109-2 Digital System Design Homework #4

Cache Unit Design

Announced at May 6th, 2021

Deadline at 23:59 PM, May 20th, 2021

TA Information
馬咏治
kane@access.ee.ntu.edu.tw

1. Problem Statement

In a typical digital system design, operations are not executed directly on the data stored in main memory. Instead, processors load data into their own registers first, do operations on the copy of data in the register, and finally store the result back into main memory. Hence, processors need to communicate with main memory frequently. Since the clock rate of main memory is slower than that of the processor, such communications are usually the bottleneck of a digital system. For the purpose of facilitating the performance, caches are introduced between processors and main memory to take advantage of locality, and thus significantly decrease the miss rate. In this homework, you are asked to design cache units on your own.

Figure 1 shows a pipelined MIPS with both data memory and instruction memory included. Please implement two kinds of cache both containing **8 blocks with 4 words in each block**, one of them in **direct-mapped** and the other in **two-way associative** architecture. Both types of the write policy, direct write-through and write-back, are available so that you may choose one by yourself. The I/O specification and interface of the desired 32-word cache unit are illustrated in Figure 2.

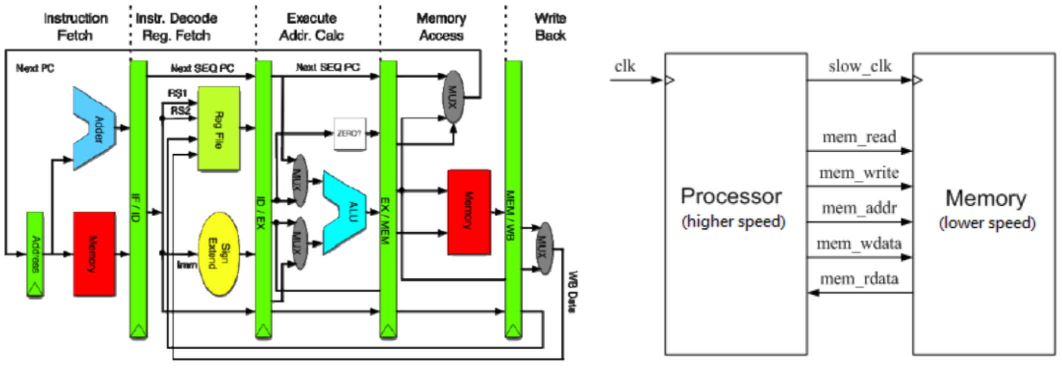


Figure 1. Pipelined MIPS processor and the memory interface.

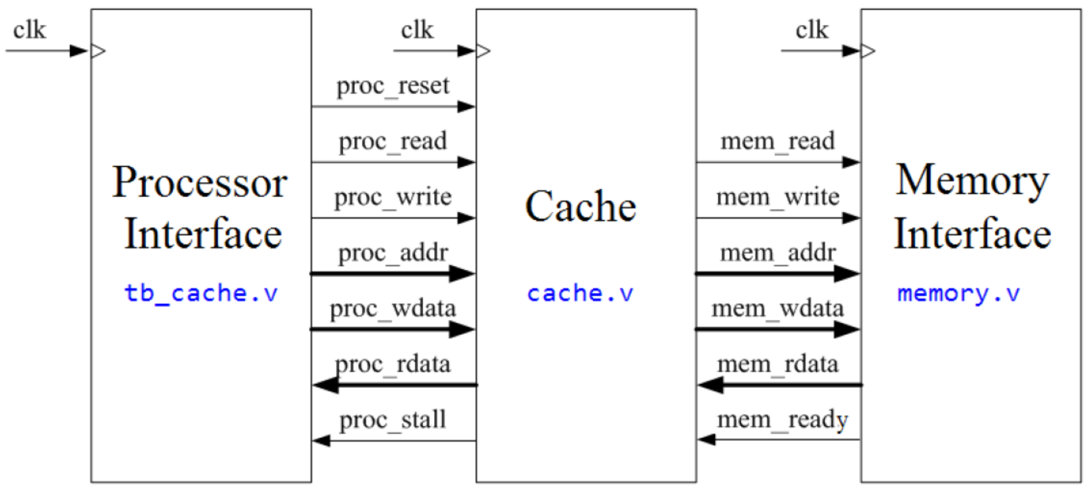


Figure 2. I/O specification and interface of the cache unit.

2. I/O Specification of the Processor Interface

Name	I/O	Width	Description
clk	I	1	positive edge trigger clock
proc_reset	I	1	synchronous active-high reset signal
proc_read	I	1	synchronous active-high read enable signal
proc_write	I	1	synchronous active-high write enable signal
proc_addr	I	30	address bus (***word address***)
proc_wdata	I	32	data bus for writing to cache
proc_rdata	O	32	data bus for reading from cache
proc_stall	O	1	active-high control signal that asks processor to wait (cache is busy)

3. I/O Specification of the Memory Interface

Name	I/O	Width	Description
mem_read	O	1	synchronous active-high read enable signal
mem_write	O	1	synchronous active-high write enable signal
mem_addr	O	28	address bus (***4-word address***)
mem_wdata	O	128	data bus for writing to slow memory
mem_rdata	I	128	data bus for reading from slow memory
mem_ready	I	1	asynchronous active-high one-cycle signal that indicates data arrives from memory / data is done written to memory

4. Address Mapping

A 32-word cache consists of 8 blocks, each containing 4 words. The processor accesses one word at a time, while the cache unit can only access 4 words at a time from the memory. Therefore, the last (rightmost) 2 bits of *proc_addr* represent the *offset* inside one block, indicating which word of the 4 words should be the target. The rest 28 bits are simply the *mem_addr* while accessing the memory. Figure 3 illustrates the address mapping between the processor, the memory, and the cache unit.

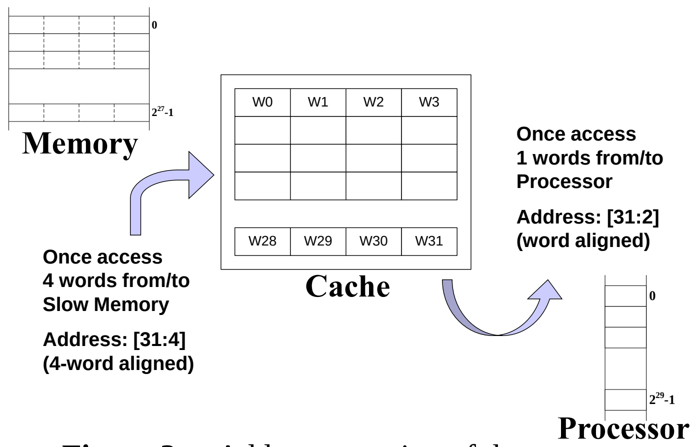

0
2^{27}-1
Memory
W0
W1
W2
W3
W28
W29
W30
W31
Cache
Once access 4 words from/to Slow Memory
Address: [31:4] (4-word aligned)
Once access 1 words from/to Processor
Address: [31:2] (word aligned)
0
2^{29}-1
Processor

Figure 3. Address mapping of the system.

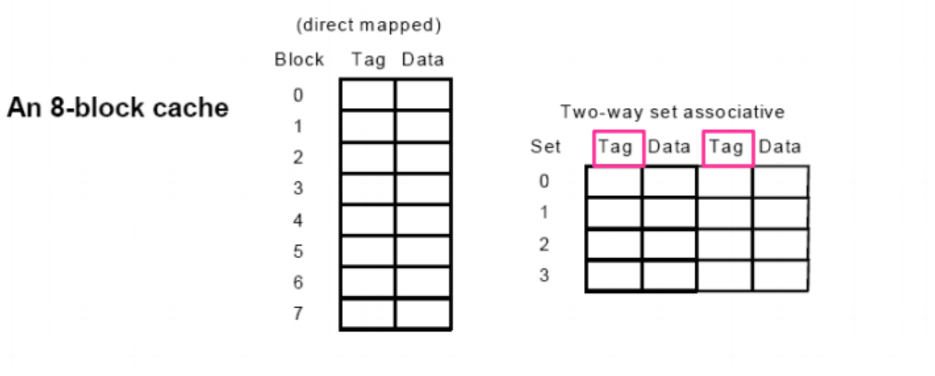


Figure 4. Address mapping style of two architectures

Reduce cache misses by adopting more flexible block-placement schemes:
(1) Direct mapped: A block can go in exactly one place in the cache.
(2) Fully-associative: A block can be placed in any location.
(3) Set-associative: A block can be placed in a restricted set of locations.

In a direct-mapped cache, the position of a memory block is given by

(block number) modulo (number of cache blocks)

In a set-associative cache, the set containing a memory block is given by

(block number) modulo (number of sets in the cache)

5. Functional and Timing Specification of the Processor Interface

In this homework, the processor interface is a pseudo processor that only performs the memory accessing tasks in order to verify the functionality of the cache unit. Therefore, all the signals from the processor interface change at the rising clock edge with a very slight input delay.

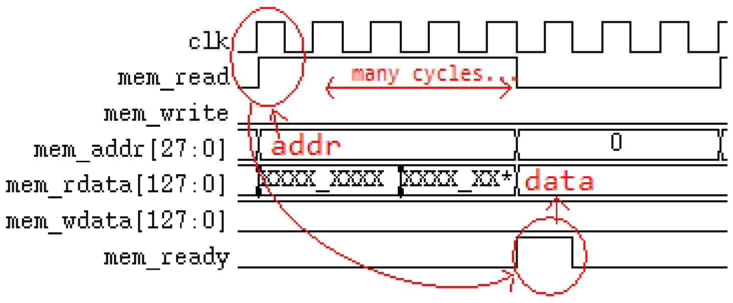
From time to time, the cache needs to access data from the memory and then wait for several cycles. Whenever this is the case, the *proc_stall* signal should be set high to stall the processor. The *proc_stall* signal should be set high before the next positive clock edge so that the processor can be correctly stalled. On the other hand, when there is no need to access memory, the cache simply finishes the job in one cycle. For example, in the read case, *proc_rdata* should be prepared and returned immediately to the processor at the same cycle if the required data are right available.

There are two possible cases where a stall is necessary:
(a) *Read miss* in both write-through and write-back caches;
(b) *Write hit* in *only write-through* caches.
Note that since *write miss* can be regarded as "*read miss then write hit*", a write miss in write-through caches involve two successive stalls, while only one stall is needed in write-back caches.

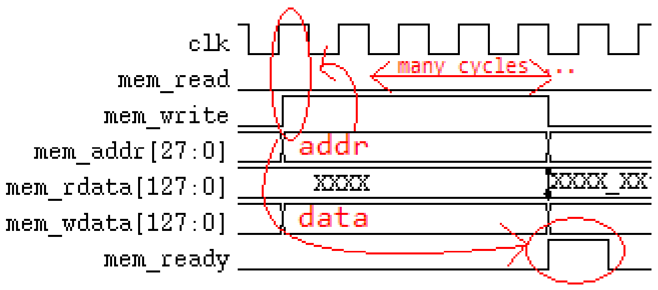
The pseudo processor in the given testbench performs only three tasks. It reads the whole initial data from the memory, rewrites the whole memory with new data, and finally reads the rewritten data back from the memory. All possible cases are taken into account, so you should design the finite state machine in the cache unit carefully and neatly.

6. Functional and Timing Specification of the Memory Interface

The memory interface is the functional model of an ordinary memory or a system bus. Figure 5 shows the timing diagram of the memory interface. After the memory is set to the read or write mode, it takes several cycles to complete the operation. The *mem_ready* signal serves as an indicator, saying that the required data is ready in read mode or the data is written correctly in write mode. If the *mem_read* and *mem_write* signals are both set high or both set low, the memory would do nothing.



(a) Read operation.



(b) Write operation.

Figure 5. Timing diagram of the memory interface.

7. Homework Requirements

Verilog Code.

Verilog code named "**cache_dm.v**" and "**cache_2way.v**", for the direct mapped and 2-way associative cache units respectively, are required. You need to synthesize your design, and make sure there are *no latches, timing violations, or negative endpoint slack*, which are things not allowed in a synthesizable design. Performance is not part of the grading criteria this time, so you are not required to optimize your design.

Post Synthesis Related Files.

These include SDF files, DDC files, and the gate-level design Verilog code.
(Six files in total. See Submission Requirement.)

Report.

The report should be named "**report.pdf**" in **PDF** format, and should contain at least the following parts:

(a) The cycle time you used in cache_syn.sdc and in tb_cache.v to pass the post-synthesis simulation respectively.
(b) General specification of the cache unit.

(such as numbers of words, placement policy, and so on);

(c) Read/write policy (write-through or write-back);

(d) Design architecture or the finite state machine of the cache unit;

(e) Performance evaluation of your cache design, including the miss rates of read/write operations, the execution cycles, the stalled cycles, and so on;

※ Hint: Try to add a few lines in tb_cache.v for calculating the miss rates, the execution cycles, and the stalled cycles on your own. Just make sure you fully understand the different conditions of I/O signals when a cache hits or misses.

(f) Compare the performance of the two architectures, and discuss the reasons for such results.

We would appreciate it if you give a discussion on the cache architectures, or describe your design guideline in details and the experiences you gain from this homework.

8. Grading Policy

Note that *the report is as important as the Verilog code*, so finish the code as soon as possible and spend time for a qualified report.

Report (30pts).

Qualified report will be given full marks. A report that is simply a copy-paste of the simulation or synthesis results will only get at most 60% for this part.

Cache design (35pts for each cache unit).

Here are the general rules. The evaluation is based on the given testbench "**tb_cache.v**".

(a) RTL simulation (20pts):

(b) Synthesis & gate-level simulation (15pts):

15pts: Pass the gate-level simulation.

10pts: Pass the gate-level simulation, with some timing violations or negative endpoint slacks.

Note that this part is graded only if your design passes the RTL simulation and no combinational circuit components are inferred latches!

Bonus.

If your cache units are implemented with other skills that enhance the performance significantly, there will be a bonus up to 10 points. You should describe your methods or architectures in detail in the report!

Late submission penalty.

20% off per day.

9. Simulation / Synthesis Notes

Take cache_dm.v as an example

[Simulation]

```
source /usr/cad/cadence/cshrc
source /usr/spring_soft/CIC/verdi.cshrc
```

RTL

```
ncverilog tb_cache.v cache_dm.v memory.v +access+r
```

Gate level

(**Remember to modify the SDFFILE name in tb_cache.v**)

```
ncverilog tb_cache.v cache_dm_syn.v memory.v tsmc13.v +define+SDF
+access+r
```

[Synthesis]

```
source /usr/cad/synopsys/CIC/synthesis.cshrc
```

Design Compiler

```
read_file –format verilog cache_dm.v
source cache_syn.sdc
compile
write_sdf –version 2.1 cache_dm_syn.sdf
write –format verilog –hier –output cache_dm_syn.v
write –format ddc     –hier –output cache_dm_syn.ddc
```

10. Submission Requirement

1. Compress all the files into a single **ZIP file** and **upload it to CEIBA**.

   Sample filename

   DSD_HW4_b05901xxx.zip

   DSD_HW4_b05901xxx_v2.zip

   The submitted ZIP file should include the following files:

   DSD_HW4_b05901xxx/

   report.pdf

```
rtl/
        cache_dm.v
        cache_2way.v
syn/
        cache_dm_syn.v
        cache_dm_syn.sdf
        cache_dm_syn.ddc
        cache_2way_syn.v
        cache_2way_syn.sdf
        cache_2way_syn.ddc
```

Submission with wrong filenames or directories will get some penalties.

2. **Deadline: 2021/05/20 23:59**