homework, you need to implement a cache that is used in an MIPS CPU. You need to follow the homework description in this homework and satisfy all the homework requirements. In addition, you need to verify your cache and CPU by using Modelsim. TAs will provide test fixtures that will run a MIPS program for your CPU and cache. TAs will use hidden test fixtures to test your cache and CPU. Please implement all the modules and use the test fixtures provided by TAs to verify your CPU and cache. You also need to take snapshots of your design and explain the snapshots in your report, including the wires and signals, etc.

General rules for deliverables

- You need to complete this homework INDIVIDUALLY. You can discuss the homework with other students, but you need to do the homework by yourself. If you copy your codes from someone else, you will not get any scores.
- When submitting your homework, compress all files into a single zip file, and upload the compressed file to Moodle.
  - Please follow the file hierarchy shown in Figure 1.

F740XXXXX(your id)(folder)

SRC(folder) * Store your source code

F740XXXXX_Report.docx (Project Report. The report template is already included. Follow the template to complete the report.)

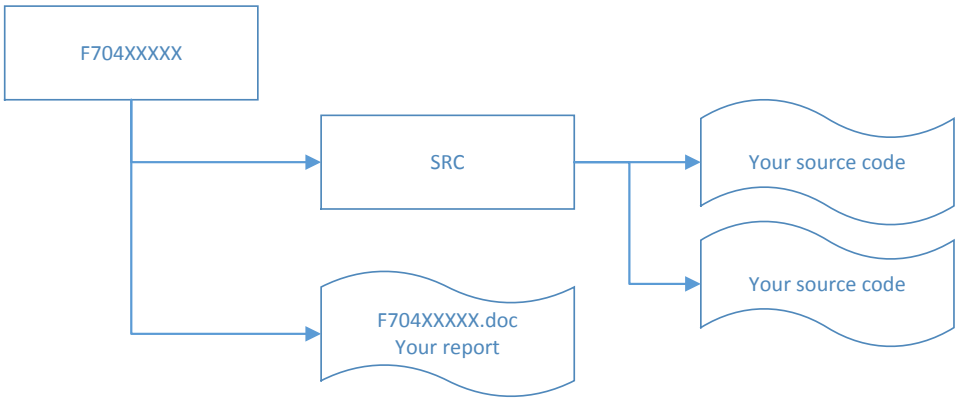

Figure 1. File hierarchy for homework submission

- **Important!** DO NOT submit your homework in the last minute. Late submission is not accepted.
- You should finish all the requirements (shown below) in this homework and the project report.

Homework Description

1. You need to implement a 128-byte directed-mapped cache that has 32 cache lines. Each cache line has one valid bit, 11 tag bits, and 4-byte data. When the cache receives the request from the CPU, there are four cases:
   a. Cache read hit – directly read the corresponding data in the cache
   b. Cache write hit – write data in both cache and memory, that is, write through is used.
   c. Cache read miss – obtain the missing data from the memory, and then place the data into the cache.
   d. Cache write miss –write into the memory only and do not write the data into the cache, i.e., write no-allocate is used.
2. Compared with Homework 3, this homework adds ICache between CPU and IM, DCache between CPU and DM. ICache and Dcache uses the same module definition but they are instantiated with different names. Figure 2 shows their relationship.
3. There are four modules, Cache_Control, Cache_valid, Cache_tag, and Cache_data, that will be used in the Cache:
   a. Cache_Control – this module handles cache hit or miss and sends signals to memory to fetch the missing data. The cache datapath is shown in Figure 3.
   b. Cache_valid – this module stores the **valid bits** of the cache.
   c. Cache_tag – this module stores the **tags** of the cache line
   d. Cache_data – this module stores the **data** of the cache

Figure 2. Relationship between CPU, ICache, DCache, IM, and DM

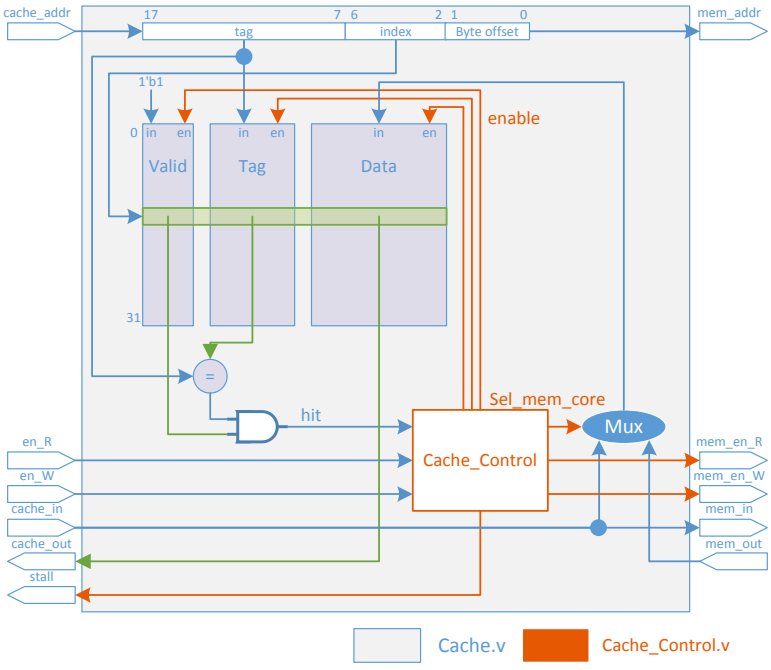

Figure 3. Cache Datapath used in this homework

4. The memory modules (IM.v, DM.v) are not ideal memories anymore, which means there exists latency during each access to the memory. In order to simplify the module, only the READ access to the memory has the latency, and it will follow the FSM (finite state machine) in Figure 4:
   a. Idle state: wait for en_Read signal
   b. Read data state: output the data and return to Idle state

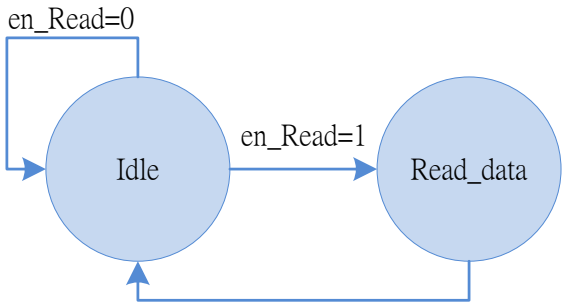

Figure 4. FSM in the memory

5. The Cache_Control also includes an finite state machine (FSM) to handle the read miss latency. The state diagram is shown in Figure 5:
   a. R_Idle: wait Read_Miss signal become 1, then raise Read_mem to inform memory
   b. R_wait: wait for the response of memory
   c. R_Read_Memory: get data from memory and **replace the corresponding cache line,** then return to Idle state

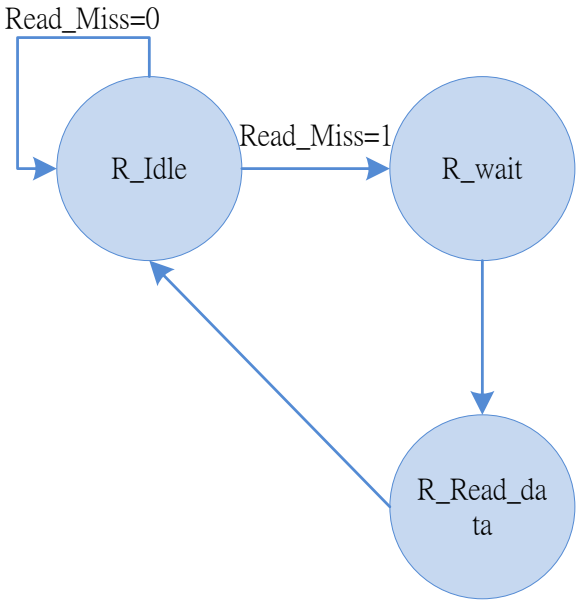

Figure 5. FSM when cache read occurs

6. Figure 6 explains all the I/O ports in cache module below:

Cache.v		
port	I/O	Description
clk	input	Clock signal
rst	input	Reset signal
stall	output	When Read miss occur, the signal used to stall the CPU
cache_addr	input	The request address from CPU
en_R	input	Represent the current access is read access
en_W	input	Represent the current access is write access
cache_in	input	The data input from CPU
cache_out	output	The data output to CPU
mem_addr	input	Memory access address
mem_en_R	input	Read memory enable signal
mem_en_W	input	Write memory enable signal
mem_in	input	The data received by memory
mem_out	output	The data that will be written into memory
Cache_Control.v		
port	I/O	Description
clk	input	Clock signal
rst	input	Reset signal
en_R	input	Represent the current access is read access
en_W	input	Represent the current access is write access
hit	input	Represent the current access is hit in the cache
Read_mem	output	Read memory enable signal
Write_mem	output	Write memory enable signal
Valid_enable	output	Enable writing valid part of cacheline
Tag_enable	output	Enable writing tag part of cacheline
Data_enable	output	Enable writing data part of cacheline
sel_mem_core	output	Select the Data_in is from memory or from the core(CPU)
stall	output	When Read miss occur, the signal used to stall the CPU
Cache_valid.v / Cache_tag.v / Cache_data.v		
port	I/O	Description
clk	input	Clock signal
rst	input	Reset signal
*_Address	input	The address (index) of the cache line
*_enable	input	The write enable signal
*_in	input	The input data that is written into address of *_Address when the *_enable is 1.
*_out	output	The output data that is read from address of *_Address

※ * = Valid, Tag, Data

Figure 6. Port description

Homework Requirements

1. Please implement these modules:
   a. Core.v (Modify your the pipelined CPU to connect with Cache).
   b. Cache_controll.v.
   c. top.v (You do not need to modify this file)

   The following files are from Homework 3 - Pipelined CPU. You may modify these files to connect with cache in this homework.
   a. HDU.v
   b. FU.v
   c. HDU.v
   d. IF_ID.v
   e. ID_EX.v
   f. EX_M.v
   g. M_WB.v
   h. Controller.v
   i. Regfile.v
   j. ALU.v
   k. PC.v
   l. Jump_Ctrl.v

2. Verify your cache and CPU with the test fixtures and take a snapshot (e.g. Figure 7). Explain the reason why the hit rate could reach such a high level **(you can explain your observing in the IM_data.dat and waveform)**

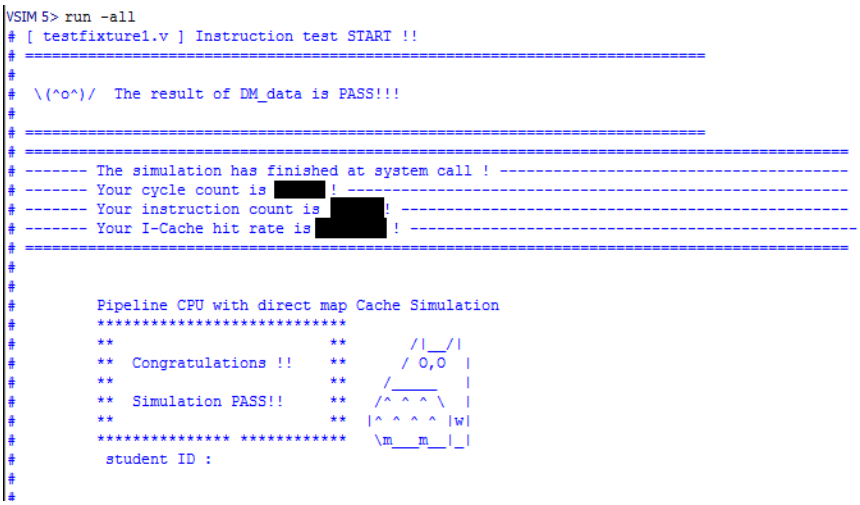

Figure 7. **S**napshot of correct simulation

3. Take snapshot
   a. Using waveforms to verify the execution results.
   b. Please annotate the waveform (as shown in Figure 8).

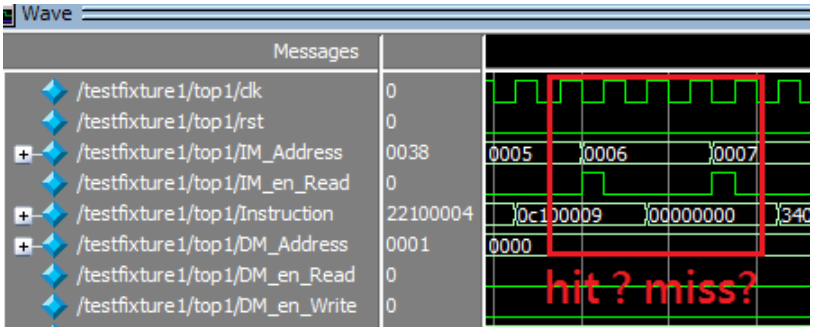

Figure 8. waveform snapshots

   c. List the following situations in the waveform snapshot. Then explain them as detailed as possible.
      i. I-Cache Miss –take snapshot of a “cache miss” access, explain what cache does and how much cycles there are between this instruction access and next instruction access.
      ii. I-Cache Hit –take snapshot of a “cache hit” access, explain what cache does and how much cycles there are between this instruction access and next instruction access.
      iii. CPU Stall –when cache miss happen, CPU must stall and wait memory. Explain how CPU stalls.
4. Finish the report.

Important

When you upload your file, please make sure you have satisfied all the homework requirements, including the **File hierarchy**, **Requirement file** and **Report format**.

If you have any questions, please contact us.