CS 5341

Oscar Galindo

**An Efficient Algorithm for Exploiting Multiple Arithmetic Units**

This work begins by indicating that even with all the technical advances that had occurred at the time, the technical advance missing was the “[optimization of] the actual performance of arithmetic operations, especially floating-point”. The author further elaborates that a solution is not to attempt a classification of operations (i.e., fixed-point, or floating-point) individually, but to attempt an identification of the relationship of any single instruction with the rest of instructions in the execution sequence to maximize execution overlap. To provide for the requirement of an increased execution overlap the author proposes the common data bus or CDB.

After presenting the syntax of instructions which is Register op Register Register, or source op buffer/source sink, and proceeds to explain the execution and decoding procedures for a single instruction. Having explained the process by which an instruction is executed and decoded the author presents a ‘challenge’. In such a challenge a load occurs that has as target a register, that same register is used in the next line to be one of the two operands needed for a multiplication instruction. The ‘challenge’ occurs, as described by the work, because “The load can be handled as before, but what about the multiply? Certainly, F0 and FLB2 cannot be sent to the multiplier … since FLB1 has not yet been set into F0”. From the previous issue, the author shows an important principle that is to be followed “… a register cannot be used until its contents reflect the result of the most recent operation to use that register as its sink[destination].” In order to implement a mechanism to address the ‘challenge’ previously presented the paper outlines three major functions of a proposed mechanism:

1. [The mechanism] must recognize the existence of a dependency.
2. [The mechanism] must cause the correct sequencing of the dependent instructions
3. [The mechanism] must distinguish between [sequences with dependencies and without dependencies, to allow execution overlap].

Immediately after presenting the needed functions for a mechanism to address the need of execution overlap the author introduces the idea of a “busy bit”. Such bit basically identifies if the previously mentioned ‘challenge’ is occurring, or when an instruction being executed has as a destination a register which is a dependency for the correct execution of the next instruction. Unfortunately, as mentioned in the work, the “busy bit” scheme prevents the overlap execution of other instructions that occur after that second instructions. So, the work proposes a better solution in associating more than one set of registers with each execution unit - this is called a reservation unit. As an example of the performance gained, the work provides a speedup in a set of executed instructions of around 30% (i.e., from 31 to 26 cycles) when reservation units are implemented with “busy bit”. Although for some other cases, like looping, the work shows this implementation of reservation units alongside busy bit fail to aid on performance.

The work introduces the CDB (i.e., Common Data Bus) -alongside reservation units and “busy bits”- by pointing out that “As far as action by the FLOS is concerned, the only thing unique to a particular instruction is the unit which will execute it.” Implying the proposed solution will come from exploiting the ‘directionality’ of where and what unit will execute an instruction. For this the work proposes to add circuitry to combine reservation stations with output ports on Floating Point Operation Buffers which have the task of combining results from the adder and the multiplier divider, the combination of arithmetic results is also called the CDB. With such implementation the CDB allows “loads to be executed without the adder and will make the result of any operation available to all units without first going through a floating-point register.” To control how data is moved through the CDB data path the work proposes to implement a 4-bit addressing encoding which is call a tagging system and whose addresses are tags. The idea behind using tags is to be able to redefine sources for the value of registers, so, that if successive instructions exist then the adders/multipliers can implicitly pass around data between them – through the exchange of sink tags between reservation units- and finally only one of those adders/multipliers gets to have its result ‘consumed’ by the register. This aids in execution performance because of a decreased transmission cost when a single register is the sink and source. In addition, this aids in execution time because it allows for the overlap execution of other independent instructions.

The work finishes pointing out that some issues exist that could stop the CDB scheme from aiding the execution performance, but that mainly happens when the same sink register is used for both single- and floating-point operations. While on the bright side CDB does comply with the three outlined functions needed to address the issues that stopped overlap execution and caused a poor percentage of use of the machine architecture. Finally, the work offers some conclusions among them is the most important conclusion “CDB utilizes the reservation stations and a simple tagging scheme to preserve precedence while encouraging concurrency … It should be evident,… that the programmer still exercises substantial control over how much concurrency will occur.”

In my opinion this paper presents an elegant and simple solution to a massive problem, which to this day we solve in this way (or so I believe). I do not believe there is much to say about this work besides it is great and that an update of this work on newer architectures would have been nice.