5.4.6.4. Multiple Prefetching

In this case, the processor fetches instructions from both possible paths. Once the branch decision is made, the unwanted path is abandoned. By prefetching both possible paths, the fetch penalty is avoided in the case of an incorrect prediction.

To fetch both paths, two buffers are employed by the pipeline. In normal execution, the first buffer is loaded with instructions from the next sequential address of the branch instruction. If a branch occurs, the contents of the first buffer are invalidated, and the secondary buffer, which has been loaded with instructions from the target address of the branch instruction, is used as the primary buffer.

This double buffering scheme ensures a constant flow of instructions and data to the pipeline and reduces the time delays caused by the draining and refilling of the pipeline. However, some amount of performance degradation is unavoidable any time the pipeline is drained.

In summary, each of the preceding techniques reduces the degradation of pipeline throughput. However, the choice of any of these techniques for a particular design depends on factors such as throughput requirements and cost constraints. In practice, due to these factors, usually a mixture of these techniques is implemented on a single processor.

5.4.7. The Intel Architecture Processors Pipeline

Figure 5.17 shows a block diagram of the Intel Architecture processors pipeline. The processing units shown in the figure represent stages of the pipeline: the fetch/decode unit, the dispatch/execute unit, the retire unit, and the instruction pool. Instructions and data are supplied to these units through the bus interface unit.

5.4.7.1. Fetch/Decode Unit

The fetch/decode unit reads a stream of instructions from the L1 instruction cache memory and decodes them into a series of microoperations. Microoperations are primitive instructions that are executed by the processor's parallel execution units. The stream of microoperations, which is still in the order of the original instruction stream, is then sent to the instruction pool.

The instruction fetch unit fetches one 32-byte cache line in each clock cycle from the instruction cache memory. It marks the beginning and end of the instructions in the cache memory lines and transmits 16 aligned bytes to the decoder. The instruction fetch unit computes the instruction pointer, based on inputs from the branch target buffer, the exception/interrupt status, and branch-prediction indications from the integer execution units. The most important part of this process is the branch prediction performed by the branch target buffer. This 512-entry buffer looks 20 to 30 instructions ahead of the program counter. Within this instruction window

there may be numerous branches, procedure calls, and returns that must be correctly predicted.

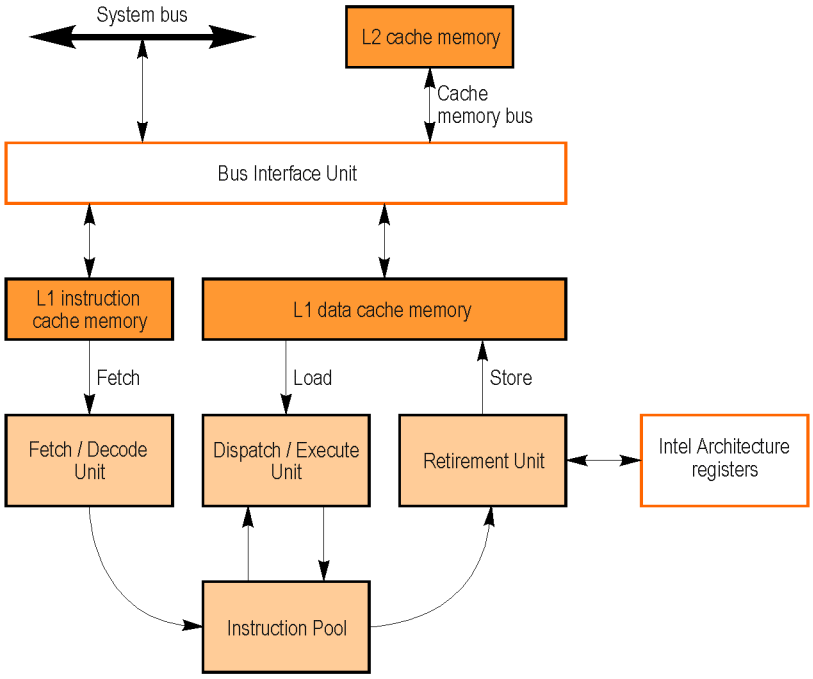

Figure 5.17. Conceptual view of the Intel Architecture processors pipeline.

The instruction decoder contains three parallel decoders: two simple instruction decoders and one complex instruction decoder. Each decoder converts an instruction into one or more triadic microoperations, with two logical sources and one logical destination. The instruction decoders also handle the decoding of instruction prefixes and looping operations. The decoders can generate up to six microoperations per clock cycle (one each for the simple instruction decoders and four for the complex instruction decoder).

The processor's register set can cause execution stalls due to register dependencies. To solve this problem, the processor provides 40 internal, general-purpose registers, which are used for the actual computations. These registers can hold both integer and floating-point values. To allocate the internal registers, the stream of microoperations from the instruction decoder is sent to the register alias table, where references to the logical registers are converted into physical register references.

In the final step of the decoding process, the allocator in the register alias table adds status bits and flags to the microoperations to prepare them for out-of-order execution, and sends the resulting microoperations to the instruction pool.

5.4.7.2. Instruction Pool

Prior to entering the instruction pool (also known as reorder buffer), the microoperation stream is in the same order as the instruction stream that was sent to the instruction decoder. The reorder buffer is an array of content-addressable memory, organized as 40 registers. It contains microoperations that are waiting to be executed, as well as those that have already been executed but not yet affected to machine state. The dispatch/execute unit can execute microoperations from the reorder buffer in any order.

5.4.7.3. Dispatch/Execute Unit

The dispatch/execute unit schedules and executes the microoperations stored in the reorder buffer taking into account data dependencies and resource availability, and temporarily stores the results of these speculative executions. The scheduling and dispatching of microoperations from the reorder buffer is handled by the reservation station. It continuously scans the reorder buffer to find microoperations that are ready to be executed (that is, all the source operands are available) and dispatches them to the available execution units. The results of a microoperation execution are returned to the reorder buffer and stored along with the microoperation until it is retired. The reservation station has five ports, and the multiple resources are accessed as shown in Figure 5.18.

The Intel Architecture processors can schedule for execution up to 5 microoperations per clock cycle, one to each resource port, but a sustained rate of 3 microoperations per clock cycle is more common. The scheduling and dispatching process supports out-of-order execution, where microoperations are dispatched to the execution units strictly according to data-flow constraints and execution resource availability, without regard to the original ordering of the instructions. When two or more microoperations of the same type (for example, floating-point operations) are available at the same time, they are executed in a FIFO order.

Execution of microoperations is handled by two integer execution units, a floating-point execution unit, two MMX (*Multimedia Extensions*) execution units, and one memory-interface unit, containing a load unit and a store unit. There is also a branch execution unit, which can handle branch microoperations. This unit has the ability to detect branch mispredictions and signal the branch target buffer to restart the pipeline. This operation is handled as follows. The instruction decoder tags each branch microoperation with both branch destination addresses (the target address and the sequential address). When the branch execution unit executes the branch microoperation, it is able to determine the destination chosen. If the predicted branch is taken, then speculatively executed microoperations are marked usable and execution continues along the predicted path. If the predicted branch is not taken, the branch execution unit changes the status of all the microoperations on the target path to remove them from the instruction pool. It then provides the proper branch destination

to the branch target buffer, which in turn restarts the pipeline from the new target address.

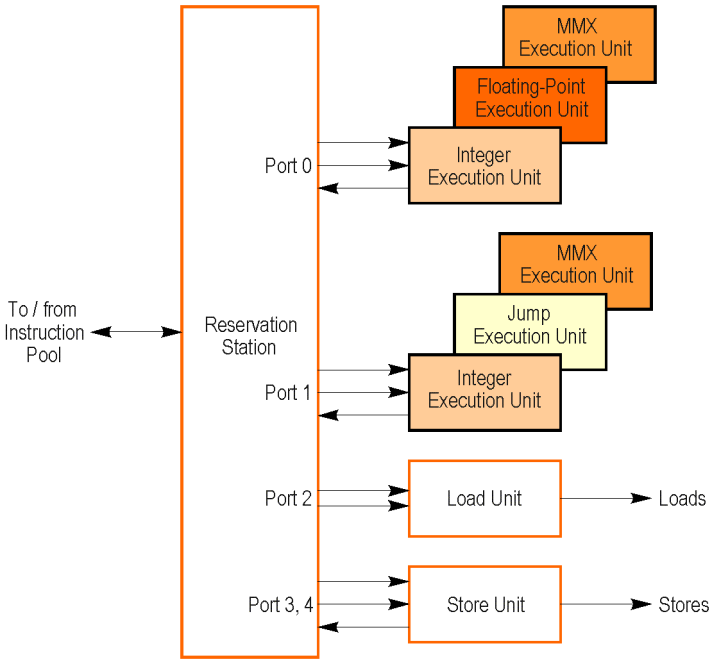

Figure 5.18. Intel Architecture processors dispatch/execute unit.

The memory interface unit handles load and store microoperations. A load access only needs to specify the memory address, so it can be encoded in one microoperation. A store access needs to specify both an address and the data to be written, so it is encoded in two microoperations. The part of the memory interface unit that handles store operations has two ports, allowing it to process the address and the data microoperations in parallel. The memory interface unit can thus execute in parallel a load and a store operation in one clock cycle.

The MMX execution units use the SIMD (*Single Instruction, Multiple Data*) technique to deliver superior performance for data types used in multimedia and communications applications. The MMX extensions (which include eight new 64-bit registers, four data types, and a set of 57 general-purpose integer instructions) accelerate the execution of applications such as motion video, graphics combined with video images, 2D and 3D graphics, image processing, audio synthesis, speech synthesis and compression, telephony, and video conferencing, which typically perform repetitive operations on large arrays of simple data elements.

The Intel Pentium III and Pentium 4 processors also contain a separate unit for streaming SIMD extensions. These extensions include eight new 128-bit floating-point registers, a packed single-precision floating-point data type, and 70 new instructions for integer and floating-point data. Applications that benefit from the streaming

SIMD extensions include advanced image processing, high quality audio, MPEG-2 video, simultaneous MPEG-2 encoding and decoding, 3D graphics, and speech recognition.

5.4.7.4. Retirement Unit

The retirement unit writes the results of speculatively executed microoperations into the user-visible registers and removes the microoperations from the reorder buffer. Like the reservation station, the retirement unit continuously checks the status of microoperations in the reorder buffer, looking for ones that have been executed and no longer have any dependencies with other microoperations in the instruction pool. It then retires completed microoperations in their original program order.

The retirement unit can retire three microoperations per clock cycle. In retiring a microoperation, it writes the results to the processor's register file and/or memory. After the results have been written, the microoperation is removed from the reorder buffer.

5.4.7.5. Bus Interface Unit

The memory subsystem for the Intel Architecture processors consists of main system memory, the primary cache memory (L1), and the secondary cache memory (L2). The bus interface unit accesses system memory through the external system bus. This 64-bit bus is a transaction-oriented bus, meaning that each bus access is treated as consisting of separate request and response operations. While the bus interface unit is waiting for a response to one bus request, it can issue numerous additional requests.

The bus interface unit accesses the L2 cache memory through a 64-bit cache memory bus. This bus is also transaction-oriented, supporting up to four concurrent cache memory accesses, and operates at the full clock speed of the processor. Access to the L1 cache memories (for instructions and data) is through internal buses, also at full clock speed. Coherency between the cache memories and system memory is maintained using the MESI (*Modified, Exclusive, Shared, Invalid*) protocol.

Memory requests from the processor's execution units go through the memory interface unit and the memory order buffer. The L1 data cache memory automatically forwards a cache miss to the L2 cache memory, and then, if necessary, the bus interface unit forwards an L2 cache miss to system memory. Memory requests to the L2 cache memory or system memory go through the memory order buffer, which functions as a scheduling and dispatch station. This buffer keeps track of all memory requests and is able to reorder some requests to prevent blocking conditions and to improve throughput. For example, the memory order buffer allows speculative load operations. Store operations are always executed in order, and speculative store operations are not allowed.