**The Allocator**

 The out-of-order execution engine has several buffers to perform its re-ordering, tracking, and sequencing operations. The Allocator logic allocates many of the key machine buffers needed by each uop to execute. If a needed resource, such as a register file entry, is unavailable for one of the three uops coming to the Allocator this clock cycle, the Allocator will stall this part of the machine. When the resources become available the Allocator assigns them to the requesting uops and allows these satisfied uops to flow down the pipeline to be executed. The Allocator allocates a Reorder Buffer (ROB) entry, which tracks the completion status of one of the 126 uops that could be in flight simultaneously in the machine. The Allocator also allocates one of the 128 integer or floating-point register entries for the result data value of the uop, and possibly a load or store buffer used to track one of the 48 loads or 24 stores in the machine pipeline. In addition, the Allocator allocates an entry in one of the two uop queues in front of the instruction schedulers.

**Register Renaming**

 The register renaming logic renames the logical IA-32 registers such as EAX onto the processors 128-entry physical register file. This allows the small, 8-entry, architecturally defined IA-32 register file to be dynamically expanded to use the 128 physical registers in the Pentium 4 processor. This renaming process removes false conflicts caused by multiple instructions creating their simultaneous but unique versions of a register such as EAX. There could be dozens of unique instances of EAX in the machine pipeline at one time. The renaming logic remembers the most current version of each register, such as EAX, in the Register Alias Table (RAT) so that a new instruction coming down the pipeline can know where to get the correct current instance of each of its input operand registers.

 As shown in Figure 5 the NetBurst microarchitecture allocates and renames the registers somewhat differently than the P6 microarchitecture. On the left of Figure 5, the P6 scheme is shown. It allocates the data result registers and the ROB entries as a single, wide entity with a data and a status field.

The ROB data field is used to store the data result value of the uop, and the ROB status field is used to track the status of the uop as it is executing in the machine. These ROB entries are allocated and deallocated sequentially and are pointed to by a sequence number that indicates the relative age of these entries.

 Upon retirement, the result data is physically copied from the ROB data result field into the separate Retirement Register File (RRF). The RAT points to the current version of each of the architectural registers such as EAX. This current register could be in the ROB or in the RRF. The NetBurst microarchitecture allocation scheme is shown on the right of Figure 5. It allocates the ROB entries and the result data Register File (RF) entries separately.

 The ROB entries, which track uop status, consist only of the status field and are allocated and deallocated sequentially. A sequence number assigned to each uop indicates its relative age. The sequence number points to the uop’s entry in the ROB array, which is similar to the P6 microarchitecture. The Register File entry is allocated from a list of available registers in the 128-entry RF–not sequentially like the ROB entries. Upon retirement, no result data values are actually moved from one physical structure to another.



**Figure 5: Pentium® III vs. Pentium® 4 processor register allocation**

**Uop Scheduling**

 The uop schedulers determine when a uop is ready to execute by tracking its input register operands. This is the heart of the out-of-order execution engine. The uop schedulers are what allow the instructions to be reordered to execute as soon as they are ready, while still maintaining the correct dependencies from the original program. The NetBurst microarchitecture has two sets of structures to aid in uop scheduling: the uop queues and the actual uop schedulers.

 There are two uop queues–one for memory operations (loads and stores) and one for non-memory operations. Each of these queues stores the uops in strict FIFO (firstin, first-out) order with respect to the uops in its own queue, but each queue is allowed to be read out-of-order with respect to the other queue. This allows the dynamic out-of-order scheduling window to be larger than just having the uop schedulers do all the reordering work.

 There are several individual uop schedulers that are used to schedule different types of uops for the various execution units on the Pentium 4 processor as shown in Figure 6. These schedulers determine when uops are ready to execute based on the readiness of their dependent input register operand sources and the availability of the execution resources the uops need to complete their operation. These schedulers are tied to four different dispatch ports. There are two execution unit dispatch ports labeled port 0 and port 1 in Figure 6.



**Figure 6: Dispatch ports in the Pentium® 4 processor**

 These ports are fast: they can dispatch up to two operations each main processor clock cycle. Multiple schedulers share each of these two dispatch ports. The fast ALU schedulers can schedule on each half of the main clock cycle while the other schedulers can only schedule once per main processor clock cycle.

 They arbitrate for the dispatch port when multiple schedulers have ready operations at once. There is also a load and a store dispatch port that can dispatch a ready load and store each clock cycle. Collectively, these uop dispatch ports can dispatch up to six uops each main clock cycle. This dispatch bandwidth exceeds the front end and retirement bandwidth, of three uops per clock, to allow for peak bursts of greater than 3 uops per clock and to allow higher flexibility in issuing uops to different dispatch ports. Figure 6 also shows the types of operations that can be dispatched to each port each clock cycle.

**4.3- Integer and Floating-Point Execution Units**

 The execution units are where the instructions are actually executed. The execution units are designed to optimize overall performance by handling the most common cases as fast as possible. There are several different execution units in the NetBurst microarchitecture. The units used to execute integer operations include the low-latency integer ALUs, the complex integer instruction unit, the load and store address generation units, and the L1 data cache.

 Floating-Point (x87), MMX, SSE (Streaming SIMD Extension), and SSE2 (Streaming SIMD Extension 2) operations are executed by the two floating-point execution blocks. MMX instructions are 64-bit packed integer SIMD operations that operate on 8, 16, or 32-bit operands. The SSE instructions are 128-bit packed IEEE single-precision floating-point operations. The Pentium 4 processor adds new forms of 128-bit SIMD instructions called SSE2. The SSE2 instructions support 128-bit packed IEEE double-precision SIMD floating-point operations and 128-bit packed integer SIMD operations. The packed integer operations support 8, 16, 32, and 64- bit operands.

 The Integer and floating-point register files sit between the schedulers and the execution units. There is a separate 128-entry register file for both the integer and the floating-point/SSE operations. Each register file also has a multi-clock bypass network that bypasses or forwards just-completed results, which have not yet been written into the register file, to the new dependent uops. This multi-clock bypass network is needed because of the very high frequency of the design.

**Low Latency Integer ALU**

 The Pentium 4 processor execution units are designed to optimize overall performance by handling the most common cases as fast as possible. The Pentium 4 processor can do fully dependent ALU operations at twice the main clock rate. The ALU-bypass loop is a key closed loop in the processor pipeline. Approximately 60- 70% of all uops in typical integer programs use this key integer ALU loop. Executing these operations at ½ the latency of the main clock helps speed up program execution for most programs. Doing the ALU operations in one half a clock cycle does not buy a 2x performance increase, but it does improve the performance for most integer applications.

 This high-speed ALU core is kept as small as possible to minimize the metal length and loading. Only the essential hardware necessary to perform the frequent ALU operations is included in this high-speed ALU execution loop. Functions that are not used very frequently, for most integer programs, are not put in this key low-latency ALU loop but are put elsewhere. Some examples of integer execution hardware put elsewhere are the multiplier, shifts, flag logic, and branch processing.

 The processor does ALU operations with an effective latency of one-half of a clock cycle. It does this operation in a sequence of three fast clock cycles (the fast clock runs at 2x the main clock rate) as shown in Figure 7. In the first fast clock cycle, the low order 16-bits are computed and are immediately available to feed the low 16-bits of a dependent operation the very next fast clock cycle. The high-order 16 bits are processed in the next fast cycle, using the carry out just generated by the low 16-bit operation. This upper 16-bit result will be available to the next dependent operation exactly when needed. This is called a staggered add. The ALU flags are processed in the third fast cycle. This staggered add means that only a 16-bit adder and its input MUXes need to be completed in a fast clock cycle. The low order 16 bits are needed at one time in order to begin the access of the L1 data cache when used as an address input.

****

**Figure 7: Staggered ALU add**

**Complex Integer Operations**

 The simple, very frequent ALU operations go to the high speed integer ALU execution units described above. Integer operations that are more complex go to separate hardware for completion. Most integer shift or rotate operations go to the complex integer dispatch port. These shift operations have a latency of four clocks. Integer multiply and divide operations also have a long latency. Typical forms of multiply and divide have a latency of about 14 and 60 clocks, respectively.

**Low Latency Level 1 (L1) Data Cache**

 The Level 1 (L1) data cache is an 8K-byte cache that is used for both integer and floating-point/SSE loads and stores. It is organized as a 4-way set-associative cache that has 64 bytes per cache line. It is a write-through cache, which means that writes to it are always copied into the L2 cache. It can do one load and one store per clock cycle. The latency of load operations is a key aspect of processor performance. This is especially true for IA-32 programs that have a lot of loads and stores because of the limited number of registers in the instruction set.

 The NetBurst microarchitecture optimizes for the lowest overall load access latency with a small, very low latency 8K byte cache backed up by a large, high-bandwidth second-level cache with medium latency. For most IA-32 programs this configuration of a small, but very low latency, L1 data cache followed by a large medium-latency L2 cache gives lower net load-access latency and therefore higher performance than a bigger, slower L1 cache. The L1 data cache operates with a 2-clock load-use latency for integer loads and a 6-clock load-use latency for floating point/ SSE loads.

 This 2-clock load latency is hard to achieve with the very high clock rates of the Pentium 4 processor. This cache uses new access algorithms to enable this very low load access latency. The new algorithm leverages the fact that almost all accesses hit the first-level data cache and the data TLB (DTLB).

 At this high frequency and with this deep machine pipeline, the distance in clocks, from the load scheduler to execution, is longer than the load execution latency itself. The uop schedulers dispatch dependent operations before the parent load has finished executing. In most cases, the scheduler assumes that the load will hit the L1 data cache.

 If the load misses the L1 data cache, there will be dependent operations in flight in the pipeline. These dependent operations that have left the scheduler will get temporarily incorrect data. This is a form of data speculation. Using a mechanism known as *replay*, logic tracks and re-executes instructions that use incorrect data. Only the dependent operations are replayed: the independent ones are allowed to complete.

 There can be up to four outstanding load misses from the L1 data cache pending at any one time in the memory subsystem.

**Store-to-Load Forwarding**

 In an out-of-order-execution processor, stores are not allowed to be committed to permanent machine state (the L1 data cache, etc.) until after the store has retired. Waiting until retirement means that all other preceding operations have completely finished. All faults, interrupts, mispredicted branches, etc. must have been signaled beforehand to make sure this store is safe to perform. With the very deep pipeline of the Pentium 4 processor it takes many clock cycles for a store to make it to retirement. Also, stores that are at retirement often have to wait for previous stores to complete their update of the data cache. This machine can have up to 24 stores in the pipeline at a time.

 Sometimes many of them have retired but have not yet committed their state into the L1 data cache. Other stores may have completed, but have not yet retired, so their results are also not yet in the L1 data cache. Often loads must use the result of one of these pending stores, especially for IA-32 programs, due to the limited number of registers available. To enable this use of pending stores, modern out-of-order execution processors have a pending store buffer that allows loads to use the pending store results before the stores have been written into the L1 data cache. This process is called store-to-load forwarding.

 To make this store-to-load-forwarding process efficient, this pending store buffer is optimized to allow efficient and quick forwarding of data to dependent loads from the pending stores. The Pentium 4 processor has a 24-entry store-forwarding buffer to match the number of stores that can be in flight at once. This forwarding is allowed if a load hits the same address as a proceeding, completed, pending store that is still in the store-forwarding buffer.

 The load must also be the same size or smaller than the pending store and have the same beginning physical address as the store, for the forwarding to take place. This is by far the most common forwarding case. If the bytes requested by a load only partially overlap a pending store or need to have some bytes come simultaneously from more than one pending store, this store-to-load forwarding is not allowed. The load must get its data from the cache and cannot complete until the store has committed its state to the cache.

 This disallowed store-to-load forwarding case can be quite costly, in terms of performance loss, if it happens very often. When it occurs, it tends to happen on older P5-core optimized applications that have not been optimized for modern, out-of-order execution microarchitectures. The newer versions of the IA-32 compilers remove most or all of these bad store-to-load forwarding cases but they are still found in many old legacy P5 optimized applications and benchmarks. This bad store-forwarding case is a big performance issue for P6-based processors and other modern processors, but due to the even deeper pipeline of the Pentium 4 processor, these cases are even more costly in performance.

**FP/SSE Execution Units**

 The Floating-Point (FP) execution cluster of the Pentium 4 processor is where the floating-point, MMX, SSE, and SSE2 instructions are executed. These instructions typically have operands from 64 to 128 bits in width. The FP/SSE register file has 128 entries and each register is 128 bits wide. This execution cluster has two 128-bit execution ports that can each begin a new operation every clock cycle. One execution port is for 128-bit general execution and one is for 128-bit register-to-register moves and memory stores. The FP/SSE engine can also complete a full 128-bit load each clock cycle.

 Early in the development cycle of the Pentium 4 processor, we had two full FP/SSE execution units, but this cost a lot of hardware and did not buy very much performance for most FP/SSE applications. Instead, we optimized the cost/performance tradeoff with a simple second port that does FP/SSE moves and FP/SSE store data primitives. This tradeoff was shown to buy most of the performance of a second full-featured port with much less die size and power cost.

 Many FP/multi-media applications have a fairly balanced set of multiplies and adds. The machine can usually keep busy interleaving a multiply and an add every two clock cycles at much less cost than fully pipelining all the FP/SSE execution hardware. In the Pentium 4 processor, the FP adder can execute one Extended-Precision (EP) addition, one Double-Precision (DP) addition, or two Single-Precision (SP) additions every clock cycle. This allows it to complete a 128-bit SSE/SSE2 packed SP or DP add uop every two clock cycles. The FP multiplier can execute either one EP multiply every two clocks, or it can execute one DP multiply or two SP multiplies every clock. This allows it to complete a 128-bit IEEE SSE/SSE2 packed SP or DP multiply uop every two clock cycles giving a peak 6 GFLOPS for single precision or 3 GFLOPS for double precision floating-point at 1.5GHz.

 Many multi-media applications interleave adds, multiplies, and pack/unpack/shuffle operations. For integer SIMD operations, which are the 64-bit wide MMX or 128-bit wide SSE2 instructions, there are three execution units that can run in parallel. The SIMD integer ALU execution hardware can process 64 SIMD integer bits per clock cycle. This allows the unit to do a new 128- bit SSE2 packed integer add uop every two clock cycles.

 A separate shuffle/unpack execution unit can also process 64 SIMD integer bits per clock cycle allowing it to do a full 128-bit shuffle/unpack uop operation each two clock cycles. MMX/SSE2 SIMD integer multiply instructions use the FP multiply hardware mentioned above to also do a 128-bit packed integer multiply uop every two clock cycles.

 The FP divider executes all divide, square root, and remainder uops. It is based on a double-pumped SRT radix-2 algorithm, producing two bits of quotient (or square root) every clock cycle.

 Achieving significantly higher floating-point and multimedia performance requires much more than just fast execution units. It requires a balanced set of capabilities that work together. These programs often have many long latency operations in their inner loops. The very deep buffering of the Pentium 4 processor (126 uops and 48 loads in flight) allows the machine to examine a large section of the program at once. The out-of-order execution hardware often unrolls the inner execution loop of these programs numerous times in its execution window. This dynamic unrolling allows the Pentium 4 processor to overlap the long-latency FP/SSE and memory instructions by finding many independent instructions to work on simultaneously. This deep window buys a lot more performance for most FP/multimedia applications than more execution units would.

 FP/multi-media applications usually need a very high bandwidth memory subsystem. Sometimes FP and multimedia applications do not fit well in the L1 data cache but do fit in the L2 cache. To optimize these applications the Pentium 4 processor has a high bandwidth path from the L2 data cache to the L1 data. Some FP/multi-media applications stream data from memory–no practical cache size will hold the data. They need a high bandwidth path to main memory to perform well. The long 128-byte L2 cache lines together with the hardware prefetcher described below help to prefetch the data that the application will soon need, effectively hiding the long memory latency. The high bandwidth system bus of the Pentium 4 processor allows this prefetching to help keep the execution engine well fed with streaming data.

**4.4- Memory Subsystem**

 The Pentium 4 processor has a highly capable memory subsystem to enable the new, emerging, high-bandwidth stream-oriented applications such as 3D, video, and content creation. The memory subsystem includes the Level 2 (L2) cache and the system bus. The L2 cache stores data that cannot fit in the Level 1 (L1) caches. The external system bus is used to access main memory when the L2 cache has a cache miss and also to access the system I/O devices.

**Level 2 Instruction and Data Cache**

 The L2 cache is a 256K-byte cache that holds both instructions that miss the Trace Cache and data that miss the L1 data cache. The L2 cache is organized as an 8-way set-associative cache with 128 bytes per cache line. These 128-byte cache lines consist of two 64-byte sectors.

 A miss in the L2 cache typically initiates two 64-byte access requests to the system bus to fill both halves of the cache line. The L2 cache is a write-back cache that allocates new cache lines on load or store misses. It has a net load-use access latency of seven clock cycles. A new cache operation can begin every two processor clock cycles for a peak bandwidth of 48Gbytes per second, when running at 1.5GHz.

 Associated with the L2 cache is a hardware prefetcher that monitors data access patterns and prefetches data automatically into the L2 cache. It attempts to stay 256 bytes ahead of the current data access locations. This prefetcher remembers the history of cache misses to detect concurrent, independent streams of data that it tries to prefetch ahead of use in the program. The prefetcher also tries to minimize prefetching unwanted data that can cause over utilization of the memory system and delay the real accesses the program needs.

**400MHz System Bus**

 The Pentium 4 processor has a system bus with 3.2 G bytes per second of bandwidth. This high bandwidth is a key enabler for applications that stream data from memory. This bandwidth is achieved with a 64-bit wide bus capable of transferring data at a rate of 400MHz. It uses a source-synchronous protocol that quad-pumps the 100MHz bus to give 400 million data transfers per second. It has a split-transaction, deeply pipelined protocol to allow the memory subsystem to overlap many simultaneous requests to actually deliver high memory bandwidths in a real system. The bus protocol has a 64- byte access length.

**5- Performance**

 The Pentium 4 processor delivers the highest SPECint\_base performance of any processor in the world. It also delivers world-class SPECfp2000 performance. These are industry standard benchmarks that evaluate general integer and floating-point application performance.

 Figure 8 shows the performance comparison of a Pentium 4 processor at 1.5GHz compared to a Pentium III processor at 1GHz for various applications. The integer applications are in the 15-20% performance gain while the FP and multi-media applications are in the 30-70% performance advantage range. For FSPEC 2000 the new SSE/SSE2 instructions buy about 5% performance gain compared to an x87-only version. As the compiler improves over time the gain from these new instructions will increase. Also, as the relative frequency of the Pentium 4 processor increases over time (as its design matures), all these performance deltas will increase.

**Figure 8: Performance comparison**

**6- Conclusion**

 The Pentium 4 processor is a new, state-of-the-art processor microarchitecture and design. It is the beginning of a new family of processors that utilize the new Intel NetBurst microarchitecture. Its deeply pipelined design delivers world-leading frequencies and performance. It uses many novel microarchitectural ideas including a Trace Cache, double-clocked ALU, new lowlatency L1 data cache algorithms, and a new high bandwidth system bus. It delivers world-class performance in the areas where added performance makes a difference including media rich environments (video, sound, and speech), 3D applications, workstation applications, and content creation.