In the introduction we have outlined the applications that require 64-bit precision, scientific simulations and CAD/CAM being notable examples. However, the transition from normal scalar code to 64-bit floating-point SSE2 code is complex and it may require some major design changes. A more conservative approach would be moving to scalar SSE2 code, i.e. using scalar instruction (you can easily identify them by the S postfix instead of D) that work on a single 64-bit datum. The top benefit of this strategy is that no parallelism is exploited, so it will naturally fit the existing scalar code, and that it does not require 16-byte alignment of memory operands; the major drawback is that it wastes a potential 2x speedup. The tradeoff between development time and expected performance determines which strategy is more sensible. It should be noted that hand-coding algorithms with SSE2 should be faster than with x87, as SSE2 offers directly addressable registers instead of the unwieldy x87 register stack. Pentium 4 processors show really poor x87 performance, far below that of the current champion AMD Athlon; it is therefore clear that the route to fast floating-point computations passes through SSE2. If Intel can get both Microsoft and Borland to work on a vectorizing compiler, the Pentium 4 may prove to be a winner, but if compiler support will be lackluster (such as the current support for MMX and SSE) it is likely that the Pentium 4 will suffer from lack of optimized software.

Here is the list of SSE2 instructions that extend SSE (adapted from Intel's preliminary documentation):

- **DATA MOVEMENT INSTRUCTIONS**

MOVAPD (move aligned packed double-precision floating-point) transfers a 128-bit double-precision floating-point operand from memory to an XMM register and vice versa, or between XMM registers. The memory address must be aligned to a 16-byte boundary, otherwise a general protection exception (GP#) is generated.
MOVUPD (move unaligned packed double-precision floating-point) transfers a 128-bit double-precision floating-point operand from memory to and XMM register and vice versa, or between XMM registers, without any requirement of alignment of the memory address.
MOVSD (move scalar double-precision floating-point) transfers a 64-bit double-precision floating-point operand from memory to the low 64 bits of an XMM register and vice versa, or between XMM registers. Alignment of the memory address is not required.
MOVHPD (move high packed double-precision floating-point) transfers a 64-bit double-precision floating-point operand from memory to the high 64 bits of an XMM register and vice versa. The low quadword of the register is left unchanged. Alignment of the memory address is not required.
MOVLPD (move low packed double-precision floating-point) transfers a 64-bit double-precision floating-point operand from memory to the low quadword of an XMM register and vice versa.

The high quadword of the register is left unchanged. Alignment of the memory address is not required.
MOVMSKPD (move packed double-precision floating-point mask) extracts the sign bit of each of the two packed double-precision floating-point numbers in an XMM register and saves them in a general purpose register. This 2-bit value can then be used as a condition to perform branching.

- **ARITHMETIC INSTRUCTIONS**

ADDPD (add packed double-precision floating-point) and **SUBPD** (subtract packed double-precision floating-point) add and subtract, respectively, two packed double precision floating-point operands.
ADDSD (add scalar double-precision floating-point) and **SUBSD** (subtract scalar double precision floating-point) add and subtract, respectively, the low quadwords of two double-precision floating-point operands; the high quadword of the source operand is passed through to the destination operand.
MULPD (multiply packed double-precision floating-point) multiplies two packed double-precision floating-point operands.
MULSD (multiply scalar double-precision floating-point) multiplies the low quadwords of two packed double-precision floating-point operands; the high quadword of the source operand is passed through to the destination operand.
DIVPD (divide packed double-precision floating-point) divides two packed double-precision floating-point operands.
DIVSD (divide scalar double-precision floating-point) divides the low 64 bits of two packed double-precision floating-point operands; the high quadword of the source operand is passed through to the destination operand.
SQRTPD (square root packed double-precision floating-point) returns the packed square roots of a packed double-precision floating-point operand to the destination operand.
SQRTSD (square root scalar double-precision floating-point) returns the square root of the low quadword of the packed double-precision floating-point source operand to the low quadword of the destination operand; the high quadword of the source operand is passed through to the destination operand.
MAXPD (maximum packed double-precision floating-point) compares the corresponding double-precision floating-point values from two packed double-precision floating-point operands and returns the numerically higher value from each comparison to the destination operand.
MAXSD (maximum scalar double-precision floating-point) compares the lowdouble-precision floating-point values from two packed double-precision floating-point operands and returns the numerically higher value from the comparison to the low quadword of the destination operand; the high quadword of the source operand is passed through to the destination operand.
MINPD (minimum packed double-precision floating-point) compares the corresponding double-precision floating-point values from two packed double-precision floating point operands and returns the numerically lower value from each comparison to the destination operand.

MINSD (minimum scalar double-precision floating-point) compares the low double-precision floating-point values from two packed double-precision floating-point operands and returns the numerically lower value from the comparison to the low quadword of the destination operand; the high quadword of the source operand is passed through to the destination operand.

- **LOGICAL INSTRUCTIONS**

ANDPD (AND of packed double-precision floating-point) returns a bitwise logical AND of two packed double-precision floating-point operands.
ANDNPD (AND NOT of packed double-precision floating-point) returns a bitwise logical AND NOT of two packed double-precision floating-point operands.
ORPD (OR of packed double-precision floating-point) returns a bitwise logical OR of two packed double-precision floating-point operands.
XORPD (XOR of packed double-precision floating-point) returns a bitwise logical XOR of two packed double-precision floating-point operands.

- **COMPARISON INSTRUCTIONS**

These instructions compare packed and scalar double-precision floating-point values and return the results of the comparison either to the destination operand or to the EFLAGS register.
CMPPD (compare packed double-precision floating-point) compares the corresponding double-precision floating-point values from two packed double-precision floating-point operands, using an immediate operand as a predicate, and returns a 64-bit mask result of all 1s or all 0s for each comparison to the destination operand. The value of the immediate operand allows the selection of any of 12 compare conditions: equal, less than, less than equal, greater than, greater than or equal, unordered, not equal, not less than, not less than or equal, not greater than, not greater than or equal, ordered.
CMPSD (compare scalar double-precision floating-point) compares the low double-precision floating-point values from two packed double-precision floating-point oper-ands, using an immediate operand as a predicate, and returns a 64-bit mask result of all 1s or all 0s for the comparison to the low quadword of the destination operand; the high quadword of the source operand is passed through to the destination operand. The immediate operand selects the compare conditions as with the CMPPD instruction.
COMISD (compare scalar double-precision floating-point and set EFLAGS) and **UCOMISD** (unordered compare scalar double-precision floating-point and set EFLAGS) instructions compare the low quadwords of two packed double-precision floating-point operands and set the ZF, PF, and CF flags in the EFLAGS register to show the result (greater than, less than, equal, or unordered). These two instructions differ as follows: the COMISD instruction signals a floating-point invalid-operation (#I) exception when a source operand is either a QNaN or

SNaN; the UCOMISD instruction only signals an invalid-operation exception when a source operand is an SNaN.

- **SHUFFLE INSTRUCTIONS**

SHUFPD (shuffle packed double-precision floating-point) places either of the two packed double-precision floating-point values from first source operand in the low quad-word of the destination operand, and places either of the two packed double-precision floating-point values from second source operand in the high quadword of the destination operand.
UNPCKHPD (unpacked high packed double-precision floating-point) performs an interleaved unpack of the high double-precision floating-point values of the two source operands. It ignores the low quadwords of the sources.
UNPCKLPD (unpacked low packed double-precision floating-point) performs an interleaved unpack of the low double-precision floating-point values of the two source operands. It ignores the high quadwords of the sources.