CHAPTER 2
PENTIUM® PRO PROCESSOR ARCHITECTURE OVERVIEW

The Pentium Pro processor has a decoupled, 12-stage, superpipelined implementation, trading less work per pipestage for more stages. The Pentium Pro processor also has a pipestage time 33 percent less than the Pentium processor, which helps achieve a higer clock rate on any given process.

The approach used by the Pentium Pro processor removes the constraint of linear instruction sequencing between the traditional "fetch" and "execute" phases, and opens up a wide instruction window using an instruction pool. This approach allows the "execute" phase of the Pentium Pro processor to have much more visibility into the program's instruction stream so that better scheduling may take place. It requires the instruction "fetch/decode" phase of the Pentium Pro processor to be much more intelligent in terms of predicting program flow. Optimized scheduling requires the fundamental "execute" phase to be replaced by decoupled "dispatch/execute" and "retire" phases. This allows instructions to be started in any order but always be completed in the original program order. The Pentium Pro processor is implemented as three independent engines coupled with an instruction pool as shown in Figure 2-1.

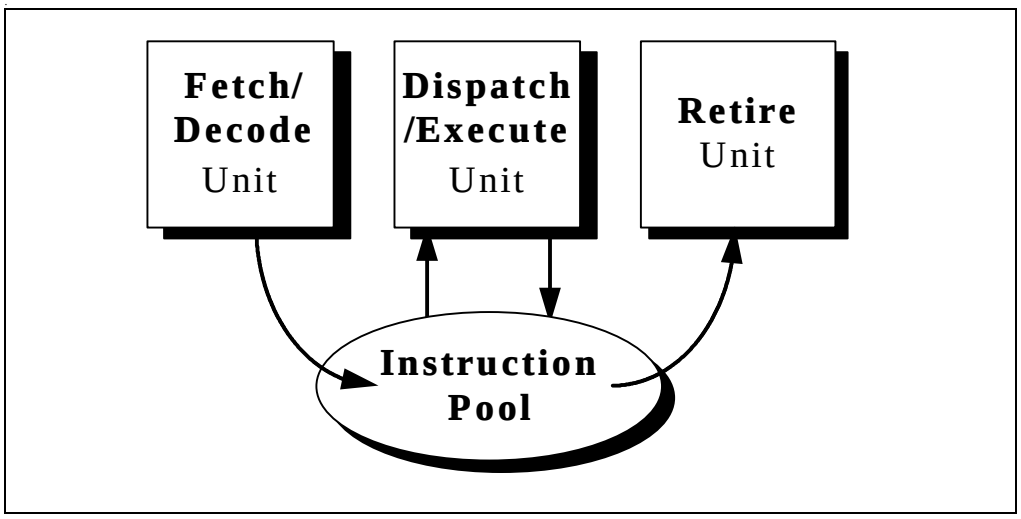

Figure 2-1. Three Engines Communicating Using an Instruction Pool

2.1. FULL CORE UTILIZATION

The three independent-engine approach was taken to more fully utilize the CPU core. Consider the code fragment in Figure 2-2:

```
r1 <= mem [r0] /* Instruction 1 */
r2 <= r1 + r2 /* Instruction 2 */
r5 <= r5 + 1 /* Instruction 3 */
r6 <= r6 - r3 /* Instruction 4 */
```

Figure 2-2. A Typical Code Fragment

The first instruction in this example is a load of r1 that, at run time, causes a cache miss. A traditional CPU core must wait for its bus interface unit to read this data from main memory and return it before moving on to instruction 2. This CPU stalls while waiting for this data and is thus being under-utilized.

To avoid this memory latency problem, the Pentium Pro processor "looks-ahead" into its instruction pool at subsequent instructions and will do useful work rather than be stalled. In the example in Figure 2-2, instruction 2 is not executable since it depends upon the result of instruction 1; however both instructions 3 and 4 are executable. The Pentium Pro processor executes instructions 3 and 4 out-of-order. The results of this out-of-order execution can not be committed to permanent machine state (i.e., the programmer-visible registers) immediately since the original program order must be maintained. The results are instead stored back in the instruction pool awaiting in-order retirement. The core executes instructions depending upon their readiness to execute, and not on their original program order, and is therefore a true dataflow engine. This approach has the side effect that instructions are typically executed out-of-order.

The cache miss on instruction 1 will take many internal clocks, so the Pentium Pro processor core continues to look ahead for other instructions that could be speculatively executed, and is typically looking 20 to 30 instructions in front of the instruction pointer. Within this 20 to 30 instruction window there will be, on average, five branches that the fetch/decode unit must correctly predict if the dispatch/execute unit is to do useful work. The sparse register set of an Intel Architecture (IA) processor will create many false dependencies on registers so the dispatch/execute unit will rename the IA registers into a larger register set to enable additional forward progress. The retire unit owns the programmer's IA register set and results are only committed to permanent machine state in these registers when it removes completed instructions from the pool in original program order.

Dynamic Execution technology can be summarized as optimally adjusting instruction execution by predicting program flow, having the ability to speculatively execute instructions in any order, and then analyzing the program's dataflow graph to choose the best order to execute the instructions.

2.2. THE PENTIUM® PRO PROCESSOR PIPELINE

In order to get a closer look at how the Pentium Pro processor implements Dynamic Execution, Figure 2-3 shows a block diagram including cache and memory interfaces. The “Units” shown in Figure 2-3 represent stages of the Pentium Pro processor pipeline.

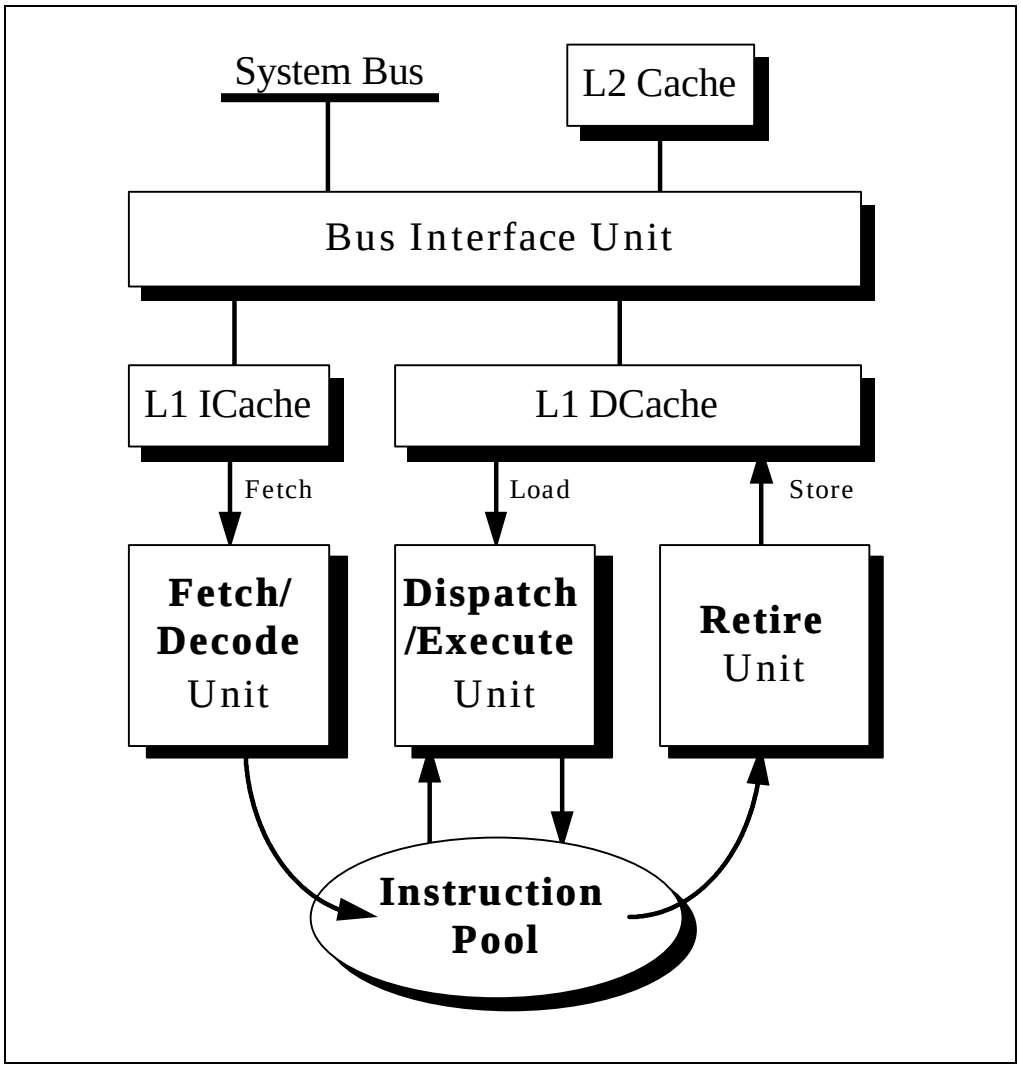

Figure 2-3. The Three Core Engines Interface with Memory via Unified Caches

- The FETCH/DECODE unit: An in-order unit that takes as input the user program instruction stream from the instruction cache, and decodes them into a series of micro-operations (µops) that represent the dataflow of that instruction stream. The pre-fetch is speculative.
- The DISPATCH/EXECUTE unit: An out-of-order unit that accepts the dataflow stream, schedules execution of the µops subject to data dependencies and resource availability and temporarily stores the results of these speculative executions.
- The RETIRE unit: An in-order unit that knows how and when to commit ("retire") the temporary, speculative results to permanent architectural state.
- The BUS INTERFACE unit: A partially ordered unit responsible for connecting the three internal units to the real world. The bus interface unit communicates directly with the L2 (second level) cache supporting up to four concurrent cache accesses. The bus interface unit also controls a transaction bus, with MESI snooping protocol, to system memory.

2.2.1. The Fetch/Decode Unit

Figure 2-4 shows a more detailed view of the Fetch/Decode Unit.

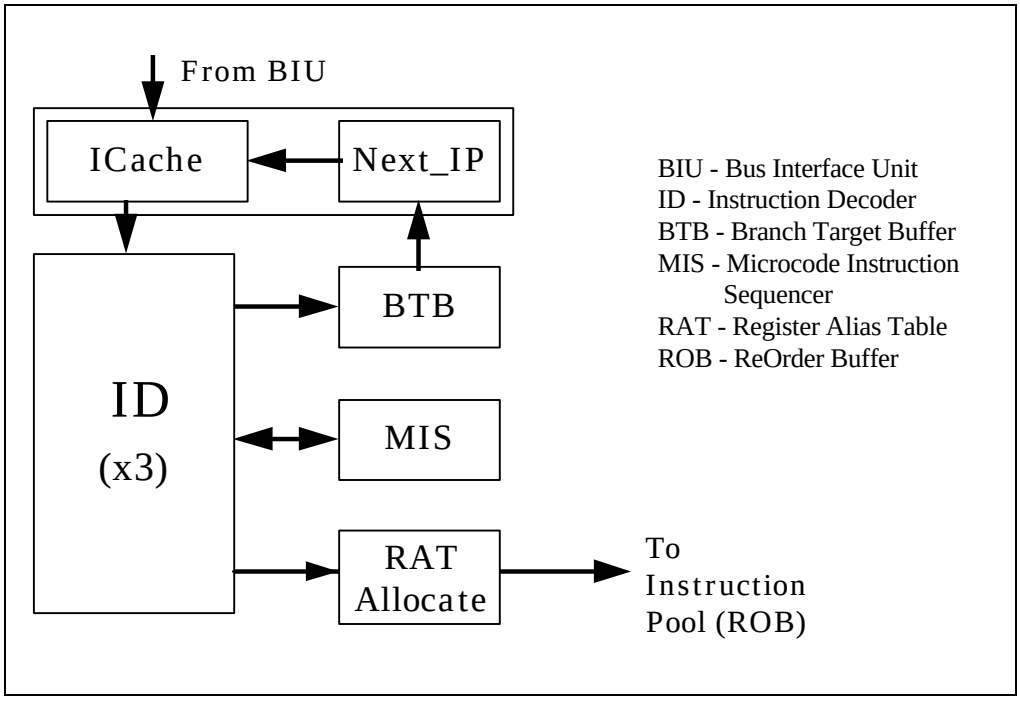

Figure 2-4. Inside the Fetch/Decode Unit

The ICache is a local instruction cache. The Next_IP unit provides the ICache index, based on inputs from the Branch Target Buffer (BTB), trap/interrupt status, and branch-misprediction indications from the integer execution section.

The ICache fetches the cache line corresponding to the index from the Next_IP, and the next line, and presents 16 aligned bytes to the decoder. The prefetched bytes are rotated so that they are justified for the instruction decoders (ID). The beginning and end of the IA instructions are marked.

Three parallel decoders accept this stream of marked bytes, and proceed to find and decode the IA instructions contained therein. The decoder converts the IA instructions into triadic µops (two logical sources, one logical destination per µop). Most IA instructions are converted directly into single µops, some instructions are decoded into one-to-four µops and the complex instructions require microcode (the box labeled MIS in Figure 2-4). This microcode is just a set of preprogrammed sequences of normal µops. The µops are queued, and sent to the Register Alias Table (RAT) unit, where the logical IA-based register references are converted into Pentium Pro processor physical register references, and to the Allocator stage, which adds status information to the µops and enters them into the instruction pool. The instruction pool is implemented as an array of Content Addressable Memory called the ReOrder Buffer (ROB).

This is the end of the in-order pipe.

2.2.2. The Dispatch/Execute Unit

The dispatch unit selects µops from the instruction pool depending upon their status. If the status indicates that a µop has all of its operands then the dispatch unit checks to see if the execution resource needed by that µop is also available. If both are true, the Reservation Station removes that µop and sends it to the resource where it is executed. The results of the µop are later returned to the pool. There are five ports on the Reservation Station, and the multiple resources are accessed as shown in Figure 2-5.

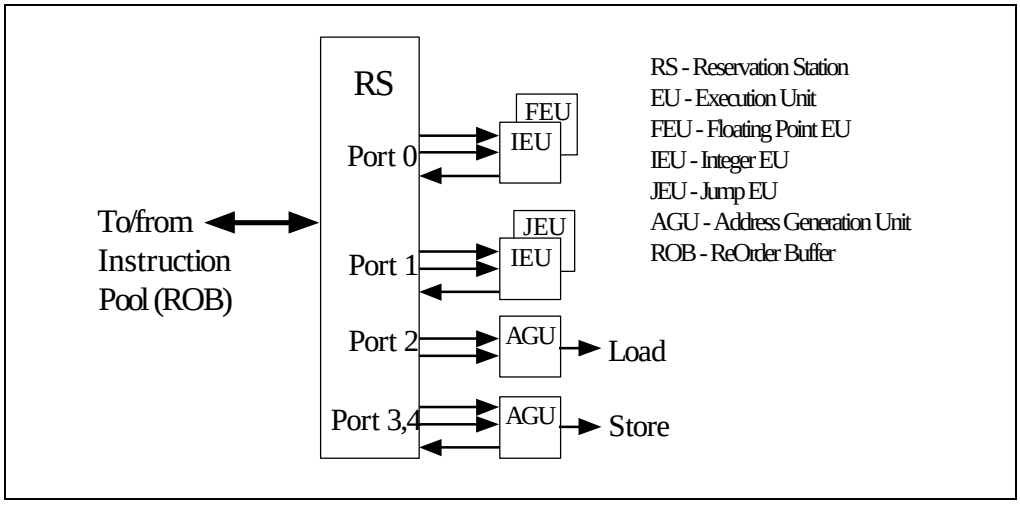

Figure 2-5. Inside the Dispatch/Execute Unit

The Pentium Pro processor can schedule at a peak rate of 5 µops per clock, one to each resource port, but a sustained rate of 3 µops per clock is typical. The activity of this scheduling process is the out-of-order process; µops are dispatched to the execution resources strictly according to dataflow constraints and resource availability, without regard to the original ordering of the program.

Note that the actual algorithm employed by this execution-scheduling process is vitally important to performance. If only one µop per resource becomes data-ready per clock cycle, then there is no choice. But if several are available, it must choose. The Pentium Pro processor uses a pseudo FIFO scheduling algorithm favoring back-to-back µops.

Note that many of the µops are branches. The Branch Target Buffer will correctly predict most of these branches but it can't correctly predict them all. Consider a BTB that is correctly predicting the backward branch at the bottom of a loop; eventually that loop is going to terminate, and when it does, that branch will be mispredicted. Branch µops are tagged (in the in-order pipeline) with their fall-through address and the destination that was predicted for them. When the branch executes, what the branch actually did is compared against what the prediction hardware said it would do. If those coincide, then the branch eventually retires, and most of the speculatively executed work behind it in the instruction pool is good.

But if they do not coincide, then the Jump Execution Unit (JEU) changes the status of all of the µops behind the branch to remove them from the instruction pool. In that case the proper branch destination is provided to the BTB which restarts the whole pipeline from the new target address.

2.2.3. The Retire Unit

Figure 2-6 shows a more detailed view of the Retire Unit.

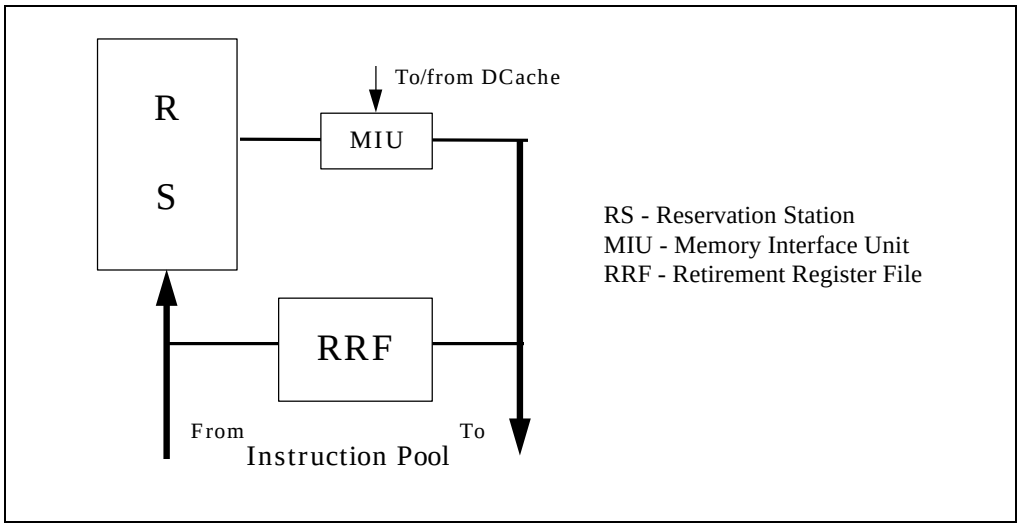

Figure 2-6. Inside the Retire Unit

The retire unit is also checking the status of µops in the instruction pool. It is looking for µops that have executed and can be removed from the pool. Once removed, the original architectural target of the µops is written as per the original IA instruction. The retirement unit must not only notice which µops are complete, it must also re-impose the original program order on them. It must also do this in the face of interrupts, traps, faults, breakpoints and mispredictions.

The retirement unit must first read the instruction pool to find the potential candidates for retirement and determine which of these candidates are next in the original program order. Then it writes the results of this cycle's retirements to both the Instruction Pool and the Retirement Register File (RRF). The retirement unit is capable of retiring 3 µops per clock.

2.2.4. The Bus Interface Unit

Figure 2-7 shows a more detailed view of the Bus Interface Unit.

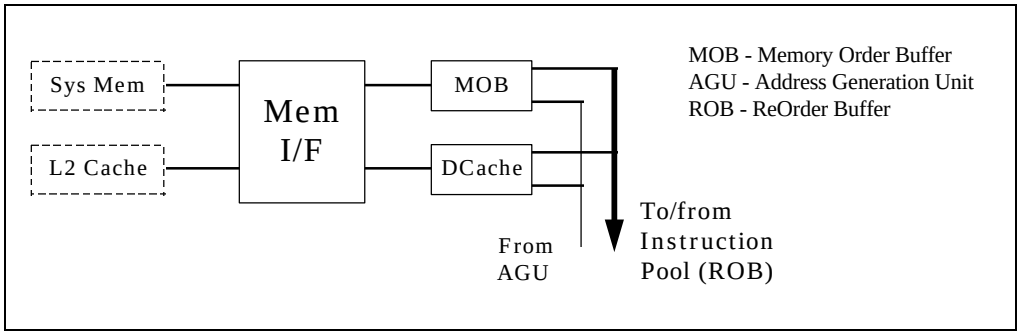

Figure 2-7. Inside the Bus Interface Unit

There are two types of memory access: loads and stores. Loads only need to specify the memory address to be accessed, the width of the data being retrieved, and the destination register. Loads are encoded into a single µop.

Stores need to provide a memory address, a data width, and the data to be written. Stores therefore require two µops, one to generate the address, and one to generate the data. These µops must later re-combine for the store to complete.

Stores are never performed speculatively since there is no transparent way to undo them. Stores are also never re-ordered among themselves. A store is dispatched only when both the address and the data are available and there are no older stores awaiting dispatch.

A study of the importance of memory access reordering concluded:

- Stores must be constrained from passing other stores, for only a small impact on performance.
- Stores can be constrained from passing loads, for an inconsequential performance loss.
- Constraining loads from passing other loads or stores has a significant impact on performance.

The Memory Order Buffer (MOB) allows loads to pass other loads and stores by acting like a reservation station and re-order buffer. It holds suspended loads and stores and re-dispatches them when a blocking condition (dependency or resource) disappears.

2.3. ARCHITECTURE SUMMARY

Dynamic Execution is this combination of improved branch prediction, speculative execution and data flow analysis that enables the Pentium Pro processor to deliver its superior performance.