

WHAT IS HIGH PERFORMANCE COMPUTING - OUT-OF-ORDER EXECUTION: THE POST-RISC ARCHITECTURE*

Charles Severance
Kevin Dowd

This work is produced by OpenStax-CN X and licensed under the
Creative Commons Attribution License 3.0[†]

We're never satisfied with the performance level of our computing equipment and neither are the processor designers. Two-way superscalar processors were very successful around 1994. Many designs were able to execute 1.6–1.8 instructions per cycle on average, using all of the tricks described so far. As we became able to manufacture chips with an ever-increasing transistor count, it seemed that we would naturally progress to four-way and then eight-way superscalar processors. The fundamental problem we face when trying to keep four functional units busy is that it's difficult to find contiguous sets of four (or eight) instructions that can be executed in parallel. It's an easy cop-out to say, "the compiler will solve it all."

The solution to these problems that will allow these processors to effectively use four functional units per cycle and hide memory latency is *out-of-order execution* and *speculative execution*. Out-of-order execution allows a later instruction to be processed before an earlier instruction is completed. The processor is "betting" that the instruction will execute, and the processor will have the precomputed "answer" the instruction needs. In some ways, portions of the RISC design philosophy are turned inside-out in these new processors.

1 Speculative Computation

To understand the post-RISC architecture, it is important to separate the concept of computing a value for an instruction and actually executing the instruction. Let's look at a simple example:

```
LD    R10,R2(R0)    Load into R10 from memory
...
FDIV  R4,R5,R6      R4 = R5 / R6
```

*Version 1.3: Aug 25, 2010 11:29 am -0500

[†]<http://creativecommons.org/licenses/by/3.0/>

Assume that (1) we are executing the load instruction, (2) R5 and R6 are already loaded from earlier instructions, (3) it takes 30 cycles to do a floating-point divide, and (4) there are no instructions that need the divide unit between the LD and the FDIV. Why not start the divide unit *computing* the FDIV right now, storing the result in some temporary scratch area? It has nothing better to do. When or if we arrive at the FDIV, we will know the result of the calculation, copy the scratch area into R4, and the FDIV will appear to *execute* in one cycle. Sound farfetched? Not for a post-RISC processor.

The post-RISC processor must be able to speculatively compute results before the processor knows whether or not an instruction will actually execute. It accomplishes this by allowing instructions to start that will never finish and allowing later instructions to start before earlier instructions finish.

To store these instructions that are in limbo between started and finished, the post-RISC processor needs some space on the processor. This space for instructions is called the *instruction reorder buffer (IRB)*.

2 The Post-RISC Pipeline

The post-RISC processor pipeline in Figure 1 (Post-RISC pipeline) looks somewhat different from the RISC pipeline. The first two stages are still instruction fetch and decode. Decode includes branch prediction using a table that indicates the probable behavior of a branch. Once instructions are decoded and branches are predicted, the instructions are placed into the IRB to be computed as soon as possible.

Post-RISC pipeline

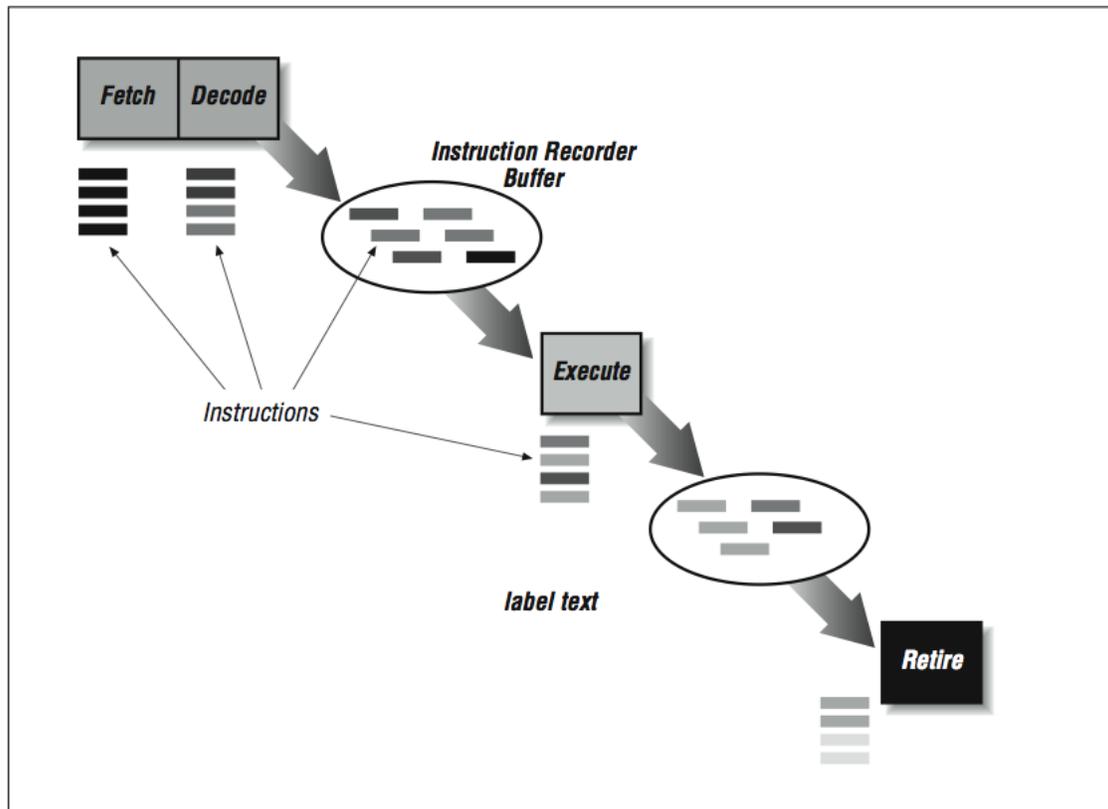


Figure 1

The IRB holds up to 60 or so instructions that are waiting to execute for one reason or another. In a sense, the fetch and decode/predict phases operate until the buffer fills up. Each time the decode unit predicts a branch, the following instructions are marked with a different indicator so they can be found easily if the prediction turns out to be wrong. Within the buffer, instructions are allowed to go to the computational units when the instruction has all of its operand values. Because the instructions are computing results without being executed, any instruction that has its input values and an available computation unit can be computed. The results of these computations are stored in extra registers not visible to the programmer called *rename registers*. The processor allocates rename registers, as they are needed for instructions being computed.

The execution units may have one or more pipeline stages, depending on the type of the instruction. This part looks very much like traditional superscalar RISC processors. Typically up to four instructions can begin computation from the IRB in any cycle, provided four instructions are available with input operands and there are sufficient computational units for those instructions.

Once the results for the instruction have been computed and stored in a rename register, the instruction must wait until the preceding instructions finish so we know that the instruction actually executes. In addition to the computed results, each instruction has flags associated with it, such as exceptions. For

example, you would not be happy if your program crashed with the following message: “Error, divide by zero. I was precomputing a divide in case you got to the instruction to save some time, but the branch was mispredicted and it turned out that you were never going to execute that divide anyway. I still had to blow you up though. No hard feelings? Signed, the post-RISC CPU.” So when a speculatively computed instruction divides by zero, the CPU must simply store that fact until it knows the instruction will execute and at that moment, the program can be legitimately crashed.

If a branch does get mispredicted, a lot of bookkeeping must occur very quickly. A message is sent to all the units to discard instructions that are part of all control flow paths beyond the incorrect branch.

Instead of calling the last phase of the pipeline “writeback,” it’s called “retire.” The retire phase is what “executes” the instructions that have already been computed. The retire phase keeps track of the instruction execution order and retires the instructions in program order, posting results from the rename registers to the actual registers and raising exceptions as necessary. Typically up to four instructions can be retired per cycle.

So the post-RISC pipeline is actually three pipelines connected by two buffers that allow instructions to be processed out of order. However, even with all of this speculative computation going on, the retire unit forces the processor to appear as a simple RISC processor with predictable execution and interrupts.