Improved Value Speculation

D. Morano
Northeastern University
dmorano@ece.neu.edu

3rd May 2002

1 Introduction

This document will outline an improved method of handling value speculation in Levo-like microarchitectures. The improvement is a relatively small change on the current method and only presents a small increase of hardware in each Active Station (AS). However, the improvement may be useful when either Processing Element (PE) or Forward Bus bandwidth needs to be conserved or reduced. The improvement reduces the amount of speculative execution that results from speculated values that originate in the current scheme, while at the same time not directly contributing to longer residency times (in clocks) for instructions within the execution window of the machine.

2 Current Value Speculation Technique

The current value speculation scheme consists of slightly different actions depending on the state of an executed instruction that is loaded into an AS. When an instruction gets dispatched (transferred from a fetch buffer to an AS) it enters the execution window and is ready to execute if resources are available. The first thing that an instruction needs before requesting an execution resource is to determine its source operands. For the MIPS-1 ISA, up to five 32-bit source operands may be required for any single instruction. Note that for relay forwarding (one of several possible forwarding strategies), initial destination operands also have to eventually be acquired. But this is not a constraint on when as AS can first make an execution request. After receiving initial source operands, ASes then receive additional operands (forwarded from previous ASes) during its residency in the execution window. This represents two different phases for getting source operands : initial operands, and subsequent operands. Each of these phases is discussed next.

2.1 Getting Initial Source Operands

Three alternatives have been proposed so far for acquiring initial source operands for an AS. Each would seem to have some merit but none has so far distinguished itself above the others so far as being better.

The first alternative is for the AS to request each of its operands (source or destination) by making requests for them on Register Backwarding buses. The AS then waits for the operands to arrive. Once all operands are present in the AS, the AS is free to request execution resources.

In the second alternative, a hybrid approach toward getting source operands is used. No backwarding request is made for the most common source operands (usually the orthogonal integer registers), but requests are made for the more rare operands such as a status register for example or an FP register (whether an FP register is considered rare or not might be debatable but that is what was done in this scheme so far). When a new column of ASes is first loaded, this signals the register filtering unit just before the column (in time-tag

ordering) to forward all of the "common" registers on forwarding buses. The operands that are requested also get forwarded as the requests are honored by either register filtering units or other ASes (time-tag ordered before the newly loaded column). Note that in this scheme, some registers may be forwarded for which no newly loaded ASes require values for. This represents a waste of bus bandwidth but would not present a problem when more bandwidth is provided than is needed. The hope in this scheme is that needed initial operands will arrive sooner at an AS since the process of requesting the most usual (common) operands was eliminated.

In the third alternative, initial values are not requested and neither are values of common registers forwarded automatically. Rather, initial values are guessed using a value predictor. This scheme is much more costly in terms of hardware resources since a value predictor represents a sizable amount of real estate. Further, due to the need for parallel dispatch of instructions to ASes, typically a separate value predictor is required for each row of the execution window (ouch).

Of course, combinations of the above techniques are possible.

2.2 Getting Subsequent Source Operands

Once an AS has its initial source operands, it competes with other ASes for execution resources. Once an AS gets to execute, it forwards its result operands. Up to three 32-bit result operands are possible in the MIPS-1 ISA. Of course, ASes in program ordered past (lower valued time-tags) may have also forwarded result operands and these are being snooped by the AS currently in view. In the present scheme, once an AS snarfs any one of its source operands from previous ASes, it initiates a request for execution again. If an AS currently requires five source operands (for example), a newly snarfed value for any one of them triggers the present AS to re-execute its instruction. Of course, this only occurs when the value snooped is different than the previous value held by the AS.

2.3 Problems With the Present Scheme

The present scheme has the merit of being simple (as far as any of this might be considered simple in the first place) and also does not impose any direct delays for requests of execution or for generating new result operands. However, there are some things about this scheme that are not as attractive as they could be. For those cases where two or more input operands get snarfed by an AS, but in different clock periods, one or more needless executions of the instruction in the AS might have occurred. Since only the results from the latest execution are retained, the results from the earlier execution might be considered wasted. In this case, we can think of one unit of execution resource as having been wasted. However, there is more wasted than just this execution resource. The result operands from both executions are forwarded to subsequent ASes. The result operands from the first execution will trigger executions for subsequent ASes that have a dependency (which may be real or not) on the forwarded operand. This forms a sort of chain reaction (following the dependency chain), but this is what is intended to maintain proper program order. The problem is that any subsequent executions based on the forwarding of the first execution results of the present AS, may also be wasted. This is so since those same subsequent executions by succeeding ASes in the dependency chain will likely execute again when the operands from the present AS's second execution gets forwarded. This represents both additionally wasted execution resources and wasted forwarding bus bandwidth. In the new scheme being proposed, some waste due to needless executions, can be eliminated.

3 Improved Speculation Scheme

The new scheme is the same as the present scheme with respect to how initial source operands are acquired by ASes. Any of the three methods discussed, as well as others, may be employed. The new scheme differs slightly from the present scheme in the handling of snarfed operands subsequent to acquiring the initial operands. In this scheme, a small counter is maintained by each AS. The counter is clocked by the system

clock. Upon snarfing an operand (its value was different than the current value), the counter is enabled to start counting. The terminal count for the counter should be some number of system clocks that is more than about two and significantly less than the average latency that instructions stay within the execution window. A good number may be about the average clocks between column shifts, plus or minus one half this or double this number.

No execution request is made by the AS immediately after snarfing an operand. Rather, a certain number of clocks (as maintained by the counter) are allowed to elapse. If the counter reaches its termination count, the AS then initiates its execution request. If a new operand is snarfed during the time the AS is waiting for counter expiration, the AS initiates its execution request immediately, using the newer operand, without waiting for counter expiration. In addition, ASes that find themselves in the column that is in the position to commit next do not follow this above rule with the counter but instead just initiates execution requests upon the arrival of each and every newly snarfed operand. This is done so as to not add any direct dead clock periods that can be attributed to a delay in the next column commitment.

Using this strategy, up to one extra operand, that is snarfed by an AS within the period of the counter duration, can be eliminated. For each snarfed operand that is eliminated, its redundant execution is also eliminated along with any of its redundant result operands. Of course, for chains of data dependent instructions, just as a chain reaction of redundant executions and forwards was created by each additionally snarfed source operand, that same chain reaction of operand forwards and executions will also be eliminated.

The idea with this strategy is to reduce contention pressure on execution resources and forwarding bus resources while not adversely affecting the residency time of instructions in the execution window. By not applying the counter expiration waiting rule for initially loaded ASes and ASes located in the next column that is schedule to commit, no dead clocks are directly added to the overall instruction residency time.

4 Other Possibilities

Other choices for how as AS waits for additional operands are possible. One such approach could be for an AS to always wait for counter expiration rather then initiating its execution request on the snarf of an operand within the counter period. This would offer the possibility of eliminating more than just one redundantly snarfed source operand and its associated execution chain reaction.

5 Conclusions

A a new scheme has been introduced that can be used to reduce the requirement for execution resources and forwarding bus bandwidth. This is accomplished through the elimination of certain redundant snarfs and associated re-executions by an AS. The new scheme does not add any clock delays to the amount of execution window residency until possible commitment. This new scheme may be useful for machine configurations may have a high AS to PE ratio where there is high contention for execution resources by the ASes. This new scheme may also be used to reduce the demand and contention for forwarding bus bandwidth. Where there is sufficient execution resources and forwarding bus bandwidth, this scheme should perform no worse than the previous speculative value handling method.