Reorder Buffer

The function of the reorder buffer is to put the instructions back in the original program order after the instructions have finished execution possibly out of order. The reorder buffer maintains an ordered list of the instructions. Instructions are added at one end of the list when they are dispatched and they are removed from the other end of the list when they are completed. In this way, instructions will be completed in the same order as they were dispatched.

The implementation of the reorder buffer is usually a circular buffer (similar to I/O buffers in operating systems) as shown in fig. 4-28b, p. 179. The dispatch stage increments the tail pointer to add instructions to the list. The complete stage increments the head pointer to remove instructions from the list. When the head pointer catches up to the tail pointer, the buffer is empty. When the pointers get to the end of the buffer, they wrap around again to the beginning of the pointer. If the tail pointer wraps around and catches the head pointer, the buffer is full.

An entry in the reorder buffer must contain enough information so that the complete stage can determine whether an instruction should complete. A typical entry might look as in fig. 4-28a, p. 179.

The Busy bit is not really required since the head (complete) and tail (dispatch) pointers are used to keep track of where “busy” entries are.

The Issued bit is probably not necessary either since complete does not care when instructions are issued.

The Finished bit is essential information for complete since instructions that have not finished cannot be allowed to complete. The finished bit must be written when an execution pipeline writes its result onto one of the forwarding busses. This requires the execution unit to have a pointer into the reorder buffer for each instruction that is finished.

The instruction address must be saved in case the instruction causes an exception. After the exception is handled, the PC will be restarted with the instruction address. Also, branch instructions may need the instruction address to determine the PC during recovery from a mispredicted branch.

The rename register number is essential for complete to know which rename register to write into the architected register. The rename register number can also be used to check the Valid bit in the rename register file to see if an instruction has finished. This would make the Finished bit unnecessary.

The reorder buffer would be a convenient place to store the logical (architected) register number also. Complete needs the architected register number to write into the architected register file.

The Speculative and Valid bits may not be necessary depending on how recovery from mispredicted branches is implemented. Speculative instructions can never get to the head of the reorder buffer because previous instructions (including branches) must complete first. Thus a branch will already be resolved and recovery initiated if needed before any (control dependent) instructions after the branch are allowed to complete. If control dependent instructions are removed from the reorder buffer during recovery, the Valid bits are unnecessary; otherwise, the control dependent instructions must be marked invalid so that they can be ignored when they reach the head of the reorder buffer.

The reorder buffer must accomplish the following operations.

1. Allocate: The dispatch stage reserves space in the reorder buffer for instructions in program order.
    a. The tail (dispatch) pointer selects a location(s) in the reorder buffer.
    b. The essential information is loaded into the reorder buffer (for example: Instruction address, rename register, architected register).
    c. The tail pointer is incremented.
2. Wait: The complete stage must wait for instructions to finish execution.
    a. All instructions at the head of the reorder buffer that are finished (or that have valid rename registers) are selected for completion.
    b. Completion stops at the first unfinished instruction.
3. Complete: Finished instructions are allowed to write results in order into the architected registers.
    a. copy rename register into architected register.
    b. deallocate rename register by setting rename and architected register Busy = 0. Deallocate reorder buffer by incrementing the head pointer.