RISC-V Cache Implementation

Konstantinos Oikonomopoulos
AEM: 04012
Electrical & Computer Engineering
University of Thessaly
Volos, Greece
koikonomop@uth.gr

Dimosthenis Zempilas
AEM: 03857
Electrical & Computer Engineering
University of Thessaly
Volos, Greece
dzempilas@uth.gr

Kostantin Ziou
AEM: 03950
Electrical & Computer Engineering
University of Thessaly
Volos, Greece
kziou@uth.gr

1. Introduciton

This document is a presentation of the work done during the summer of 2025 by the authors, a continuation of the work by P. Nanousis in the *"RISC-Y"* project which was the continuation of the work of A. Giannousas, A. Kastoras and I. Panagou. RISC-V is an open-source ISA which's implementation with VHDL (verilog) was provided to us by the previous projects. The improvement added was separate Instruction and Data Caches as well as ideal Main Memories, the implementation of which is described in the following sections.

2. Framework (soc)

The Instruction and Data Cache are the same module instantiated differently depending on the needs of each. That means that they work the same on the inside but treated as a "black-box", they require different wiring/connections to fulfill their purpose. More than just what stage of the pipeline they are introduced in, for example, Instruction Cache is never enabled for writing (Write-Enabled, "wen"), therefore the enabling input of the Cache module, which is dynamically set for the Data Cache, is set to low (0). Likewise the input for writing inside the Cache from the core is also always set to low. The rest is obvious through intuition as to what are the differences; the address for the Instruction Cache is Program Counter (PC), while the address for the Data Cache is the data address provided, similarly on's output is the instruction to be decoded in the next stage while the other's is the data stored in the given address.

3. Cache

The main implementation (provided in Github) is that of a No-Write-Allocate, Write-Back, 2-Way Set Associative, 8 set Cache with LRU replacement. Quite simply an address is indexed (2-way associativity) in the Cache and there is a case of finding and not finding the memory requested. In case of a Hit in the Cache, for reading operations, the data is given to the output, while for writing operations, the data is overwritten on the address, setting the Dirty bit to high (1), which signifies data aliasing with the main Memory. In both operations, the LRU is changed accordingly to which line was most recently accessed. In the case of a Miss, for reading operations, an appropriate signal is given to the Main Memory which provides the data of the requested address to be written in the next cycle in the Cache, while the data is being written in the Cache, on the same cyle, the output connected to the core is provided with the same data coming out of the Main Memory with forwarding logic, while for writing operations, because the Cache is No-Write-Allocate, the data as well as the address is overwritten on the appropriate Cache address without prompting the Main Memory. The replacement address for both types of operations in the case of a Miss is done with the indexing of the LRU algorithm, while it should be noted that both operations also alter the LRU bit. If the Dirty bit is set to high then the data stored is sent back to the Main Memory to be written along with a signal to enable writing operations. To guarantee that there are not false Hits in the empty Cache, valid bits are introduced to differentiate between randomly set bits and actual data.

4. Main Memory

The Main Memory is split into Instruction and Data Memories, which is equivalent to segmenting a singular memory. The Main Memory is treated as a black box since the focus of the project was mostly the Cache. Addresses are indices to a 2-dimensional array. There are 1024 32-bit lines currently, while much less are required for the tests/examples implemented. Despite it being an ideal memory with only 1 cycle stall, all the signals and connections required for a safe implementation according to industry standards are in place.

5. Final notes and Future Work

To much the determent of the authors, a byte selector is in place. When reading from the Cache the byte selector indexes which bytes are read, while when writing information from the main memory to the cache the entire data is written to prevent needless aliasing (in spite of the dirty bit). The byte selector is also used when forwarding data from the Main Memory to prevent inconsistencies. Instruction and Data pre-fetching for the Cache could be implemented in future work, while parametrization of the Cache's size and associativity is still a work in progress. Parallelism is always fun so adding multiple cores and appropriately changing the writing policy to avoid aliasing is a challenge worth its time.