VOLUME 13, NUMBER 2
FEBRUARY 15, 1999

MICROPROCESSOR REPORT
THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE

The Russians Are Coming

Supercomputer Maker Elbrus Seeks to Join x86/IA-64 Melee

by Keith Diefendorff

Well-known supercomputer architect Boris Babaian—often called the Seymour Cray of Russia—has disclosed a new processor that his company, Elbrus International, is developing. The E2k processor uses EPIC (explicitly parallel instruction computing) techniques similar to those Intel and HP will use in their upcoming Merced. According to Babaian, however, the E2k will deliver higher performance with lower power and less silicon.

The Elbrus team has the credentials and experience to deliver. The core of the team has been together for over 40 years, developing supercomputers for the Soviet Union's defense establishments (see sidebar on following page). The company has developed computers based on superscalar, shared-memory multiprocessing, and EPIC techniques, long before papers on those subjects appeared in the West. Lacking state-of-the-art semiconductor technology and access to the funds available in capitalist nations, however, Elbrus supercomputers have never quite matched the performance of their Western counterparts. So it shall be for the E2k, unless the company can find a partner or other source of capital.

Even if the E2k never sees the light of day, it is a fascinating device on technical grounds, and it could be a harbinger of things to come in processors such as Transmeta's rumored x86-compatible VLIW engine and Intel's Merced.

Impressive Performance Claims

Babaian's claims for the E2k would seem unbelievable, if not for the credibility of the Elbrus team. In a 0.18-micron six-layer-metal process, he says, the E2k will run at 1.2 GHz and deliver 135 SPECint95 and 350 SPECfp95, yet it will require only 35 W of power and 126 mm^2 of silicon (with 256K of on-chip L2 cache). We project that in a similar process Merced will operate at 800 MHz and deliver 45 SPECint95 and 70 SPECfp95 in 300 mm^2 of silicon at 60 W. Merced, however, is ahead of the E2k in development by at least a year.

Even more amazing, Babaian claims the E2k processor will be x86 binary compatible and, after a few tweaks, IA-64 (Merced) compatible as well. To achieve this feat, Elbrus will rely on binary compilation assisted by emulation hooks in the processor, a strategy which, not coincidentally, is similar to the tack that Transmeta is apparently taking (see MPR 12/7/98, p. 9). The similarity might arise from the fact that Transmeta cofounder and CEO Dave Ditzel spent several years at Sun working with Elbrus. Babaian believes, however, that his company's binary-compilation technology is more advanced than any other on the planet.

Admittedly, the E2k is only an executable Verilog database at this time, so Babaian's claims are not completely tested. The E2k architecture, however, is based on many techniques that were proved in the Elbrus-3 processor, which was fabricated in 1991. That design, based on an ancient process, utilized 15 million transistors in about 3,000 LSI and MSI chips and delivered twice the performance of a Cray Y-MP.

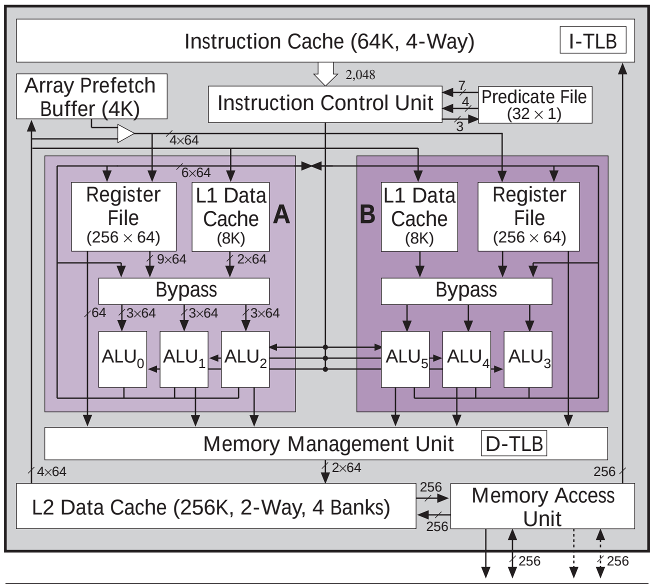


Figure 1. The E2k data path is divided into two clusters (A and B), with duplicate copies of the register files and L1 caches to provide a large number of fast ports for ultrahigh operand bandwidth.

Adding to our skepticism about Elbrus's claims is the fact that the company has been unable to afford the CAD tools to simulate the E2k down to the transistor level. But the 366-person Elbrus team is resourceful, developing many of its own CAD tools. Babaian also points out that the E2k's microarchitecture is conceptually simple, which allowed his team to handcraft every critical path in the processor and to verify the timing of each with HSPICE—a time-consuming procedure for sure, but in post-cold-war Russia, engineering time is more plentiful than rubles for computer-simulation farms.

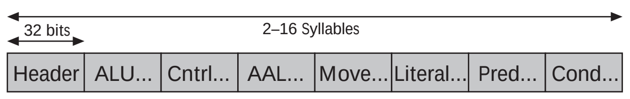
Elbrus Selects EPIC, Shuns Superscalar

In 1978, almost 15 years ahead of Western superscalar processors, Elbrus implemented a two-issue out-of-order processor with register renaming and speculative execution. Disappointed with the performance/complexity tradeoff of that design, in 1986 the company began developing architectures based on explicit fine-grain parallelism and wide instruction words—concepts that today are collectively called EPIC. Since finishing Elbrus-3, which was based on these concepts, Babaian's team has been refining the architecture, circuit techniques, and compilers for the E2k.

As Figure 1 shows, the E2k has some features in common with IA-64, notably a very large general register file and a predicate register file to support predicated execution. The two designs both employ mechanisms to avoid the code inefficiency of traditional VLIW machines, which required NOPs to fill unused execution slots. The E2k, however, is different from Merced in its details and has features that are either not in that chip or have not yet been disclosed.

To improve code efficiency, IA-64 uses fixed-length (128-bit) instruction bundles, each containing three instructions. Each bundle has a template field that specifies which instructions in current or future bundles can execute in parallel. The E2k solves the problem with a variable-length instruction format, as Figure 2 shows. Each E2k instruction consists of a header syllable followed by one to fifteen 32-bit instruction syllables (the architecture is extensible to 32 syllables).

The header field specifies the length of the instruction, thus avoiding the need to decode it to determine its length, a problem with most variable-length instruction sets. Subsequent syllables specify operations that can all be executed in parallel. The syllables not only control execution-unit function and operand selection but also predicate processing, branch/predicate conditions, literal constants, and movement of data from the array prefetch buffer (described later).



Header:	length and structure info (1)
ALU:	execution unit function (6)
Cntl:	prepare to branch control (3)
AAL:	additional ALU function for chained operations (2)
Move:	move data from array prefetch buffer to register (4)
Literal:	supply literal constant to execution unit (4)
Pred:	predicate logic calculations (3)
Cond:	predicate and execution-unit mask (3)

Figure 2. The E2k uses a variable-length VLIW format with up to sixteen 32-bit instruction syllables describing operations that the CPU is free to execute in parallel. The header specifies the number and types of syllables in a given VLIW instruction. Maximum number of each type of syllable is shown in parentheses. (Source: Elbrus)

Lots of Registers, Lots of Ports

Architecturally, the E2k uses a unified 256-entry register file, whereas IA-64 uses separate 128-entry integer and floating-point files. Babaian says that the unified file reduces data shuffling and eliminates fragmentation loss (not enough of one type of register or the other), increasing the effective register namespace over a split-file approach. A large namespace is crucial for avoiding false dependencies in machines without dynamic register renaming.

The unified register file also offers a symmetry that Elbrus finds useful for delivering uniformly high operand bandwidth to its execution units. Operands are delivered through 30 register ports—20 read and 10 write—supplying an operand bandwidth of up to 288 Gbytes/s (at 1.2 GHz).

E2k's register file is scoreboarded to interlock the pipeline. The compiler uses its knowledge of the pipeline for optimal scheduling only; the hardware ensures that results are logically correct, even in the face of unpredictable delays, such as cache misses. Lack of this feature contributed to the failure of earlier statically scheduled machines, such as Multiflow's VLIW (see MPR 2/14/94, p. 18), Floating Point Systems' array processor, and Intel's i860.

Register Window Speeds Context Switch

Large register files require considerable time to save and restore across procedure and program boundaries. The E2k solves the problem with register windowing. The approach, however, is different than SPARC's fixed-size register windows: in the E2k, the register file is treated as a circular buffer with a single variable-sized window (up to 192 registers) positioned by a base register. Register addressing within each procedure context is relative to the current base.

On a procedure call, the window is advanced by adding to the base. If the register file overflows, hardware spills registers to the stack in memory to make room for the new window. Similarly, hardware fills the register file from the stack when underflow occurs on a procedure return. Registers are moved to or from memory in the background, overlapping spill/fill traffic with instruction execution.

The windowing mechanism is not free. The cost is the time to add the base offset to each operand specifier before accessing registers. The extra time requires an additional pipeline stage, which, although it does not impact the performance of straightline code, does penalize branch performance. The E2k deals with the additional pipeline delay with its branch-preparation capability, as we shall see later.

Predicated Execution Removes Branches

Elbrus goes to great lengths either to avoid branches or, when that is unavoidable, minimize their effects. To avoid branches, the E2k uses the same technique used by IA-64: predicated execution. The details are different, but the ideas similar.

The E2k has a 32-entry predicate register file, with each predicate available in either its true or complement form. A single VLIW instruction can generate up to four predicates (via compare operations), and it can also make logical combinations of up to four predicates, returning three. Like the general register file, the predicate file has register windowing.

Each operation in a VLIW instruction can be separately enabled or disabled based on the value of up to six different predicates that are specified by conditional syllables.

IA-64's 64-predicate register file provides similar capabilities to the E2k's 32 registers, since IA-64 stores the true and complement forms of each predicate in separate registers. IA-64's ability to specify different predicates for every instruction in a VLIW bundle is slightly more flexible than the E2k's scheme, but Babaian says that the additional flexibility is rarely useful and uses more instruction-encoding bits.

When in Doubt, Go Both Ways

When a branch condition can be calculated ahead of the branch, the compiler can often use predicated execution to eliminate the branch. In cases where the condition cannot be resolved ahead of the branch, the compiler invokes the E2k's explicit speculative-execution mechanism. Using this mechanism, the condition can be removed from the critical path and, in some cases, the branch itself eliminated.

To invoke this feature, the compiler "unzips" the code, causing the hardware to execute both paths leading from the branch simultaneously (each path getting a portion of the machine resources). While the condition remains unresolved, instructions are issued speculatively, the appropriate results being selected after the condition is resolved. If both paths can be completed before the condition is resolved, the branch need not be executed at all.

Speculative execution is invoked by an opcode bit in each operation syllable. Speculative instructions have no side effects. In the case of an exception, the results are tagged invalid and recovery information is written to the result register. Nonspeculative instructions trap if they access an invalid operand. This provides the same capability as Merced's speculative load but in a more general way.

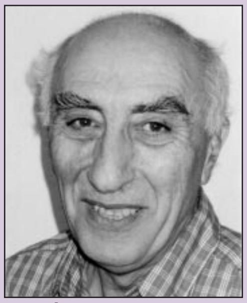
Prepare to Branch

If speculative execution is still under way when the condition resolves, control is transferred to the correct path, so machine resources are not wasted on irrelevant code. For the pipeline, the situation is similar to a branch misprediction in a traditional processor, as it must be flushed and refilled with instructions on the correct path.

To shorten this delay, the E2k uses a branch-preparation operation that fetches the target of the branch from the cache (or prefetches it into the cache if necessary) and moves it through the first four stages of the pipeline. Since branch preparation doesn't depend on the branch condition, it can be moved up in the code sequence, well ahead of the branch. Thus, when the branch is reached, the target instruction stream has already partially executed, effectively shortening the pipeline, as Figure 3 shows. The E2k allows three branch preparation operations to be in progress at the same time.

With predication, unzipping (speculative execution), and branch preparation, the E2k is able to accomplish nearly all control transfers without penalty. The result is that the critical path through the code is limited only by data dependencies (at least for integer code, which typically lacks parallelism, leaving enough machine resources to execute

A Brief History of Elbrus



Russian supercomputer architect Boris Babaian

1957–59: M-40, the first Elbrus computer; built with vacuum tubes.

1964–69: 5E92b, ran the first Russian antimissile system protecting Moscow; transistor construction.

1973–79: Elbrus-1, superscalar RISC processor with out-of-order execution, speculative execution, and register renaming. Capability-based security with dynamic type checking. Ten-CPU shared-memory multiprocessor.

1977–84: Elbrus-2, ten-processor supercomputer, ran critical applications for the Soviet government, including the Russian missile-defense system.

1983: Binary-compiler work started.

1985–1991: Elbrus-3, an EPIC-based VLIW CPU. Sixteen-processor shared-memory multiprocessor.

1991–1999: Low-voltage CMOS. Worked with Sun Microsystems on OS, compilers, and MediaLib. Also worked with Compass (now Avant!) on CAD tools.

1994–1999: Developed E2k architecture to Verilog RTL stage, compilers, and binary-compilation software.

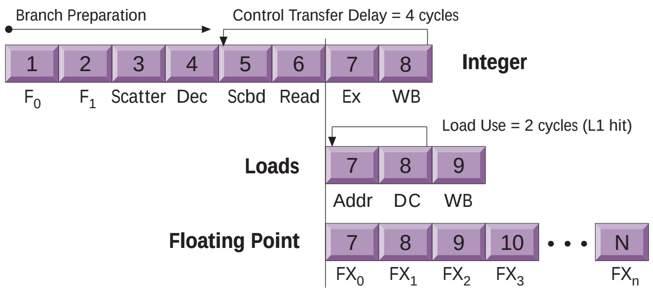


Figure 3. For a high-frequency processor, the E2k uses a remarkably short pipeline, which is effectively made even shorter by its branch-preparation capability.

multiple control paths). These features should contribute greatly to the E2k's SPECint95 performance.

To keep pace with data-dependency-limited execution, the E2k provides enormous instruction-cache bandwidth. Its 64K four-way set-associative instruction cache can deliver four maximum-length (64-byte) VLIW instructions per cycle (2,048 bits!). The instruction-cache line is a long 256 bytes, allowing only 256 lines in the cache. Although the instruction cache can fill at a peak rate of 32 bytes per cycle, instructions must come all the way from memory, as the on-chip L2 holds only data, not instructions. Such long lines, so few entries, and such long fill latency will either be a performance problem or a testament to the compiler's ability to linearize code.

Supercomputer Background Evident

The E2k provides two features that are obvious descendants of Elbrus's supercomputer history: loop overlapping and an array prefetch buffer.

Loops are the basic construct that signals the presence of parallelism in a program. But exploiting the parallelism in a loop isn't easy—especially without dynamic register renaming to prevent the loop from stalling on register-name reuse between iterations. The traditional approach, and the one likely to be used in Merced, is loop unrolling with software renaming. That technique, however, results in considerable code expansion, which reduces instruction-cache efficiency and increases memory-bandwidth requirements.

To circumvent these effects, the E2k provides a simple hardware-renaming scheme. Using the same mechanism used for register windowing, the compiler sets up rotating bases for the general and predicate-register files. Within the loop, register addressing is relative to this base. On each iteration, the base is advanced, supplying a new set of registers. With a different set of registers for each iteration, loops are free to be overlapped to the maximum extent supported by the machine resources.

A related feature is the E2k's support for loop startup (prologue) and termination (epilogue). The E2k allows these functions to be provided within the loop itself by disabling instruction side effects (e.g., memory stores) during initial loop iterations and disabling some operations (e.g., memory loads) on the final iterations. The code savings can be substantial, especially in long floating-point loops.

To speed access to array elements—the most common type of loop data—the E2k provides an array prefetch buffer. This buffer is filled by an asynchronous hardware prefetch engine that runs ahead of the main loop, overlapping long memory accesses with loop execution. Organized as a FIFO queue, the array prefetch buffer linearizes array elements that are often scattered on various strides across memory. In addition, the array prefetch buffer prevents array elements from polluting the data cache and significantly reduces pressure on the register file.

The 4K array prefetch buffer provides up to 64 prefetch areas. The buffer is dual banked, and each bank is dual ported, allowing up to four 64-bit reads every cycle. The E2k's loop overlapping and array-prefetching features contribute significantly to performance on highly parallel code, as attested to by its impressive 350-SPECfp95 estimate.

Clusters Keep Signal Paths Short

From the start, Elbrus's objective was a high clock rate. That objective was one of the motivations behind the selection of an EPIC architecture, which moves the complex job of instruction scheduling from runtime (hardware) to compile time (software). Beyond architecture, however, Elbrus has taken implementation steps to further boost frequency.

To minimize wire lengths and limit the loading on critical circuits, Elbrus split the microarchitecture into two nearly identical clusters, similar to the approach used by the Alpha 21264 (see MPR 10/28/96, p. 11). In the E2k, each cluster contains half of the execution units as well as its own copy of the register file and the L1 data cache. Each copy of the register file has 10 read and 10 write ports, for a total of 20 read and 10 write ports. Data must be written to both files to keep them coherent, as is the case for the L1 caches.

The time-critical register-bypass circuits, which forward data directly from one instruction to the next to minimize latency, are also split to increase their speed. This partitioning minimizes the latency of data flowing within a cluster, trading off an additional cycle of latency for data crossing between clusters. The compiler is aware of this hazard and is careful to allocate operations to the appropriate cluster, so the delay rarely stalls the pipeline.

Like the register file, the L1 data cache is split into two identical halves that always contain the same data. The L1 cache halves were kept small (8K), direct-mapped, and write-through to minimize latency. Both halves are dual ported and have a latency of two cycles. Being fully pipelined, the L1 can deliver a total load bandwidth on hits of over 38 Gbytes/s. The L1 is fed from a common 256K L2 data cache, which, like the L1, is nonblocking. The L2, however, has a longer latency of eight cycles and is two-way set-associative, copyback, and interleaved four ways. The E2k's memory access unit can support either one or two 256-bit concurrent memory buses.

Like the caches, the E2k's data TLB uses a two-level hierarchical organization with split single-cycle 16-entry fully associative level-one TLBs that are backed by a common two-cycle 512-entry four-way set-associative level-two TLB. The TLB hierarchy can translate addresses for two simultaneous accesses to the L2 data cache (or memory). While the L2 cache is physically indexed and tagged, the L1 is virtually indexed and tagged, so it does not require TLB accesses.

Massive Potential ILP

Each of the E2k's clusters comprises three execution-unit pipelines; each unit takes three register-source operands and returns a single result operand to the register file. The two clusters are symmetric except for division, which is restricted to the B cluster. Execution units within each cluster, however, are not symmetric. One of the execution units in each cluster, for example, provides two cascaded ALUs that implement a combined-integer capability that allows two chained-integer operations to be issued to one execution unit. Table 1 shows the maximum issue rates, latencies, throughput, and structural hazards for a few of the E2k's common operations.

The E2k can issue a maximum of 14 operations per cycle on scalar integer code and 16 ops/cycle on floating-point scalar code. In loop mode, the E2k can issue at a whopping 23 ops/cycle, including loop counters and address incrementers. The operation mix, data dependencies, and memory latency rarely allow these rates to be sustained, although the E2k's ability to execute two paths in parallel keeps the units highly utilized, even on integer code with little parallelism.

Circuit Design Boosts Frequency

The E2k's circuit-design innovations are as impressive as its microarchitecture. In its current state, the E2k uses 10 gates (equivalent fan-in-2 fan-out-3 NAND gates) in its worst-case paths. Elbrus's 1.2-GHz claim, however, is made assuming 12 gates, providing a 20% safety margin in its estimates.

For much of the logic, Elbrus uses standard static-CMOS design to permit automatic synthesis. For its high-speed paths, however, Elbrus implements self-reset logic. These circuits have some advantages over the dynamic circuits used in most modern high-performance processors, including the current record speed holder, the 21264.

To achieve high speed, self-reset gates, like conventional dynamic gates, use distinct evaluate and reset (precharge) phases. Self-reset gates, however, do not use a clock. In Elbrus's design, a wave of data is propagated through the chain of logic gates, with the falling edge of the output pulses serving as the reset for subsequent gates. Without the resistance of a clock transistor in the input-transistor stack, the load charges faster, speeding the gate over traditional dynamic gates—by 10–15%, according to Elbrus. Logic design can be trickier with self-reset gates and in some cases can require more transistors. The transistors, however, are generally smaller, so there is usually no net die-size penalty.

Self-reset gates also save power, because only the gates logically needing precharge get it. In addition, without clocks on the gates, the clock tree is significantly lighter, so it consumes less power. In the Alpha, for example, clock amplifiers consume about 40% of the chip power; Babaian says that self-reset logic could reduce this by a factor of five or more. Furthermore, without all the gates evaluating and precharging on a common clock, the switching noise is more distributed and the noise margin improved. As with most modern processors, active clock gating is used to power-down idle execution units, saving additional power.

Taking a page from Cray's book, Elbrus uses a clock scheme that maximizes the portion of the cycle that is available to perform logic. The Elbrus system uses a single-phase clock driving an alternating sequence of active-high and active-low latches with self-reset logic between. The Elbrus clock system suffers little from the effects of clock skew and latch setup-time requirements, recovering most of the 10–15% of the cycle that is lost to these overheads in conventional microprocessor clocking systems. Elbrus's unique clock scheme also lends itself to time borrowing, a feature that can avoid increasing the cycle time to accommodate a circuit that just barely misses a clock edge.

Low-Swing Signals for Long Lines

To reduce power, noise, and transmission-line delay on long lines, such as buses and global routing, Elbrus uses low-voltage signaling, reminiscent of ECL days. The E2k uses a separate 0.7-V supply to power its low-swing bus drivers. The drivers operate differentially, swinging above and below this reference voltage by 100 mV. At the receiving end, the low-level signals are amplified to single-ended digital signals by high-speed sense amplifiers. According to Elbrus, this technique dissipates 75% less dynamic power than conventional long-line drivers, and it is used extensively throughout the E2k, most notably in its speed-critical register-bypass buses.

The low-swing technique is also used in the E2k's register file, allowing the use of common word and bit lines for read and write operations and enabling a read and a write on every cycle. Register reads occur in the first half of each clock cycle, with data propagating to the execution units in the second half. Register writes occur in the second half of the cycle, with data propagating from the execution units to the register file in the first half.

A clever trick is used to save power in the register file: when multiple ports target the same register, only a single bit-line is activated, with the value distributed to multiple destinations through the register-bypass circuits.

Hardware Only Half the Story

Building fast EPIC hardware is one thing; getting good performance, another. With EPIC, performance is at the mercy of the compiler, which has the sole responsibility for scheduling

	Issue	Each Operation		Cluster A			Cluster B		
Operation	**Rate**	**Latency**	**Thruput**	**0**	**1**	**2**	**3**	**4**	**5**
Integer ALU	6 ops	1 cycle	1 cycle	•	•	•	•	•	•
Int Combined	2 ops	2 cycle	1 cycle		•			•	
Load/Store L1	4 ops	2 cycles	1 cycle	•		•	•		•
Load/Store L2	4 ops	8 cycles	1 cycle	•		•	•		•
FP Add (32/64)	4 ops	4 cycles	1 cycle	•	•		•	•	
FP Add (80)	4 ops	5 cycles	2 cycles	•	•		•	•	
Multiply (32/64)	4 ops	4–5	1 cycle	•	•		•	•	
FP Mul (80)	4 ops	6–7	2 cycles	•	•		•	•	
FP Madd (32/64)	4 ops	8–9	1 cycle	•	•		•	•	
Divide (32)	1 op	10–13	2 cycles						•
Divide (64)	1 op	10–17	2 cycles						•
FP Div (32/64)	1 op	11/14	2 cycles						•
MMX Mul/Shift	2 ops	2 cycles	1 cycle		•			•	
MMX Add/Sub	2 ops	1 cycle	1 cycle	•			•		

Table 1. With the combined-integer feature, eight integer operations can be issued every cycle. With FP multiply-add, the E2k can achieve a peak issue rate of 23 operations per cycle while looping. (Source: Elbrus)

machine resources to maximize performance. Elbrus appears no less adept on this front than on architecture or circuits.

Like most EPIC compilers, the Elbrus compiler performs all the classical high-level optimizations, such as inter- and intraprocedural data-flow analysis and transformations, global register allocation, hyperblock construction, and loop splitting. The key to the Elbrus compiler, however, is its ability to exploit the E2k's architectural features.

For example, the compiler aggressively applies the E2k's speculative execution and loop overlapping. For these, the compiler splits the code into regions of high parallelism and limited parallelism. In high-parallelism regions, the compiler schedules code to maximize throughput, then applies loop overlapping to minimize the code size and to maximize the register usage in order to fully load the execution units.

In the limited-parallelism regions, the compiler identifies critical paths and schedules the code to maximize execution speed. For this, it moves the calculation of control-flow conditions off the critical path by unzipping the code and applying speculative execution (executing both paths simultaneously). The compiler then uses profile-driven code motion and scheduling to stitch the regions back together efficiently.

Simulations show the E2k and its compiler to be effective, currently achieving 3.6 operations per cycle on SPECint95, about twice the IPC of the 21264. The Elbrus compiler team expects to boost that figure to 4.5 ops/cycle soon. On SPECfp95, the compiler is now achieving 10 ops/cycle, almost three times that of the 21264, with a goal of 11.5.

Binary Compiler the Only Hope for Success

Elbrus long ago realized that its chances of successfully bringing its EPIC technology to market as a new architecture were nil. Therefore, it set a goal to be x86 compatible, a feat the company intends to achieve with a binary compiler assisted by emulation features built into the E2k hardware.

Binary compilers are not new to Elbrus. The company began work on the technology in the early 1980s, and it has now matured to the point that Elbrus feels confident in boasting of the best binary compiler in existence. In simulations, Elbrus has demonstrated emulated x86 and SPARC performance that is 70–90% the speed of native E2k code on SPECfp.

Unlike other binary translators, such as Alpha's FX!32 (see MPR 3/5/96, p. 11), Elbrus's does not depend on any characteristics of the emulated software environment. This fact will allow it to achieve, theoretically, 100% binary compatibility for any x86 program on any OS. Emulation with E2k will be completely transparent to the user, functioning exactly like a real x86 processor even when debugging.

Elbrus uses a combination of static and dynamic binary compilation, in contrast to Transmeta, which apparently will rely on dynamic compilation. Elbrus, however, takes a similar approach to that revealed in the recent Transmeta patent (see MPR 12/7/98, p. 9) by providing features in the E2k hardware to improve the processor as a target for binary compilation. The E2k's emulation features fall into two categories: generic assists and platform-specific assists.

Generic emulation-assist features include biendian byte ordering, TLB hooks to detect self-modifying code, a call/return cache, memory address disambiguation, and two virtual-address spaces. The call/return cache improves performance on switch tables, which the binary compiler depends on to translate emulated indirect-jump addresses to the E2k addresses in binary-translated code.

The binary compiler keeps memory stores in strict program order, but it can hoist loads above stores to hide memory latency and boost performance. For this, the binary compiler utilizes the E2k's disambiguation memory (DM), which checks for address aliases at runtime. The DM disallows offending stores and signals a fault that can be detected by a check instruction placed at the original position of the load.

An important feature for achieving its goal of 100% compatibility is the E2k's dual virtual-address space. With this feature, the E2k can maintain one address space exactly as it would appear on the foreign machine and use the second address space for emulation code. This firewall prevents any disturbance of the emulated environment by the host, a major source of incompatibility in existing emulators.

In addition to these generic emulation-assist features, the E2k provides x86-specific features that include mirroring of the x86's address-translation architecture (segmentation, page size, page-table layout, etc.), LOCK prefix implementation for multiprocessor interlocks, an 80-bit floating-point data type (that uses two E2k registers), and ALU-primitive compatibility with x86, including the MMX ALUs.

Due to the striking similarities between the IA-64 architecture and the E2k, Elbrus believes it will easily be able to modify the E2k hardware to emulate both x86 and IA-64. The difficulty of doing that, however, cannot be completely assessed until the details of IA-64 are disclosed.

And Secure to Boot

A feature of the E2k that reveals its roots in the defense industry is its air-tight security. Elbrus's security model is built on capabilities, an architecture that ended abruptly in the West with the Intel 432 disaster. Unlike nearly all previous capability-based machines—which relied on capability lists and suffered severe performance problems for it—the E2k architecture is based on tagged data (two tag bits per 32-bit word). The last three generations of Elbrus supercomputers have all used this technique, and the Russian designers have learned to implement it with no performance penalty.

The feature is more than an esoteric nicety for spooks. The OS can use capabilities to provide ironclad protection against virus propagation, although Microsoft is unlikely to modify its OS to exploit the feature. But with tags, the E2k also implements fully dynamic type checking in hardware, a feature the compiler can use to provide runtime protection against program errors, such as array indexes out of bounds, accesses to uninitialized data, and dangling pointers. Elbrus

For More Information

For more information about Elbrus International (Moscow) or the E2k processor, contact Boris Babaian at 650.618.1581 or send e-mail to *boris.babaian@mcst.ru*.

designers tell the story of the first time they ran SPECint92 on Elbrus-3 and discovered 33 such errors in the code.

Very Low Power on the Horizon

Although 35 W is amazingly little power for a processor of the E2k's capability, it is not low enough for Elbrus. For several years the company has been working on very low voltage CMOS and believes it has discovered ways to avoid the basic problems of poor noise immunity and transistor-speed loss that have thwarted previous efforts. With its techniques, the company thinks it can operate a processor on a 200-mV supply in SOI with little sacrifice in speed.

The proof, of course, is in the pudding, and the company says it is a year away from verifying the concepts with a test chip. Another year would be needed to complete the device libraries and render the E2k in such a process. Elbrus has not yet filed patents on its low-voltage techniques, so it isn't willing to disclose enough information for others to evaluate the technology. The prospects, however, are tantalizing: think of a true supercomputer-class microprocessor in your PDA.

An Interesting Curiosity

Elbrus team members are understandably proud of their baby. It is likely that their enthusiasm, as for most fathers, has led to some inflated claims for the E2k. But the capability and credibility of the Elbrus team does not permit the E2k to be dismissed out of hand. There is little reason to suspect that the team is overstating its processor's capabilities by any more than other competent design teams tend to when still a year from tapeout. Even if Elbrus missed all its targets, as listed in Table 2, by 20%, the E2k would still be 1 GHz, 110 SPECint95, 280 SPECfp95, 150 mm^2, and 45 W—an awesome chip by any standard.

We can only hope that someone will see fit to fund the Elbrus team, so their ideas can be put to the test. It would indeed be a shame if the talent of the Elbrus team and the technology in the E2k and its compilers were lost for lack of a few tens of millions of dollars. But the chances of Elbrus ever seeing its pride and joy in silicon are, alas, slim. The free-fall of the Russian economy has left little hope of finding funds in the company's homeland to launch an attack on Intel. And venture capitalists in the States have little appetite for funding anyone naive enough to consider competing with Intel. (The poor Russians just don't know enough to be scared.) The company could look for refuge in the less challenging embedded space, but this would be a serious waste of the processor's technology and the team's expertise.

Feature	Elbrus E2k
Issue Rate (scalar, integer)	14 operations/cycle
Issue Rate (scalar, floating point)	16 operations/cycle
Issue Rate (loop, floating point)	23 operations/cycle
Clock Rate (in 0.18 µm, 6 LM)	1.2 GHz
Pipeline Length	8 stages (9 for loads)
Execution Units	6 ALU, 4 ld/st, 3 pred, 1 br
L1 Data Cache	8K, direct-mapped
L2 Data Cache	256K, 2-way, 4 banks
Data TLB	16 associative + 512/4-way
Array Prefetch Buffer	4K, 64 areas
Instruction Cache	64K, 4-way set-associative
Instruction TLB	64 entry, fully-associative
Register File	256 registers × 64 bits
Register File Ports	20 read, 10 write
Transistors	28 million
Die Size (in 0.18 µm, 6 LM)	126 mm^2
Power Dissipation	35 W
SPECint/fp95 (est at 1 GHz)	135/350

Table 2. Being a year from tapeout requires some derating of Elbrus's claims for the E2k, but its specifications are still impressive. (Source: Elbrus)

Regardless of Elbrus's architectural prowess, it has little chance of being successful without first-class process technology. Although good architecture can be an advantage, it cannot, over the long run, make up for lagging process technology. This suggests that Elbrus should try to partner with a large semiconductor vendor: AMD, IBM, even Intel come to mind. Those companies, however, have already chosen their paths and would be reluctant to change course for schedule reasons, never mind the NIH (not invented here) syndrome. A partnership with such a company would certainly improve Elbrus's chances, as the company's biggest weakness, aside from lack of resources, is inexperience in implementing multimillion-transistor chips in modern CMOS processes.

As a last resort, the company could license its technology to an existing RISC vendor, such as Compaq, IBM, or Sun, then just fade quietly into the Russian sunset. The company has several U.S. patents granted, several more pending, and another 70 ready for filing. It also claims to be unencumbered by any Intel intellectual-property rights.

Based on the Ditzel-Elbrus connection, it will be interesting to see how many features Transmeta's product ends up having in common with the E2k. Even more interesting is the IA-64 connection. Perhaps it is a coincidence—and perhaps not—that the E2k has striking similarities to IA-64 and that HP executive Peter Rosenbladt was in Moscow meeting with Babaian on August 19, 1991, as the EPIC-based Elbrus-3 supercomputer was being fabricated (and tanks were rolling through the Russian capital in the now-infamous *coup d'état*).

If nothing else, the E2k is a reminder—as damaging as it may be to some egos—that computer architecture is not the sole province of the West. Evidently, RISC, superscalar, multiprocessing, fast ALUs, high-speed CMOS circuits, capabilities, optimizing compilers, binary compilers, and perhaps even EPIC were not such new ideas after all. The E2k also demonstrates that not all good ideas in computer architecture have yet been absorbed into microprocessors. M