# DanSoft Develops VLIW Design

## *Slovakian Startup Seeks Partners for Innovative Instruction Set and CPU*

*by Linley Gwennap*

Pushing the boundaries of instruction-set design, startup DanSoft has developed an innovative VLIW (very long instruction word) architecture and dual-processor CPU design that it believes will outperform any RISC-based processor. The company, located in Eastern Europe, is seeking a foundry partner to build and help market the processor, which is aimed at PCs and high-end consumer devices.

The design started as academic research by Radoslav Danilák, who founded DanSoft with government and private support to commercialize the design. Many of the basic ideas are similar to those being pursued by other academic programs and, in all likelihood, by the Intel/HP alliance and other top-secret industry projects. We believe DanSoft, if it can find a foundry partner, has a legitimate possibility of gaining commercial success, most likely in the embedded market. But whether or not the company succeeds, its design demonstrates some interesting twists that are likely to appear in future processors from mainstream vendors.

DanSoft has designed its architecture for speed: the company believes its design could reach 800–900 MHz in 0.25-micron CMOS. The VLIW design reduces the size of the implementation, shortens the pipeline, and simplifies instruction execution. With a small CPU, the Dan 2433 chip can hold two processors, each capable of executing four RISC-like instructions per cycle. A uniprocessor version, the Dan 1432, would be even smaller and more appropriate for consumer devices. Given its current state, however, the design is unlikely to appear in products before 1999.

### Instruction Grouping Avoids Complexity

The basic DanSoft instructions are similar to traditional RISC instructions; they are about 32 bits wide and directly reference a set of 32 integer registers of 32 bits and 32 floating-point registers of 64 bits. The instruction set contains basic operations such as arithmetic and logical calculations, shifts, loads, stores, and branches. These work mainly as in a RISC processor but with a few twists. In addition, conditional moves and SIMD multimedia extensions match the capabilities of the most recent RISC instruction sets.

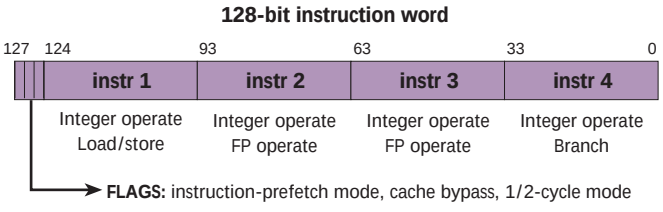


**Figure 1.** Each DanSoft instruction word contains four separate instructions. All four can be integer operations; up to two can be floating-point operations. There can be at most one load/store instruction and one conditional branch.

Like other VLIW designs *(see* **0802MSB.PDF***)*, the DanSoft architecture groups these basic instructions into longer instruction words. In DanSoft's design, a 128-bit instruction word holds four basic instructions, as Figure 1 shows. The compiler guarantees the four instructions have no dependencies; thus, a DanSoft processor contains no complex dependency-checking logic and can simply issue all four instructions at once.

To avoid resource conflicts, the basic DanSoft design includes four complete integer units and enough ports on the integer register file to support four integer operations per cycle. The floating-point unit is divided into an adder and a multiplier; an instruction word may contain two FP instructions if one is a multiply and the other is not. In FP-intensive code, the other two instruction slots can be used for loop control and to load and store operands, as Figure 1 shows.

### Compiler Drives Parallelism

A key to achieving performance in a VLIW design is for the compiler to insert useful instructions into the four instruction slots a large percentage of the time. Without enough parallelism in the code stream, each instruction word may need one or two NOPs, bloating code size and reducing the efficiency of the CPU.

DanSoft's design allows the compiler to explicitly perform speculative execution. In this way, the compiler can reorder instructions beyond branch boundaries to fill extra instruction slots. When instructions are executed speculatively, the danger is that an exception can be triggered, either by a load or by an arithmetic operation. If the speculation is later found to be incorrect, a lengthy exception handler may have been invoked for no reason.

To avoid this scenario, DanSoft includes an "E" bit in arithmetic and load instructions. If this bit is set, exceptions are logged in the register file but ignored by that instruction. The exception is triggered only if the erroneous result is read by a later instruction without the E-bit set. When executing speculative instructions, the compiler simply sets their E bits. The downside of this method is that exceptions are not precise; they can be signaled several cycles after they occur, and the DanSoft processor doesn't maintain enough history to track the original fault. Although imprecise exceptions make debugging more difficult, Alpha and POWER made the same tradeoff to achieve higher performance.

Philips's Trimedia design *(see* **091506.PDF***)*, another VLIW instruction set, achieves a similar effect using predica-

tion. Trimedia operations contain guard bits and are executed only if the guard bits meet specific conditions. The DanSoft method reduces the impact on instruction size (only one extra bit) and simplifies the execution path at the cost of adding one bit per register to log exceptions.

By combining the E-bit option with aggressive compiler optimizations such as loop unrolling and subroutine inlining, DanSoft's compiler generates less than 10% NOPs on most applications, according to the company.

Like Mitsubishi's LIW chip *(see* **101601.PDF***)*, the DanSoft design includes a two-cycle execution mode. In this mode, the first two instructions are executed in one clock cycle, followed by the second two instructions in the next. A flag in each instruction word controls the execution mode. The two-cycle mode is useful in reducing memory space for code segments with limited parallelism; the company claims it reduces NOPs to less than 5% of instructions.

## Nanothreading Improves Parallelism

Despite the best efforts of the compiler, processor execution will occasionally stall due to cache misses and other dynamic events. Because the VLIW design can't rearrange instructions on the fly to avoid stalls, DanSoft uses nanothreading to improve processor utilization.

The DanSoft processor contains a second program counter that points to a nanothread. This PC has only 9 bits, so the nanothread must reside in the same 8K page as the main thread. Whenever the processor stalls on the main thread, it automatically begins fetching instructions from the nanothread. Only one set of registers is available, so the two threads must share the register file. Typically, the nanothread will focus on a simple task, such as prefetching data into buffers, that can be done asynchronously to the main thread. In this case, few registers are required.

This method is a subset of full multithreading using multiple PCs and multiple register sets. The nanothread requires much less hardware: only a 9-bit PC and some simple control logic. Nanothreading will not provide the same level of parallelism as multithreading, but it could provide a performance boost if handled properly by the programmer. DanSoft's compiler contains constructs to simplify the construction of nanothreads, but it cannot create nanothreads without explicit help from the programmer.

## Loop Coloring Overlaps Loads

After simulating the initial simple VLIW design, DanSoft found a performance bottleneck in many loops. For enough cycles to be inserted between loading data and using it to allow for an L1 cache miss, loops must typically be unrolled. This common compiler technique requires lots of registers to cover multiple simultaneous loop iterations; it also increases code size by duplicating instructions.

To avoid this problem, DanSoft uses a technique it calls loop coloring. The lower eight registers can be renamed using 24 additional registers. This technique is simpler than the renaming in high-end RISC chips, because only the lower eight registers can be renamed, and registers are renamed as a block. Register references are expanded to 6 bits; if the upper bit is set, the instruction can refer to the values of the registers from previous iterations of the loop. This mechanism allows loops to be unrolled four times (or more, for simpler loops) without adding four times as many instructions.

| Hint | Prediction | Weight | Fetch Stream |
|---|---|---|---|
| 000 | Fall-though | Heavy | Fetch fall-through |
| 001 | Fall-though | More | Fetch fall-through |
| 010 | Fall-though | Slightly | Fetch both paths |
| 011 | Fall-though | Same | Fetch both paths |
| 111 | Taken | Same | Fetch both paths |
| 110 | Taken | Slightly | Fetch both paths |
| 101 | Taken | More | Fetch taken path |
| 100 | Taken | Heavy | Fetch taken path |

**Table 1.** The compiler can indicate eight levels of branch prediction to the CPU, which responds by following the predicted path and, in some cases, prefetching instructions from the other path. Although the initial CPU has only four responses, future implementations may interpret the eight hints differently.

In case this technique still doesn't provide enough cycles to cover cache misses, the CPU includes instruction queues that are invoked during loop coloring. Each of the four instruction slots has its own queue, and instructions are always issued from the queues in order. If one instruction is stalled—for example, an integer operation waiting for data from a load—the others queues can continue. This method avoids stalling the entire CPU.

In this situation, however, instructions can be executed out of order, because the queues can get out of sync. Register scoreboarding keeps instructions from executing before their data is ready, and all memory references are in the same queue and thus always executed in order, preventing obvious problems. But because the chip lacks the reorder buffer or shadow registers found in most out-of-order chips, recovering from an unexpected exception is difficult.

Loop coloring increases parallelism and reduces the chances of stalling during a cache miss. It requires far fewer instructions than explicit loop unrolling, reducing code size and improving instruction-cache efficiency. It adds complexity compared with a pure VLIW design but is still far simpler than an out-of-order RISC machine. Loop coloring is effective, however, only when exceptions can be avoided.

## Compiler-Driven Branch Prediction

DanSoft assumes most hardware implementations will not use dynamic branch prediction, except for a simple subroutine return buffer. Branch instructions include 3 bits to direct the instruction-fetch stream. These bits specify eight levels of branch direction, as Table 1 shows. For the middle four cases, the processor fetches from both the branch target and the fall-through path, storing the unused instructions in a small queue. In these cases, if the branch is mispredicted, the penalty is only 1–2 cycles.

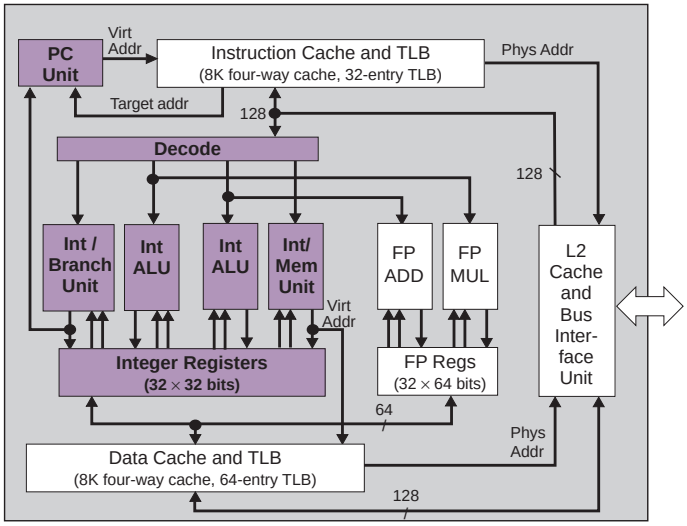


**Figure 2.** A single-processor implementation of the DanSoft design contains an integer core (purple), floating-point unit, small primary caches, and an interface to the external cache and bus.

DanSoft estimates its static-prediction schemes can achieve at least 85% accuracy on most applications. In contrast, advanced dynamic branch-prediction schemes such as the two-level method used in Pentium Pro and Digital's 21264 *(see* **101402.PDF***)* deliver 90–95% accuracy. Both of these processors have much longer misprediction penalties than the DanSoft device; the smaller penalty compensates for the lower accuracy.

DanSoft's branch instructions use pseudo-relative addressing. The low 10 bits of the address contain a portion of the absolute address of the target instruction. The hardware uses these bits to immediately index the instruction cache for the next access, achieving zero-cycle branching for correctly predicted branches. The upper bits of the branch address are a relative value that is added to the current location to calculate the final branch address.

The 21264 implements zero-cycle branching in a similar fashion, but the chip must calculate the target address after the branch is decoded and store it separately in the cache. The DanSoft method requires no additional cache storage. Since the instruction format allows only one branch per instruction word and the branch is always in the same place, the DanSoft method is easy to implement in hardware.

## Simple Processor Implementation

The simpler implementation is the Dan 1432. As Figure 2 shows, the 1432 is very straightforward compared with other four-way superscalar processors. Instructions flow from the cache to the execution units with minimal decoding. There is no branch prediction (other than a 16-entry subroutine buffer), no dependency checking, limited register renaming (only for loop coloring), and no reorder buffer. Deleting this overhead logic leaves room for four full integer ALUs, one of which also handles loads and stores, and an FPU.

For basic integer operations, the pipeline contains only five stages, keeping the mispredicted branch penalty to 2–3 cycles. The shorter penalty applies to some conditional branches, such as less than zero, for which the condition is resolved by the end of the decode stage.

For maximum speed and reduced hardware complexity, none of the integer units contains a full shifter, although they can shift one bit left or right. All other shifts, as well as integer multiplication, are handled through the FPU. The physical location of the shifter has no software-visible effect. The only performance impact is the limit of one shift per cycle, but most superscalar processors have a similar limit. The DanSoft design removes the shifter from a critical timing path, which should improve clock speed.

The FPU also handles some SIMD instructions. For example, each part of the FPU can launch one double-precision operation or two single-precision operations per cycle. This technique is also used in the MIPS V instruction set *(see* **101505.PDF***)*. When operating in SIMD mode, the FPU can sustain two single-precision multiplies and two single-precision adds per cycle, or 3.2 GFLOPS at 800 MHz.

## Short Pipeline Reduces Penalties

To facilitate high clock speeds, the instruction-cache access is divided across two cycles. Again, this is similar to the technique used in the 21264, slated to operate in excess of 500 MHz. After the first cycle, at least the low 10 bits of the instruction must be available; these determine if the instruction is a predicted-taken branch and provide the cache index of the target instruction. These bits can be used to start the next access immediately; otherwise, cache access continues in a sequential fashion. By the end of the second cycle, the complete instruction has been fetched and the instruction TLB has been read.

The 1432 decodes instructions and accesses the register file during the decode stage. All single-cycle instructions are completed in the next stage. The 1432 uses register scoreboarding to avoid stalls due to long-latency instructions. The processor can continue executing instructions during this latency; it stalls only if an instruction attempts to read the result being computed.

Data-cache accesses add up to two cycles. A data-cache miss is treated as a long-latency operation; execution continues until the missing word is needed. To avoid stalls, the DanSoft compiler attempts to prefetch data into the cache. All load instructions contain prefetch bits that instruct the processor to load the next 32 or 64 bytes. The prefetch can be specified to occur only if the data is in the secondary cache, or it can access main memory if necessary.

## Small Primary Caches

Digital adopted a two-cycle cache access for the 21264, mainly to permit expanding the primary caches to 64K each. In contrast, DanSoft has specified just 8K instruction and data caches, although the cache size could easily be

### For More Information

The Dan x4xx processors are not yet available for sale. DanSoft has a technical manual available describing its architecture. Contact DanSoft Ltd. (Košice, Slovak Republic) at 42.95.633.1206; fax 42.95.633.3134; or access the Web at *www.napri.sk/dansoft.*

increased. These caches are four-way set-associative, with small, 16-byte lines to increase their hit rate, but they still hold far less information than competing processors. The company believes that these small caches will enable very high clock speeds.

DanSoft has added several features to improve the efficiency of these caches. Instruction and data prefetching is controlled by the compiler, not the hardware; this method improves the chances that the prefetched information is actually needed. Each instruction word contains a bit to specify whether the next four or eight instructions are to be loaded into prefetch buffers; these instructions are loaded into the cache only when accessed. This level of control avoids polluting the cache with unneeded instructions. Similarly, load instructions contain a bit to specify either 32 or 64 bytes to prefetch.

In addition, all load instructions contain two hint bits that specify whether the data should be stored in both the L1 and L2 caches, just the L2, or neither. In this way, the compiler can control the contents of the caches, preventing rarely used data from forcing useful information out of the cache. These hints can also improve access times: if the caches are bypassed, there is no need to check the tags, and a main-memory access is started without delay.

With these features, the company estimates the miss rate of the caches on typical PC applications will be 4–6% for the instruction cache and 8–10% for the data cache. While these rates are good for such small caches, they are worse than for the 32K or 64K primary caches found in many modern processors.

The DanSoft design assumes a large external secondary cache that is closely coupled to the CPU chip, probably in the same physical package (as in Pentium Pro, for example). Even with this expensive packaging and fast custom cache chips, the delay in accessing the secondary cache will be roughly 8–9 cycles at 900 MHz. With this delay, avoiding primary cache misses is imperative. Proper use of prefetching, supported in some cases by nanothreading, and inserting many instructions between loading data and using it, supported by loop coloring, should avoid many cache misses.

## Dual Processors on Chip

The small primary caches and relatively simple integer core allows DanSoft to fit two complete integer processors on a single die, creating the 2433. As Figure 3 shows, the processor

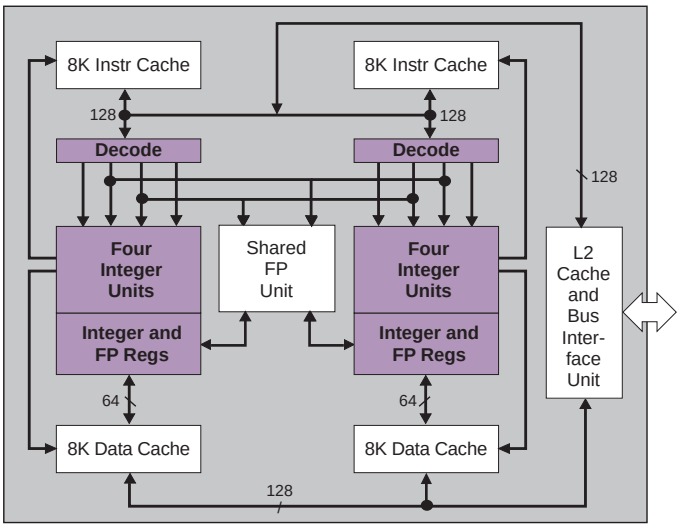


**Figure 3.** The 2433 contains two processors that share a floating-point unit and system interface.

cores share the dual-issue floating-point unit as well as the system interface, which connects to both the second-level cache and the main system bus.

Most programs will not use the floating-point unit at all, and even for FP-intensive programs, the peak performance of several GFLOPS for the single FPU should be adequate. Instead of assigning the FPU to one processor, it is shared by both through a simple arbiter that allows each processor to access the FPU for one cycle if the other is waiting. Both processors have their own FP register file to maintain separate contexts. In most cases, DanSoft expects one processor to handle FP-intensive threads while the other takes care of integer-only housekeeping, but the sharing arrangement provides flexibility with little hardware overhead.

While the high single-precision FP throughput of the 2433's FPU is ideal for 3D geometry, DanSoft is considering adding a separate rendering unit on the processor chip. This unit, shared by both processors, would perform polygon fills, texture mapping, alpha blending, and Z-buffering while the processors handled the geometry calculations. This combination would eliminate the need for an external media processor or graphics chip.

Both processors access the external system through a 128-bit bus. This bus can fetch an entire 16-byte cache line in a single access, quickly freeing the bus for another access. DanSoft has not completely specified this interface but assumes the bus protocol includes a separate address bus and multiple pending transactions.

The size of the secondary cache can vary, but DanSoft's simulations assume 512K. To back up this cache, DanSoft expects a high-bandwidth Rambus-style memory subsystem. Again, this interface has not been specified.

DanSoft has not completed a circuit design but estimates the single-processor design will require 3.7 million transistors: 1.2 million for the two 8K caches plus 2.5 million for the four-way superscalar VLIW core. By comparison,

UltraSparc, a four-way superscalar RISC, contains 2.0 million logic transistors.

The two-processor 2433 design is expected to contain 5.5 million transistors, about the same as Pentium Pro. These estimates imply the 2433 would require about the same die area as Pentium Pro (196 $mm^2$ in a 0.35-micron four-layer-metal process), assuming an equivalent process technology and Intel's level of circuit-design expertise. The latter may be a stretch, but in a typical 0.35-micron process, the design should easily fit within a 300-$mm^2$ die. DanSoft's aggressive estimates show the 2433 fitting into about 150 $mm^2$ in a 0.35-micron process or 100 $mm^2$ in 0.25-micron CMOS.

## Performance Could Exceed Digital's 21264

DanSoft compares its design to the Alpha 21264's and believes that the 2433 can run 25% faster due to its simpler design. Although the elimination of out-of-order logic simplifies the design, it doesn't necessarily allow high clock speeds; Digital has carefully inserted extra pipe stages where needed. Like the 21264, the DanSoft chip is designed to clock at the speed of the integer ALU, but the startup has removed the shifter from the ALU, shortening the critical path. The company estimates this change, combined with the narrower ALU (32 bits vs. 64) could improve clock speed by 25%.

On the other hand, to realize this gain, DanSoft would have to find a circuit-design partner that can match Digital's expertise in high-speed logic design. We believe the DanSoft chip, with reasonable circuit design, could perhaps match Digital's clock speeds. We expect Digital to deliver parts at 800–900 MHz using 0.25-micron CMOS, so DanSoft's claims to reach these speeds are not unreasonable.

Assuming the 2433 could match the 21264's clock speed, the next issue is the instructions per cycle (IPC). Both processors have a peak rate of four integer operations per cycle. If DanSoft has a strong compiler, it could theoretically exceed the IPC of the 21264, since the Alpha chip can schedule instructions within an 80-instruction window, whereas the compiler can examine hundreds or thousands of instructions at once. For recompiled code, however, the 21264 compiler can rearrange instructions in the same way as the DanSoft compiler, so there is no advantage for the VLIW design. DanSoft gains a small advantage from instructions, such as conditional load, that are not found in Alpha.

Compilers can only look at static instruction scheduling; the 21264 can handle dynamic events. In particular, the Alpha chip's extensive dynamic branch prediction logic should outperform the static prediction of the 2433. On the other hand, the DanSoft design has a much smaller mispredicted branch penalty, so this effect may even out. In summary, the 2433 could achieve IPC similar to the 21264, assuming DanSoft has similar or better compiler technology.

The trump card of the VLIW design is the second processor unit, crammed onto the chip by eliminating the complexity of the 21264's out-of-order design. (Note that this complexity is almost entirely for executing "old" unrecompiled binaries, a problem DanSoft doesn't face.) Each of the two DanSoft CPUs could theoretically match the throughput of the 21264. For Windows NT applications that take advantage of the second CPU, this design could deliver a significant performance advantage over the 21264.

To match the performance of the 21264 on real applications, however, the 2433 needs an improved cache structure and high-bandwidth system interface. The 8K primary caches are inadequate for modern software; they can be enlarged without modifying the core architecture. Designing a good system interface is more difficult. If DanSoft finds the right system partner, it could unleash the performance of the VLIW core. The startup is engaged in high-level discussions with at least one major semiconductor vendor that may fab its parts.

## Ideas Need More Development

One major criticism of VLIW designs in general is the difficulty of improving performance without breaking binary compatibility. By combining VLIW with multiple processors on a chip, DanSoft shows one method of getting around this problem. Varying clock speeds and cache sizes could offer additional price/performance points. For the longer term, the company is developing a seven-way VLIW design that would execute four-way binaries: each instruction word will simply use four of the seven available execution slots.

DanSoft's design is the first general-purpose VLIW microprocessor to be fleshed out to this level in public. The small European vendor has developed some interesting ideas that show significant promise in delivering performance better than that of the leading processors expected in the near future. Some of these ideas are likely to appear in other next-generation instruction sets, including the HP/Intel IA-64.

To be successful in the market, DanSoft has a long path ahead. The company needs a strong system partner with competitive circuit-design capabilities and IC process technology. Together, these companies must implement the VLIW core design and add a high-performance cache structure and system interface. DanSoft must complete a competitive compiler and other development tools.

To play in the PC market, a port of Windows NT is required, and Microsoft's willingness to support a radical new architecture with no installed base is near zero. A more likely possibility is the embedded market, where binary compatibility is not as important. DanSoft's design is only slightly more radical than Philips's TM-1 media processor or TI's 'C6x DSP *(see* **110204.PDF***)*. With many emerging applications seeking immense signal-processing capabilities, the power of the DanSoft design could be compelling.

The DanSoft chip is much further from implementation than these two devices, however. Until the Slovakian startup finds a credible partner, its technology will remain unavailable for product designs. As Alpha has shown, however, the lure of leadership performance is a forceful one, and DanSoft is counting on landing a big fish with it. M