Introducing the Intel i860 64-Bit Microprocessor

A million-transistor budget helps this RISC deliver balanced MIPS, Mflops, and graphics performance with no data bottlenecks.

Les Kohn
Neal Margulis

Intel Corp.

The single-chip i860 CPU—a 64-bit, RISC-based microprocessor—executes parallel instructions using mainframe and supercomputer architectural concepts. We designed the 1,000,000-transistor, 10 mm × 15 mm processor (see Figure 1 on the next page) for balanced integer, floating-point, and graphics performance, using the company's latest generation CAD tools and 1-micrometer semiconductor process.

To accommodate our performance goals, we divided the chip area evenly between blocks for integer operations, floating-point operations, and instruction and data cache memories. Inclusion of the RISC (reduced instruction set computing) core, floating-point units, and caches on one chip lets us design wider internal buses, eliminate interchip communication overhead, and offer higher performance. As a result, the i860 avoids off-chip delays and allows users to scale the clock beyond the current 33- and 40-MHz speeds.

We designed the i860 for performance-driven applications such as workstations, minicomputers, application accelerators for existing processors, and parallel supercomputers. The i860 CPU design began with the specification of a general-purpose RISC integer core. However, we felt it necessary to go beyond the traditional 32-bit, one-instruction-per-clock RISC processor. A 64-bit architecture provides the data and instruction bandwidth needed to support multiple operations in each clock cycle. The balanced performance between integer and floating-point computations produces the raw computing power required to support demanding applications such as modeling and simulations.

Finally, we recognized a synergistic opportunity to incorporate a 3D graphics unit that supports interactive visualization of results. The architecture of the i860 CPU provides a complete platform for software vendors developing i860 applications.

Architecture overview. The i860 CPU includes the following units on one chip (see Figure 2):

- the RISC integer core,
- a memory management unit with paging,
- a floating-point control unit,
- a floating-point adder unit,
- a floating-point multiplier unit,
- a 3D graphics unit,

0272-1732/89/0800-0015$01.0 © 1989 IEEE

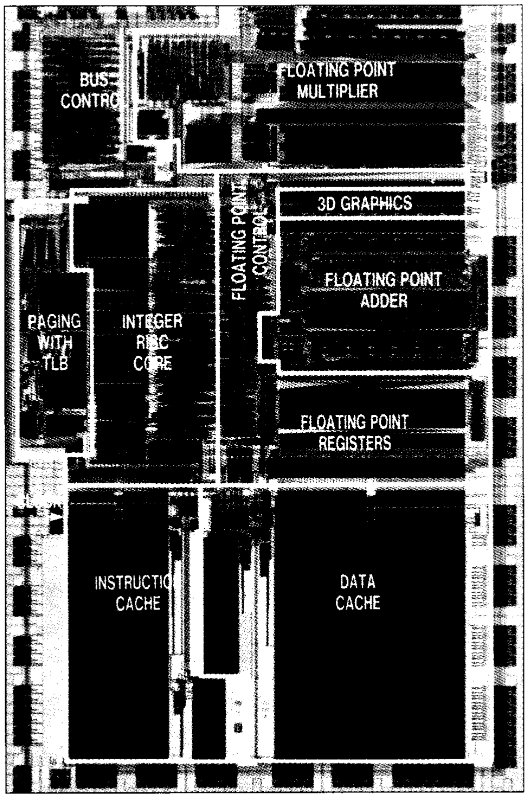

Figure 1. Die photograph of the i860 CPU.

- a 4-Kbyte instruction cache,
- an 8-Kbyte data cache, and
- a bus control unit.

Parallel execution. To support the performance available from multiple functional units, the i860 CPU issues up to three operations each clock cycle. In single-instruction mode, the processor issues either a RISC core instruction or a floating-point instruction each cycle. This mode is useful when the instruction performs scalar operations such as operating system routines.

In dual-instruction mode, the RISC core fetches two 32-bit instructions each clock cycle using the 64-bit-wide instruction cache. One 32-bit instruction moves to the RISC core, and the other moves to the floating-point section for parallel execution. This mode allows the RISC core to keep the floating-point units fed by fetching and storing information and performing loop control, while the floating-point section operates on the data.

The floating-point instructions include a set of operations that initiate both an add and a multiply. The add and multiply, combined with the integer operation, result in three operations each clock cycle. With this fine-grained parallelism, the architecture can support traditional vector processing by software libraries that implement a vector instruction set. The inner loops of the software vector routines operate up to the peak floating-point hardware rate of 80 million floating-point operations per second. Consistent with RISC philosophy, the i860 CPU achieves the performance of hardware vector instructions without the complex control logic of hardware vector instructions. The fine-grained parallelism can also be used in other parallel algorithms that cannot be vectorized.

Register and addressing model. The i860 microprocessor contains separate register files for the integer and floating-point units to support parallel execution. In addition to these register files, as can be seen in Figure 3 on page 18, are six control registers and four special-purpose registers. The RISC core contains the integer register file of thirty-two 32-bit registers, designated R0 through R31 and used for storing addresses or data. The floating-point control unit contains a separate set of thirty-two 32-bit floating-point registers designated F0 through F31. These registers can be addressed individually, as sixteen 64-bit registers, or as eight 128-bit registers. The integer registers contain three ports. Five ports in the floating-point registers allow them to be used as a data staging area for performing loads and stores in parallel with floating-point operations.

The i860 operates on standard integer and floating-point data, as well as pixel data formats for graphics operations. All operations on the integer registers execute on 32-bit data as signed or unsigned operations and additional add and subtract instructions that operate on 64-bit-long words. All 64-bit operations occur in the floating-point registers.

The i860 microprocessor supports a paged virtual address space of four gigabytes. Therefore, data and instructions can be stored anywhere in that space, and multibyte data values are addressed by specifying their lowest addressed byte. Data must be accessed on boundaries that are multiples of their size. For example, two-byte data must be aligned to an address divisible by two, four-byte data on an address divisible by four, and so on, up to 16-byte data values. Data in memory can be stored in either little-endian or big-endian format. (Little-endian format sends the least significant byte, D7-D0, first to the lowest memory address, while big-endian sends the most significant byte first.) Code is always stored in little-endian format. Support for big-endian data allows the processor to operate on data produced by a big-endian processor, without performing a lengthy data conversion.

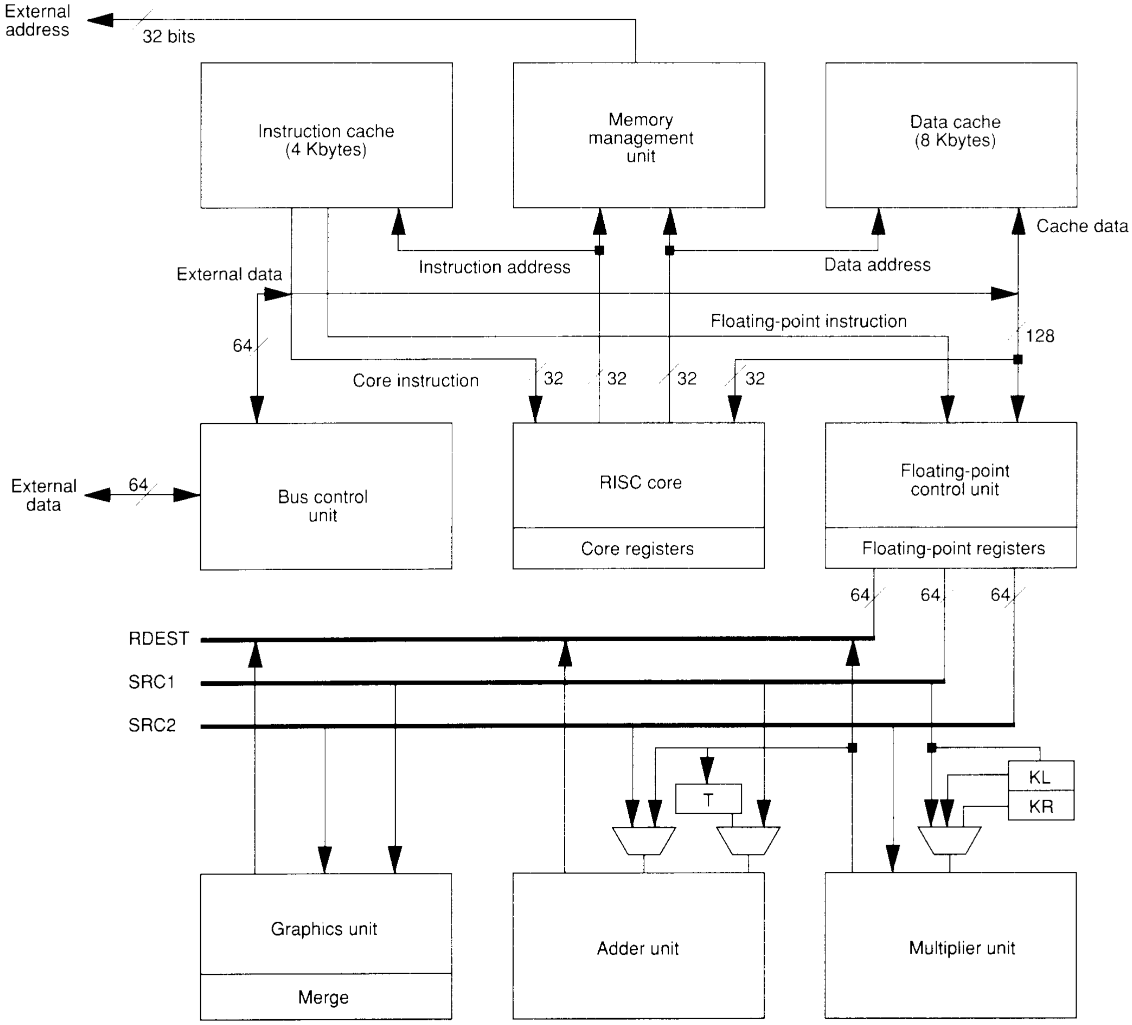

Figure 2. Functional units and data paths of the i860 microprocessor.

RISC core

The RISC core fetches both integer and floating-point instructions. It executes load, store, integer, bit, and control transfer instructions. Table 1 on page 19 lists the full instruction set with the 42 core unit instructions and their mnemonics in the left column. All instructions are 32 bits long and follow the load/store, three-operand style of traditional RISC designs. Only load and store instructions operate on memory; all other instructions operate on registers. Most instructions allow users to specify two source registers and a third register for storing the results.

A key feature of the core unit is its ability to execute most instructions in one clock cycle. The RISC core contains a pipeline consisting of four stages: fetch, decode, execute, and write. We used several techniques to hide clock cycles of instructions that may take more time to complete. Integer register loads from memory take one execution cycle, and the next instruction can begin on the following cycle.

The processor uses a scoreboarding technique to guarantee proper operation of the code and allow the highest possible performance. The scoreboard keeps a history of which registers await data from memory. The actual loading of data takes one clock cycle if it is held in the cache memory buffer available for ready access, but several cycles if it is in main memory. Using scoreboarding, the i860 microprocessor continues execution unless a subsequent instruction attempts to use the data before it is loaded. This condition would cause execution to freeze. An optimizing compiler can organize the code so that freezing rarely occurs by not referencing the load data in the following cycle. Because the hardware implements scoreboarding, it is never necessary to insert NO-OP instructions.

We included several control flow optimizations in the core instruction set. The conditional branch instructions have variations with and without a delay slot. A delay slot allows the processor to execute an instruction following a branch while it is fetching from the branch target. Having both delayed and nondelayed variations of branch instructions allows the compiler to optimize the code easily, whether a branch is likely to be taken or not. Test and branch instructions execute in one clock cycle, a savings of one cycle when testing special cases. Finally, another one-cycle loop control instruction usefully handles tight loops, such as those in vector routines.

Instead of providing a limited set of locked operations, the RISC core provides lock and unlock instructions. With these two instructions a sequence of up to 32 instructions can be interlocked for multiprocessor synchronization. Thus, traditional test and set opera-

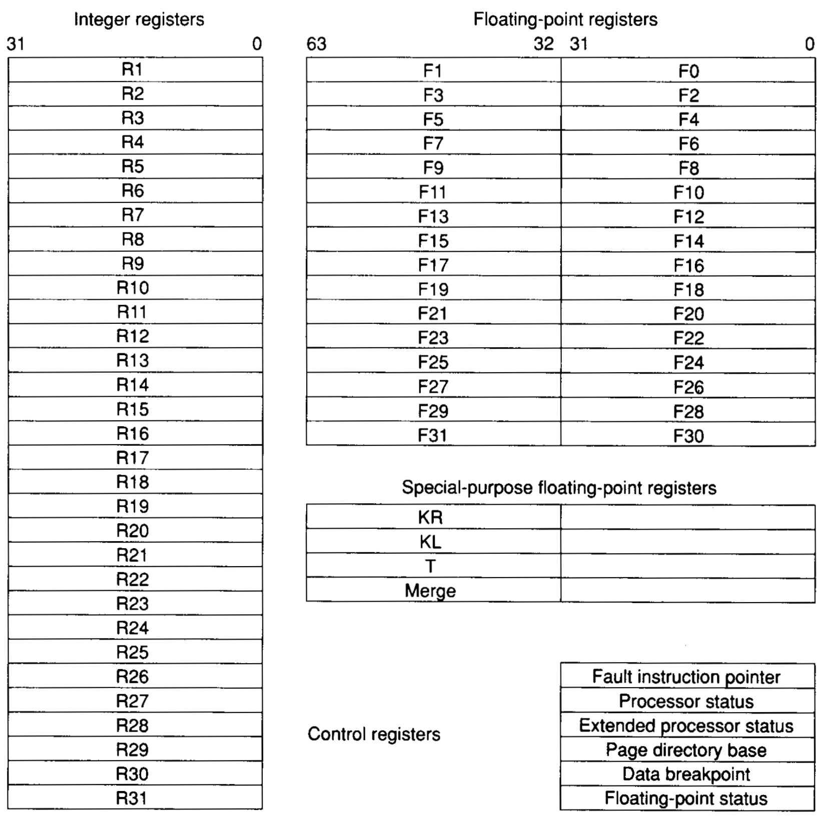

Figure 3. Register set.

Table 1.
Instruction-set summary.

Mnemonic	Description
Core unit	
Load and store instructions	
LD.X	Load integer
ST.X	Store integer
FLD.Y	F-P load
PFLD.Z	Pipelined F-P load
FST.Y	F-P store
PST.D	Pixel store
Register-to-register moves	
IXFR	Transfer integer to F-P register
FXFR	Transfer F-P to integer register
Integer arithmetic instructions	
ADDU	Add unsigned
ADDS	Add signed
SUBU	Subtract unsigned
SUBS	Subtract signed
Shift instructions	
SHL	Shift left
SHR	Shift right
SHRA	Shift right arithmetic
SHRD	Shift right double
Logical instructions	
AND	Logical AND
ANDH	Logical AND high
ANDNOT	Logical AND NOT
ANDNOTH	Logical AND NOT high
OR	Logical OR
ORH	Logical OR high
XOR	Logical exclusive OR
XORH	Logical exclusive OR high
Control-transfer instructions	
TRAP	Software trap
INTOVR	Software trap on integer overflow
BR	Branch direct
BRI	Branch indirect
BC	Branch on CC
BC.T	Branch on CC taken
BNC	Branch on not CC
BNC.T	Branch on not CC taken
BTE	Branch if equal
BTNE	Branch if not equal
BLA	Branch on LCC and add
CALL	Subroutine call
CALLI	Indirect subroutine call
System control instructions	
FLUSH	Cache flush
LD.C	Load from control register
ST.C	Store to control register
LOCK	Begin interlocked sequence
UNLOCK	End interlocked sequence
Floating-point unit	
Floating-point multiplier instructions	
FMUL.P	F-P multiply
PFMUL.P	Pipelined F-P multiply
PFMUL3.DD	Three-stage pipelined F-P multiply
FMLOW.P	F-P multiply low
FRCP.P	F-P reciprocal
FRSQR.P	F-P reciprocal square root
Floating-point adder instructions	
FADD.P	F-P add
PFADD.P	Pipelined F-P add
FSUB.P	F-P subtract
PFSUB.P	Pipelined F-P subtract
PFGT.P	Pipelined F-P greater-than compare
PFEQ.P	Pipelined F-P equal compare
FIX.P	F-P to integer conversion
PFIX.P	Pipelined F-P to integer conversion
FTRUNC.P	F-P to integer truncation
PFTRUNC.P	Pipelined F-P to integer truncation
PFLE.P	Pipelined F-P less than or equal
PAMOV	F-P adder move
PFAMOV	Pipelined F-P adder move
Dual-operation instructions	
PFAM.P	Pipelined F-P add and multiply
PFSM.P	Pipelined F-P subtract and multiply
PFMAM	Pipelined F-P multiply with add
PFMSM	Pipelined F-P multiply with subtract
Long integer instructions	
FLSUB.Z	Long-integer subtract
PFLSUB.Z	Pipelined long-integer subtract
FLADD.Z	Long-integer add
PFLADD.Z	Pipelined long-integer add
Graphics instructions	
FZCHKS	16-bit *z*-buffer check
PFZCHKS	Pipelined 16-bit *z*-buffer check
FZCHLD	32-bit *z*-buffer check
PFZCHLD	Pipelined 32-bit *z*-buffer check
FADDP	Add with pixel merge
PFADDP	Pipelined add with pixel merge
FADDZ	Add with *z* merge
PFADDZ	Pipelined add with *z* merge
FORM	OR with merge register
PFORM	Pipelined OR with merge register
Assembler pseudo-operations	
MOV	Integer register-register move
FMOV.Q	F-P register-register move
PFMOV.Q	Pipelined F-P register-register move
NOP	Core no-operation
FNOP	F-P no-operation

CC Condition code
F-P Floating-point
LCC Load condition code

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase®
Smarter legal research.