

## Method and apparatus for the selective scoreboarding of computation results

*M. K. Gschwind*

Statically scheduled machines based on long instruction word architectures offer many benefits, among them a high amount of available parallelism and simple control structure due to compiler based scheduling. As a result, we expect statically scheduled machines to achieve higher frequency with simpler design than would be possible with out-of-order superscalar processor designs.

Statically scheduled machines do have a disadvantage when dealing with dynamic events, such as cache hit or miss detection. Early VLIW machines were designed without caches, to achieve predictability in memory access. However, such designs suffer in memory performance. To achieve high performance, VLIW architectures must have adequate support for using caches. A simple VLIW design might use an architecture based on a stall-on-miss design, whereby the entire processor is stalled when a cache miss occurs. This design point is the most natural design option for the static nature of VLIW architectures.

However, stall-on-miss introduces stalls even when there is no immediate use for the result which is loaded. This may occur because a load instruction has been scheduled early enough to allow ample time for cache miss service, or for a speculatively issued load which will never be used because an alternative path is executed instead. An alternative implementation choice is stall-on-use whereby the processor only stalls if a results is unavailable when another instruction attempts to access it. Availability of resources for implementing stall-on-use policies are usually based on scoreboarding.

Most modern machines employ scoreboarding to achieve good performance. In dynamically scheduled architectures, the appropriate logic is at the core of instruction dispatch and thus a required part of the functionality. However, stall-on-use logic and scoreboarding are more expensive to implement in statically scheduled machines since the aim of such machines is to eliminate most of the control logic which performs this functionality in dynamically scheduled machines.

In particular, scoreboarding introduces critical paths for the reading of scoreboarding of input operands and marking output operands as unavailable until the operation has completed. This has to be performed in a single cycle, and has to include broadcasting of scoreboard updates to multiple copies of the register files, or clusters. A further complication is the resetting of scoreboard operations when an operation is aborted, e.g., due to a TLB miss or some other exceptional condition.

This problem is addressed by the present invention which has been used in the context of the BOA binary translation EPIC architecture [1]: Scoreboarding is used selectively for instructions with unpredictable execution time. All other instructions, i.e., instructions with a known execution time are not scoreboarded. In addition, instructions with unknown execution time are not scoreboarded during the first several cycles of their execution. Instead, static scheduling is

required to at least cover a specified number of cycles corresponding to the unscoreboarded execution phase of an instruction before placing a dependent operation which should interlock.

This approach offers the advantage of eliminating the tight path from reading the scoreboard bits for the input registers, and setting the output register scoreboards by broadcasting them to all functional units. In addition, the architecture can be structured to perform all tests for exception occurrence during the initial phase when the outputs have not yet been scoreboarded. Thus, when such instruction is aborted the scoreboard information does not need to be revoked.

A similar approach can also be applied to supply scoreboarding only for long-running instructions (e.g., floating point or divide operations) in a statically scheduled architecture, to eliminate the need to find a sufficient number of independent operations to execute in order to cover the latency of such operations.

### ***References***

[1] M. Gschwind et al., Dynamic and Transparent Binary Translation, IEEE Computer, March 2000.