Memory Hierarchy for RISC-V Processor

Introduction

Up until now the considered 5-stage RISC-V processor design utilized small-capacity instruction and data memories with asynchronous reads and synchronous writes. It was assumed that these memories can be always read or written in one cycle. This lab’s goal is to develop a processor with a more realistic memory hierarchy. The specific focus of this lab is on implementing instruction cache. For simplicity, we will assume no virtual memory and use previous design for the data memory stage. Besides gaining experience with cache design, this lab also exposes to the implementation of simple handshake protocols for sending and receiving data via buses that are common in memory hierarchies.

Baseline Instruction cache

The specific task is to implement instruction cache module that consists of cache and cache controller. Instruction cache module is connected to main memory (e.g., implemented with typical synchronous dynamic access memory, SDRAM for short) via controller (Figure 1). The controller arbitrates read and write requests from data and instruction caches. For simplicity, this lab assumes emulated ideal data cache (with no misses) and emulated functionality of main memory and its controller.

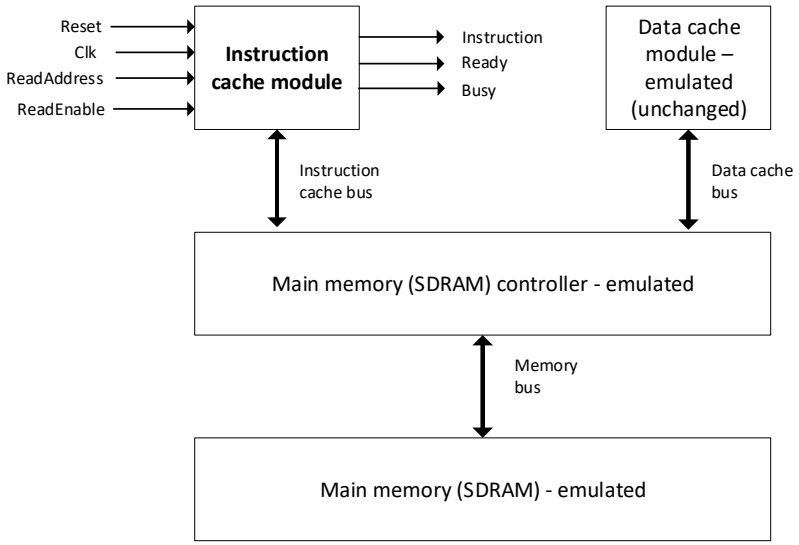


Figure 1. The considered memory hierarchy.

Main points for the implementation are summarized below:

- The L1 cache is 4-way associative with random replacement policy (RRP). Random replacement policy assumes randomly chosen ways for replacement. For larger number of ways, the miss rate of RRP’s cache is just slightly higher

than that of cache with least recently used policy. The RRP's upside is simpler implementation, e.g., no need to keep additional bits and updating them during read/write operation. The number of sets and the number of words per block (and hence cache capacity) should be parameterized. The default values are 8 sets and 4 32-bit words (32 bytes) per block, i.e., 512B capacity.[1]

- To speed up read operation, assume that the whole cache line (i.e., data, tag, and valid bits of all ways) are read in parallel. Because of random replacement policy, there are no bits that have to be updated on a read or write operation.

- Reset synchronously clears all fields in a cache. In our implementation, the reset is only applied at the cold start.

- Both reads and writes to a cache are **synchronous** with rising edge of a clock (Figure 2). Further, we assume single-port cache implementation, i.e., with one address port shared between reads and writes operation. Read operation can be performed only when cache is not busy, i.e., Busy = 0. The cache controller raises Busy output when it writes to cache.

- Read operation is initiated by raising ReadEnable and by supplying a valid read address ReadAddress at least setup time before the rising edge of the clock. The purpose of the ReadEnable signal is to prevent from performing redundant reads, e.g. when fetch stage is stalled, or reads from junk addresses which can pollute cache and/or raise exception.

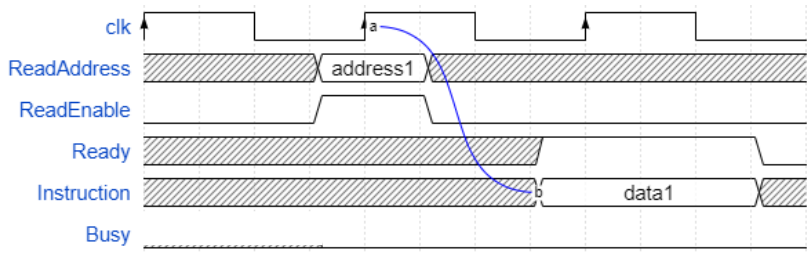
- Ready signal indicates when the instruction cache module outputs a valid instruction. If cache access is a hit, the valid instruction data appears at the Instruction and Ready signal is set to 1 at some time after the rising edge of the clock but within the same clock cycle. If cache access is miss, cache controller sends request to read block of data from main memory via SDRAM controller. In that case, Ready = 0 until valid data are read from main memory and supplied to the Instruction. The high Ready signal and the valid data at the Instruction are only kept for one cycle so it is up to the consumer of data (processor) not to lose it.

- SDRAM controller operates at processor clock. It has somewhat similar interface to instruction cache module, though performs burst reads at block granularity and has different handshaking protocol. The bus consists of ReadAddress, DataIn, ReadRequest, and DataReady signals (Figure 3). The ReadRequest = 1 and the valid ReadAddress should be supplied until receiving DataReady signal. The SDRAM controller sends the first word of data to cache

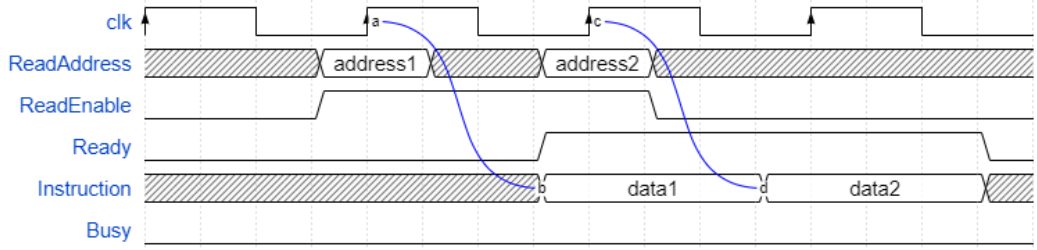
[1] Rocket core also uses RRP and has default values of 16KB capacity, 4 ways, and 64 bytes per block blocks for both instruction and data L1 caches.

controller via DataIn after T_0 delay and raise DataReady to indicate beginning of the block transfer. The subsequent words are then supplied, one per T_{burst} cycle. The DataReady signal is lowered by the SDRAM controller automatically after all words are supplied so that the new read block operation can be initiated by cache controller, if needed. We will assume that reading the first word takes $T_0 = 40$ cycles, while $T_{burst} = 1$. Therefore, reading the N-word block from SDRAM takes $T_0 + T_{burst} \times (N\text{-}1)$ processor cycles. [2]

(a)



(b)



(c)

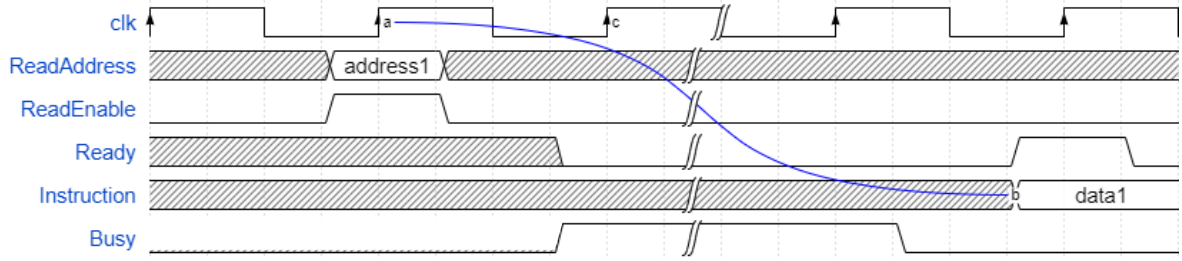
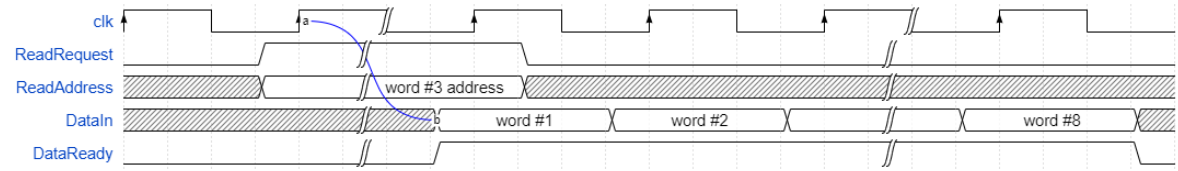


Figure 2. Timing diagrams for reading from instruction cache for (a) a single hit, (b) back to back hits, and (c) example of miss. The miss penalty depends on the cache implementation, which is at least $T_0 + T_{burst} \times (N\text{-}1) + 1$ for the baseline implementation, i.e., $T_0 + T_{burst} \times (N\text{-}1)$ for reading the block plus 1 extra cycle to write it to the cache. It is $T_0 + 1$ for the advanced (discussed below) implementation.

[2] In more realistic implementation, T_0 can vary due to, e.g., performed refresh operation or SDRAM controller servicing higher priority data cache requests. $T_{burst} > 1$ is typical due to slower and/or narrower memory bus. Also, the accessed words in a burst mode operation should be all within the same physical row (called page) of SDRAM. In this lab we neglect all of these issues.

- In the baseline implementation, the cache controller reads the whole block from the main memory before supplying the word in question to the processor. For example, if the miss was on the 2nd word of the 8-word block, the cache controller will read first 0th word, then 1st word, and so on, and only after receiving 7th word of a block, it will supply 2nd word to the processor. Specifically, after all the data of the block are received, the cache controller should update the randomly selected way in cache (block, tag, valid), and, in the next cycle, it places the word in question on the Instruction and raise Ready signal.

(a)



(b)

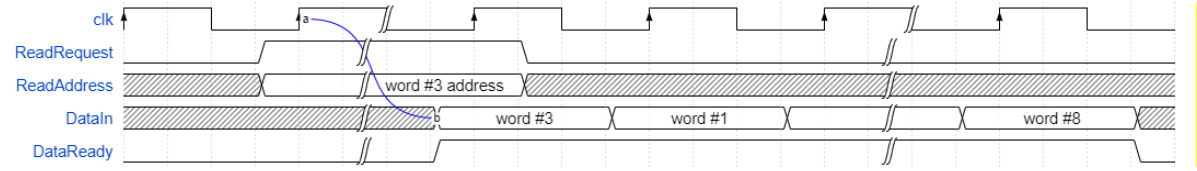


Figure 3. Timing diagrams for missing on the word #3 and hence reading the corresponding block from main memory via SDRAM controller for (a) baseline and (b) advanced (critical-word-first) designs.

Advanced implementation with critical word first and early restart

In the more advanced, critical-word-first implementation, cache controller requests and receives word in question of the block first (Figure 3b). Once received from SDRAM controller, the cache controller places this word on Instruction output and raise Ready output, not waiting for the whole block to be transferred. This is called early restart mechanism. In parallel, the cache controller continues fetching remaining words of the block. As earlier, the cache controller writes the block (in one cycle) after all words are received, raising Busy signal during the write cycle to prevent read operation. The cache controller could process future reads to the block currently being fetched, if the required words are already available from temporary buffer.

Hardware prefetcher

Hardware prefetcher can reduce miss penalty.[3] When there is miss on the word in the block with address A, hardware prefetcher reads additional blocks at addresses A+1, A+2, … , and place them in the local prefetch buffer. With spatial locality, there is a good chance that next misses will be into one of those blocks and, if so, they can be faster processed by reading from local buffer.

For the read request, the cache controller checks both cache and prefetch buffer in parallel. There are three important cases – (1) the requested word can be in cache (hit), (2) not in cache (miss) but in prefetch buffer, and (3) neither in cache or buffer. For (1), the requested word is supplied from cache as usual. For (2), the requested word is supplied from prefetch buffer with no miss penalty. In the next cycle, the block in question from the prefetch buffer is written into cache, and the prefetching of a new block from main memory is initiated to replace the block that was moved to cache. For (3), cache controller requests the block in question from main memory just like for a typical cache miss as described in previous sections, and after that request adjacent block(s) for the prefetch buffer.

Block transfer from SDRAM controller to cache controller cannot be interrupted. Hence, prefetching too many blocks can clog the bus and increase the miss penalty. For these reasons, consider prefetching just one additional block. Also, for simplicity, assume prefetch buffer stores only one block.

The already initiated prefetching of the block must be finished even there is outstanding miss that goes to different block. However, a block prefetch could be canceled if it is not yet started.

Project assignment

1) Develop RTL design and verify it functionality for the baseline implementation of the instruction cache. Use provided skeleton files (for "ucsbece154b_imem.v", "ucsbece154b_emm_srdam.v" modules) and modify "ucsbece154b_datapath.v", "ucsbece154b_top_tb.v" and "ucsbece154b_controller.v" accordingly.

   As mentioned above, make sure to implement read synchronously in Verilog with rising edge clock. The synthesis tools should automatically infer such design as synchronous RAM and map it to much more efficient (i.e., denser, higher capacity, faster) block RAM, which is a dedicated SRAM arrays, in FPGA. In contrast, the design with asynchronous read is implemented with much sparser, slower arrays of flipflops or LUTs, i.e., the so-called distributed RAM in FPGA.

2) Develop RTL design for the critical-word-first and early-restart design and verify its functionality.

[3] Rocket core has optional prefetcher for both L1 data and L1 instruction caches

3) Implement prefetcher circuit for the advanced design.

4) Measure miss rates (via additional counter in the testbench) and explore the dependency of miss rate on the capacity and block size on the assembly codes (e.g., those used in Labs #1 and #2) for baseline, advanced designs, and prefetcher with advanced design.

In your RTL code, one should be able to choose the specific design (baseline, advanced, and advanced + prefetcher) by configuring a parameter.

You may use your solution for lab2, i.e., replace ucsbece154b_datapath.v and ucsbece154b_controller.v in the provided package with your files from lab2.

For verification and performance analysis, modify “ucsbece154b_top_tb.v” for testing to measure miss rates and program execution time.

Hints for the project:

- Synchronous read can be performed by feeding PCNewF as a read address to the instruction cache module in parallel to the PC register (as opposed to using PCF as it was done in the lab2). This is similar to Rocket L1 instruction cache implementation.

- In this lab, the unwanted (redundant) reads from instruction cache only occur when stalling fetch stage, so that ReadEnable = ~StallF.

- Ready signal from cache module could be used to determine whether to insert a bubble or push a valid Instruction into IF/ID pipeline register.

- Before writing and debugging your code, analyze miss behavior of the cache for the given program so that you know what to expect.

Grading and deadlines

- **Part A [20 points].** Lab check off, due **May 14th** in CSIL lab
- **Part B [60 points].** RTL code, due **May 16th**
- **Part C [20 points].** Lab report due **May 16th**

What to Turn In

1. Send RTL files and report in pdf format as a single compressed file to TA. The report should contain the following items:

    a. FSM diagrams for all implementations
    b. The block level diagram for the prefetcher

c. Cache miss rates and processor performance for three considered cases (baseline, advanced, prefetcher + advanced) for the default settings
d. Cache miss rates as a function of cache size and block size (e.g. fixing one and varying the other)
e. Analysis and discussion of the results
f. Indicate how many hours you spent on this lab