RETROSPECTIVE:

# Lockup-Free Instruction Fetch/Prefetch Cache Organization

*David Kroft*

14 Kings Inn Trail
Thornhill, Ontario
L3T 1T7

How does one begin to describe the dreams, thoughts and fears that surround a discovery of a different view of some old concepts or the employment of old accepted methodology to new avenues? It is probably best to start the account by describing the field of Computer Architecture, in particular, the area of hierarchical memory design, that was prevalent around and before the time the ideas came to light.

In the mid seventies, I was fortunate to be selected as one of the members of a design team to design and develop the central processing unit (CPU) for a low end model of Control Data's new line of main frame computers. Since integrated logic circuit components were now readily available and, consequently, computer hardware was much cheaper, the new line would feature a highly expanded instruction set — the move in vogue at that time was toward complex instruction set computers (CISCs). Note, that due to the price of hardware, all former CPUs were of the reduced or minimum instruction sets varieties (RISCs). The fall of the cost of hard logic and memory, also, allowed for the implementing of the new hierarchical memory design concepts into these next computers to be designed, developed and manufactured. Control Data, or should I say, the technical visionaries of Control Data at that time had differing opinions as to the advantages of hierarchical memory design. I recall one of these visionaries telling me the following: "We, at Control Data, have the know-how to design central memories big enough, fast enough and put them close enough to the CPU that memory hierarchy would never be needed." I was, however, able to convince at least a sufficient number of these technical gurus that a Cache Memory would be advantageous to the new CPU so that I was given the assignment to design the Cache for this low end model of the line. The fact that IBM was now incorporating Caches in all their new designs, obviously, helped me considerably with my persuasion. Note that there were other later computers designs at Control Data that did not have Caches — believe it or not.

So, there I was in the mid seventies, having never designed anything real up to then — I had accomplished a lot on paper — given with the assignment to solely perform the system design and logic design implementation of a Cache for this new low end model with its extended instruction set. I say, solely, because no Cache had ever been included in any of the Control Data designs previously and there were a number of "Doubting Thomases." Due to the opposition or possibly since I had still had the courage and adventure of youth — I was only in my early thirties, I embarked on a very ambitious design approach.

First, I included most of the latest Cache concepts that were contained in the literature. I decided on a set associative organization with 128 sets (rows) of four or eight blocks (lines) per set and a block size of 32 bytes (4 8-byte CDC words). In addition, write through and a least recently used (LRU) replacement algorithm were chosen. These selections were made after much study and consideration of the simulation data in the literature and with the restrictions imposed by the then available memory components (a fast 256 by anything was not on the horizon while a fast 128 by 1 was there.) The two set sizes (columns), (4 and 8), allowed for the two required Cache sizes of 16K and 32K bytes. In retrospect, the complexity (i.e. additional cost) of the additional associativity for the marginally higher hit rate for the larger Cache was the only decision I regret.

Next, I decided to incorporate the additional cost of having the Cache's address space be an unique virtual address space thereby putting the cache closer to the processor; the translate lookaside buffer (TLB) would now operate in parallel with the Cache rather than before it. Note that the Cache only needs the real memory address from the TLB on a miss. Lastly, just before the miss overlap feature (from paper) inclusion in the design, I discovered a way to allow for the graceful degradation of the Cache via the disabling of any one of the four 8K bytes sections at a time. The above required just the need to determine the maximum LRU of the enabled blocks in a set when block replacement was needed.

Recall that this new line of machines envisioned by Control Data had an extended instruction set. In fact, the instruction set included almost all the instructions of all formats that were present in one machine or another. Many of the instructions made more than one reference to memory. At first, I thought, how can one order the memory references so that the first references always produce a hit in the Cache to prevent impact on the following ones. At the same time, I saw that the Cache had effectively two input ports, one from the execution unit and one from the instruction unit. Why should the hit/miss of one port impact the other, I reflected? I was told that a two port Cache with each port totally independent of one another was an open problem with no solution known as yet and the route most manufacturers were taking to circumvent this two port problem was to have two Caches, one for each unit — these units, it was known, each had different classes of data. There were, now, two cases for miss overlap or for Cache hit(s) being processed while a Cache miss was outstanding. How can this be accomplished?

I associated the above problem with everyday occurrences. I pondered "what if" there was a queue of people waiting for service from one particular individual and this individual could service these people either with one of two possible scenarios - a long time consuming service due to the need for additional parts, information, etc. or a short quick response. The solution for minimum wait time would not be solved by reordering the people in the line; a second individual is required. How should this second individual help? The second individual would back up the first by going to get the necessary parts, information, etc. while the first continued to service the queue. The above thought process permitted me to see a vision for overlapping misses. I will save all the relevant information about a miss, forward a request to a "block getting unit" for the block needed and then, continue processing the input service requests. A block receiving unit will interrogate the relevant information about a miss, forward the response to the waiting individual and update its own unit, the Cache buffer, if advisable. The block getting unit and the block receiving unit are this second individual. A light has come on; the long process of determining all the details and workings must now be done to determine viability. Fortunately, all fell into place, if not initially, then during debug.

As indicated at the beginning, a discovery is just a different look at some old concepts or the use of some old concepts for new mechanisms. Mine was the latter.