THE INTEL 80860

Superscalar architectures bring a new level of performance to microprocessors

Neal Margulis

Editor's Note: *Earlier this year, Intel announced its first full-scale RISC processor, the 80860. The 80860 operates at up to 50 MHz and has been called a one-chip supercomputer. (See "Intel's Cray-on-a-Chip" in the May BYTE.) We asked Neal Margulis, Intel's chief applications engineer for high-performance processors, to provide us with this in-depth description of the chip.*

The 80860 microprocessor uses an advanced architecture to deliver balanced integer and floating-point performance. With over a million transistors, the 80860 contains a RISC core, an FPU, a memory management unit (MMU), a graphics unit, and separate instruction and data caches.

The RISC core and the MMU allow the 80860 to run multitasking operating systems. Its floating-point capability supports advanced modeling, signal and voice processing, and simulation and CAD applications. The processor provides the computation and display support for three-dimensional visualization. Visualization lets a system display an enormous amount of numeric data as computer-generated graphical images, from which users can easily identify patterns.

Unlike a typical system where these capabilities are spread across several chips, the 80860 was designed from the ground up to integrate all these features. The result is a fully defined architecture for handling integer math, memory management, high-performance floating-point math, and 3-D graphics.

The 80860 eliminates the need for support chips such as floating-point accelerators, vector processors, digital signal processor chips, and graphics coprocessors. A fully defined architecture that starts with the CPU eases the software developer's job and results in more powerful applications software.

Analyzing the Architecture

The major functional units of the 80860 microprocessor and the paths between them are shown in figure 1. By incorporating all the functional units on a single chip, the chip's designers were able to optimize the communication channels between them. The wide internal buses balance data bandwidth with the processing speed of multiple execution units. Separate 32-word register sets for the RISC core and the FPUs provide further support for concurrent execution.

The widespread use of pipelining throughout the chip enhances performance. The RISC core has a four-stage pipeline consisting of fetch, decode, execute, and write stages. The floating-point adder and multiplier also incorporate pipelining with three stages each. In addition, a three-stage load pipeline is matched to the external processor bus, which supports three outstanding cycles.

The RISC core fetches both integer and floating-point instructions from the instruction cache. The CPU allows the programmer to specify two execution modes: single- and dual-instruction.

Single-instruction mode is the traditional execution mode, in which instructions are fetched sequentially. Pipelining allows the sequential instructions to overlap so that multiple instructions are in various stages of completion at any one time.

The 80860 goes a step beyond instruction overlapping with dual-instruction mode. This mode initiates two instructions at once, one for the RISC core and one for the FPU. The FPU achieves one floating-point result per clock cycle and has "dual-operation" instructions, in which an add and a multiply operation execute simultaneously.

Programmers can combine the dual-instruction mode and the dual-operation mode to achieve three operations per clock cycle. With this execution model, the RISC core can execute load, store, and loop-control instructions during floating-point operations.

The result is that peak performance can be maintained while executing the inner loops of common applications. The on-chip data cache or external memory can load data into the floating-point registers.

continued

Graphics

The floating-point hardware of the 80860 processor efficiently performs graphics transformations, including the rotation, scaling, translation, and advanced lighting calculations required for 3-D graphics. Displaying 3-D images requires special operations for shading and for hidden surface removal. The graphics unit hardware speeds these back-end rendering operations (i.e., operations that go from polygons to pixels) using the floating-point registers and wide data paths to operate on multiple pixels simultaneously.

The graphics instructions include intensity interpolation, z interpolation, and z-buffer check. Intensity interpolation allows for smooth linear changes in pixel intensity or color. The z instructions let the programmer determine which objects should be displayed based on their proximity to the viewer. The RISC core also performs a pixel store instruction in parallel with the graphics operations.

Virtual Memory

The 80860 microprocessor supports an address space of 4 gigabytes. The MMU includes a four-way set-associative 64-entry translation look-aside buffer. The TLB translations are performed in one clock cycle and in parallel with the cache accesses.

The MMU implements paged virtual memory management and protection. The Intel 80386 and 80486 microprocessors have the same two-level paging scheme. This commonality allows these processors to interact more easily in a common operating environment, and it facilitates the porting of virtual memory software written in C.

RISC Instruction Set

In the 80860's instruction set, all instructions are 32 bits long and use the three-operand, load/store style typical of RISC processors (see table 1). The three-operand format allows arithmetic, logical, and shift instructions to specify two source registers and a destination register. The only operations that operate on memory are load and store, with arithmetic performed on the registers.

The core unit can execute instructions in one clock cycle. Several techniques allow instructions to execute in one cycle, although their completion may require additional cycles. Loads from memory take one execution cycle, and the next instruction

continued

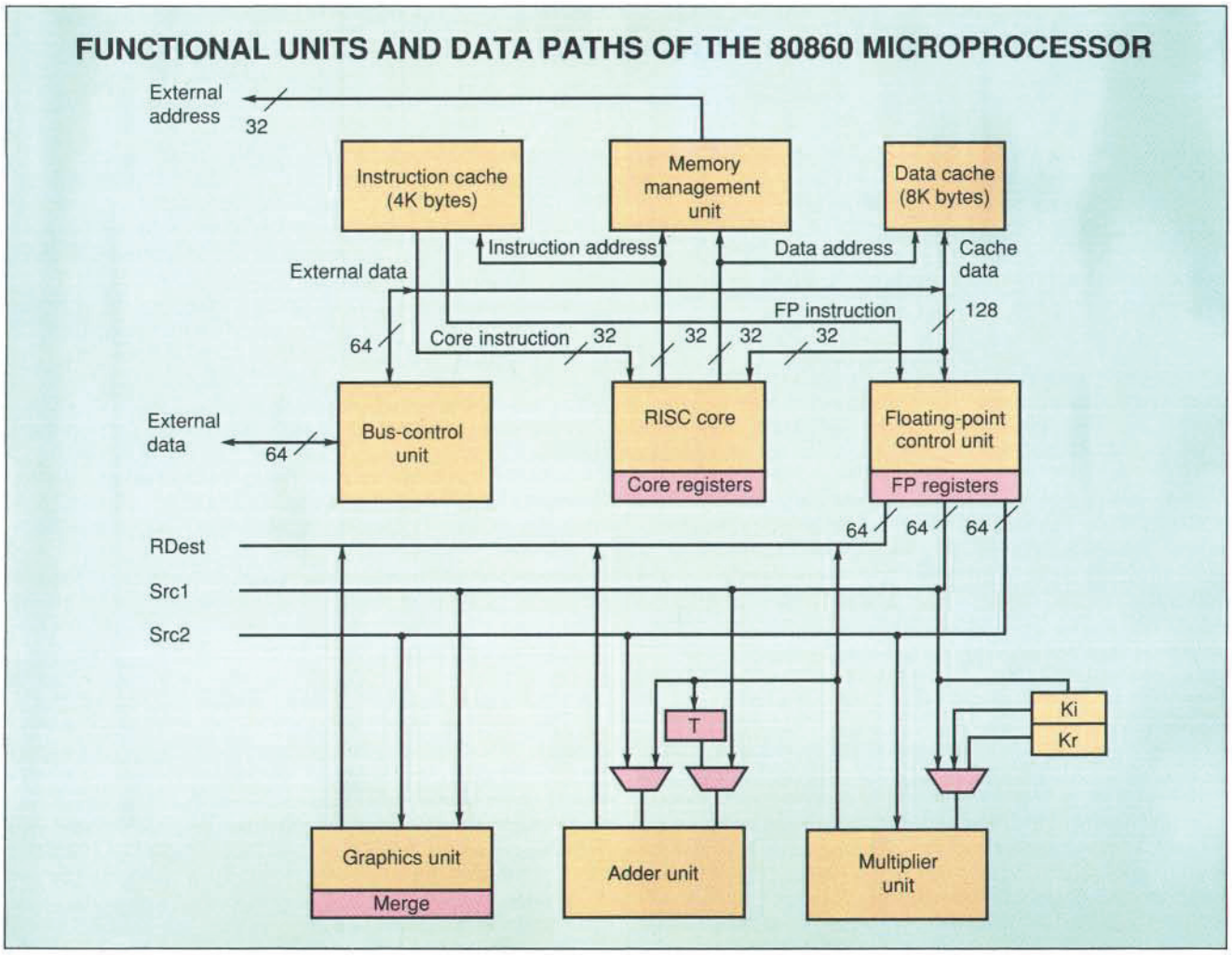

Figure 1: *Wide data buses inside the chip and an external 64-bit bus supply the necessary bandwidth to support multiple operations per clock cycle. Four floating-point registers can be loaded in one clock cycle with the 128-bit path from the data cache.*

Table 1: *The 80860 processor's instruction set has a full set of integer and floating-point instructions, each of which is 32 bits wide. The RISC core performs all the processor's loads and stores.*

THE 80860 MICROPROCESSOR'S INSTRUCTION SET

Core unit

Mnemonic	Description
Load and store instructions	
ld.x	Load integer
st.x	Store integer
fld.y	FP load
pfld.z	Pipelined FP load
fst.y	FP store
pst.d	Pixel store
Register-to-register moves	
ixfr	Transfer integer to FP register
fxfr	Transfer FP to integer register
Integer arithmetic instructions	
ddu	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

FPU

Mnemonic	Description
Floating-point multiplier instructions	
fmul.p	FP multiply
pfmul.p	Pipelined FP multiply
pfmul3.dd	Three-stage pipelined FP multiply
fmlow.p	FP multiply low
frcp.p	FP reciprocal
frsqr.p	FP reciprocal square root
Floating-point adder instructions	
fadd.p	FP add
pfadd.p	Pipelined FP add
fsub.p	FP subtract
pfsub.p	Pipelined FP subtract
pfgt.p	Pipelined FP greater than compare
pfeq.p	Pipelined FP equal compare
fix.p	FP-to-integer conversion
pfle.p	Pipelined FP less than or equal to
famou.p	FP adder move
pfamou.p	Pipelined FP adder move
pfix.p	Pipelined FP-to-integer conversion
ftrunc.p	FP-to-integer truncation
pftrunc.p	Pipelined FP-to-integer truncation
Dual-operation instructions	
fam.p	Pipelined FP add and multiply
pfsm.p	Pipelined FP subtract and multiply
pfmam	Pipelined FP multiply with add
pfmsm	Pipelined FP multiply with subtract
Long-integer instructions	
fisub.z	Long-integer subtract
pfisub.z	Pipelined long-integer subtract
fiadd.z	Long-integer add
pfiadd.z	Pipelined long-integer add
Graphics instructions	
fzchks	16-bit *z*-buffer check
pfzchks	Pipelined 16-bit *z*-buffer check
fzchkl	32-bit *z*-buffer check
pfzchkl	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

Mnemonic	Description
mov	Integer register-register move
fmov.q	FP register-register move
pfmov.q	Pipelined FP register-register move
nop	Core no-operation
fnop	FP no-operation
pfle.p	Pipelined FP less than or equal to

can begin on the following cycle.

The processor uses a technique called *scoreboarding* to guarantee proper operation of the code at the highest possible performance. The scoreboard keeps a history of which registers have pending loads. The actual loading of the data takes one clock cycle if the data is in the cache, or several cycles if it's still in memory.

With traditional microprocessors, the next instruction cannot start executing until the data is returned. Scoreboarding allows execution to continue unless a subsequent instruction attempts to use the register being loaded. In this case, the processor will wait for the data to be returned. Optimizing compilers for the 80860 microprocessor organize the code so that such freeze conditions rarely occur.

The 80860's instruction set includes several control flow optimizations. Programmers can code conditional branch instructions with or without a delay slot. A delay slot allows the processor to execute the instruction after a branch while it is fetching the branch target. With both delayed and nondelayed variations, the compiler can easily optimize the code according to whether or not the branch is likely to be taken.

Branches can be performed conditionally based on the condition-code bit (e.g., `bc`, `bnc`, `bc.t`, and `bnc.t`) or through a comparison of two registers (`bte` and `btne`). There is also a branch-loop-and-add instruction, `bla`, that performs a test, a branch, and an add, all in a single instruction. This instruction reduces program loop overhead.

Floating-Point Instructions

The floating-point hardware can operate on either single- or double-precision IEEE standard 754 floating-point values and on 32- and 64-bit integers. The FPU includes pipelined add and multiply units, which can operate in either scalar or pipelined mode. Source operands for each unit are supplied by the general-purpose floating-point registers, by the special registers (e.g., KR, KI, and T), or by the output of the unit itself.

Scalar-mode instructions specify the operation, the source registers, and the destination register. Once issued, these instructions advance through the unit on each clock cycle until they are completed. Although only one scalar-mode floating-point operation can proceed at a time, it can be overlapped with the execution of RISC core instructions. Scalar-mode execution requires three clock cycles for adds and single-precision multiplies, and four cycles for double-precision multiplies.

Pipelined floating-point instructions advance the three-stage add and/or multiply unit by one stage with each new instruction. This explicit control allows a pipelined floating-point instruction to execute and produce a result with every cycle. Like the scalar-mode instructions, pipelined instructions specify the source registers and the destination register. However, the destination register of pipelined floating-point instructions is the result of a calculation that begins with a prior instruction.

In the example in figure 2, the adder begins by adding the numbers in f2 and f7. Because this is the first instruction of the series and the pipeline is not yet full, the result emerging from the adder is not needed and is sent to f0, which is always 0 and is used as a null destination. On each successive clock cycle, an add instruction is issued to advance the pipeline. When the sum of the first add becomes available at the result stage, it is stored to the destination specified by that instruction.

In this example, three cycles after the f2 plus f7 operation starts, the result is stored to f12 by the instruction that is initiating the addition of f5 and f10. Once all the desired add instructions have started, three dummy adds are used at the end to flush the desired results through the pipeline. When a long series of numbers is added, the overhead of filling and flushing the pipeline is negligible. When only one or two adds are performed, using scalar mode minimizes pipeline overhead.

Parallelism

Dual-operation instructions allow software to perform add and multiply instructions at the same time. One 32-bit instruction initiates both an add and a multiply operation. Although the two operations require six operands (four source operands and two destination operands), the instruction format specifies only three. The use of special registers KR, KI, and T, along with data-path bypassing, provides the additional operands.

The 32 variations of dual-operation instructions specify the source and destination operands for the adder and multiplier. With these instructions, programmers can efficiently implement applications such as fast Fourier transforms (FFTs), graphics transforms, and matrix operations.

The following example shows the acceleration that the use of pipelined operations makes possible. A series of 100 numbers is multiplied by a constant, added to a second series of 100 numbers, and stored:

```
DO 10, I=1, 100 10 X[i]= A[i] * C + B[i]
```

If the add and multiply are executed in scalar mode, a result is produced every six clock cycles. For example,

```
fmul C, A[i], temp fadd temp, B[i], X[i]
```

First the multiply is performed and the result stored in a temporary register. Then the add is performed with the temporary register, and the result is stored. Each multiply and add take three clock cycles, and the six-cycle sequence is repeated 100 times for a total of 600 cycles of floating-point execution time.

With pipelined, dual-operation instructions, the add and

continued

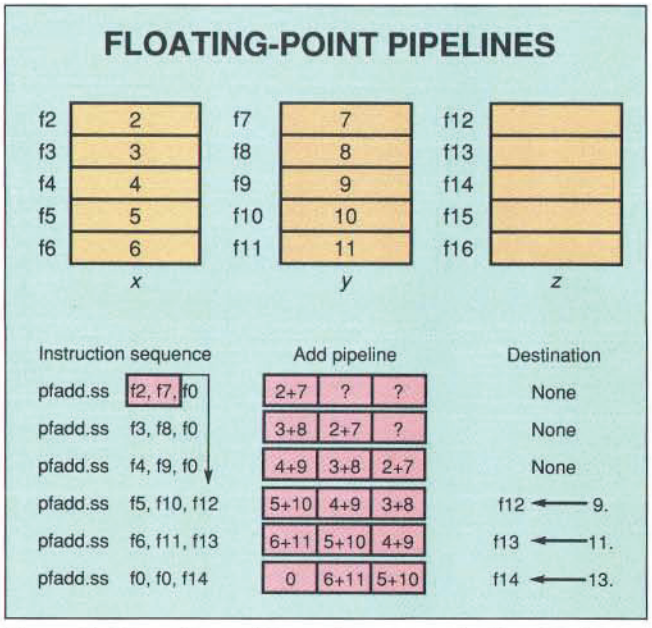

Figure 2: *The adder unit has a three-stage pipeline. Each instruction can start an add operation and store the result from a previous add. Every clock cycle produces a new result. Here the adder implements* z ← x + y.

multiply are performed in parallel and a new result is produced during each clock cycle. For example,

```
r2p1 B[i-3], A[i], X[i-6]
```

This illustrates the `pfam` dual-operation instruction variation specified as `r2p1 src1, src2, rdest`. The instruction specifies that the multiply is initiated with the constant register KR and `src2` as the operands, that the add is initiated with the result from the multiply and `src1` as the operands, and that the result from an earlier add is stored to `rdest`.

Because the three stages of the add and multiply pipelines are chained in series, the result comes from the operation that began six clock cycles previously. By overlapping the multiply and add, the loop of 600 cycles in scalar mode is reduced to 100 cycles of floating-point execution time.

The FPU's ability to produce new results every clock cycle gives it a tremendous appetite for data. To provide this data, the RISC core can operate in parallel with the FPU to move data in and out of the floating-point registers and to provide program flow control. The processor enters dual-instruction mode if you specify a `d.` prefix in the floating-point instruction mnemonic.

Once in dual-instruction mode, the instruction sequence consists of 64-bit aligned instruction pairs. The upper half contains the integer instruction, and the lower half contains the floating-point instruction. In dual-instruction mode, the 64-bit-wide instruction cache allows the execution of a pair of instructions every cycle. The modes can be changed with no overhead for any number of instructions.

By taking advantage of the 128-bit data-cache path, the RISC core can load up to four floating-point registers per cycle with the `fld` instruction. Also, a special load instruction called `pfld` loads the floating-point registers from external memory without the data being placed into the cache. This unique instruction allows data that will be referenced only once to be loaded from external memory, while data that is being continually referenced can be kept in the on-board cache.

Like the floating-point execution units, the `pfld` uses a three-stage load pipeline. This instruction specifies a load address and a destination register. Each `pfld` advances the load pipeline one stage and stores the result from the load address that began three instructions previously. Auto-increment addressing avoids using a separate add instruction by automatically incrementing the base register before each load.

Performance Evaluation

One common measure of integer performance is millions of instructions per second. At 40 MHz, the 80860 processor delivers 33 VAX MIPS (based on the Stanford integer benchmark suite) and performs 85,000 Dhrystones on version 1.1 and 80,000 on version 2.1. These results illustrate the RISC core's optimized instruction set and its ability to execute most instructions in one clock cycle. Combined with efficient memory management, the 80860 processor performs well in large applications that use virtual memory.

The Whetstone is a common benchmark used to gauge scalar floating-point performance. The 80860 processor achieves 24 million Whetstones at 40 MHz. The peak million-floating-point-operations-per-second ratings for the 80860 are 80 MFLOPS single-precision and 60 MFLOPS double-precision.

Although peak performance numbers are often misleading,

continued

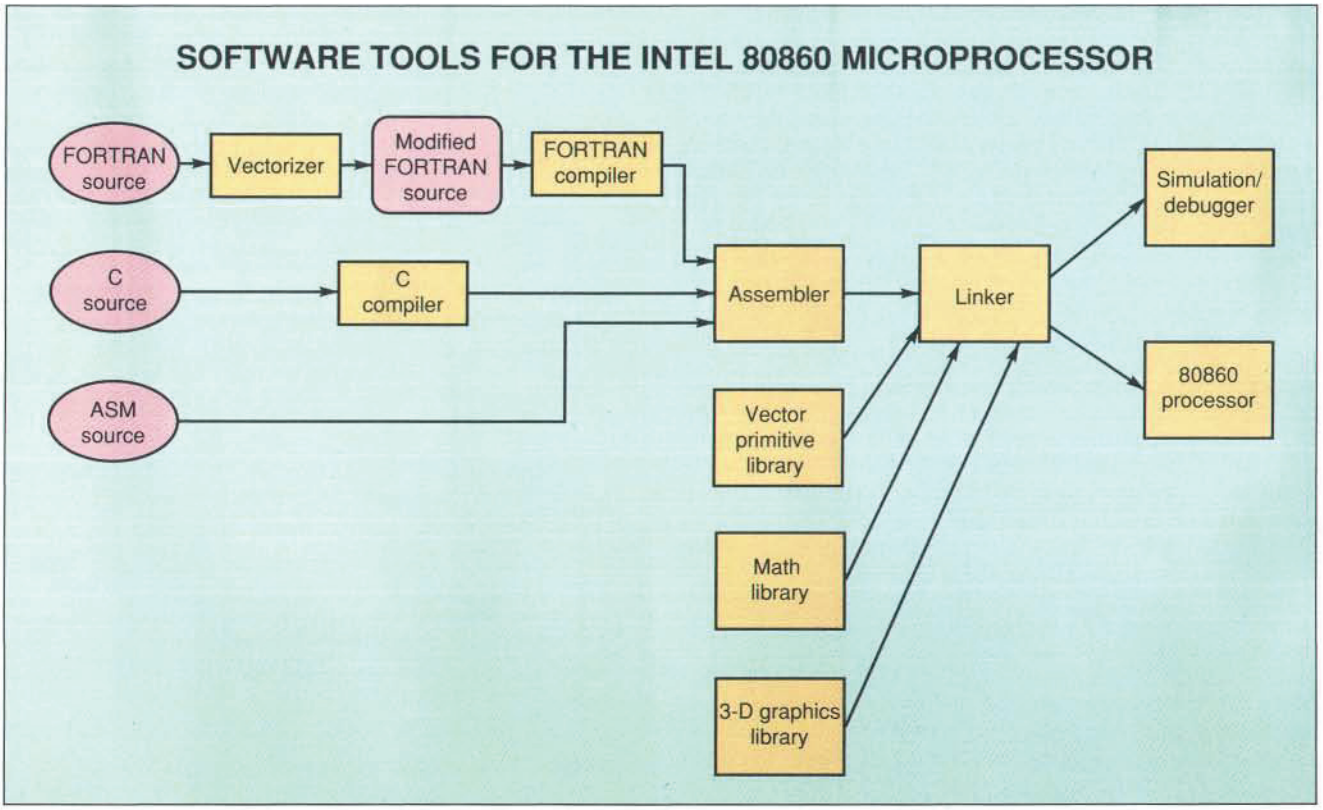

Figure 3: *Scientific FORTRAN applications take advantage of the vectorizing precompiler that automatically calls the vector library. C programs also can call the libraries.*

dual-instruction mode and dual-operation mode allow the microprocessor to achieve peak performance for inner loops of common matrix operations. In the LINPACK benchmark, the 80860 attains over 10 MFLOPS. The inner loop of a complex FFT requires 10 floating-point operations. The 80860 performs the 10 operations in six clock cycles, calculating a 1024-point FFT in 1 millisecond.

Tools of the Trade

Intel's internal development teams and independent vendors are providing a full complement of software development tools and operating systems for the 80860. Figure 3 shows the development tools' environment, including C and FORTRAN compilers, an assembler/linker, a simulator/debugger, and a FORTRAN vectorizer. In addition, there are the mathematical, vector primitive, and 3-D graphics libraries.

The initial development environments use cross compilers hosted on Unix System V/386 and OS/2. The optimizations used in the compilers include coloring for register allocation, register-based parameter passing for calls, interblock common subexpression and loop invariant elimination, constant propagation, strength reduction, extensive peephole optimizations, and instruction scheduling.

Programmers write much of their engineering and scientific applications in FORTRAN because it is so well suited for vectorization. The 80860 support includes a FORTRAN vectorizing precompiler. Vectorization is performed on DO and IF loops, outer loops, and forward-branching conditional operations. The precompiler recognizes these structures and generates calls to a set of highly optimized, hand-coded procedures. These procedures take full advantage of dual-instruction and dual-operation modes, managing the data cache as a vector register.

In addition, programmers can access these procedures from other high-level languages. Work is under way to further increase the degree of parallelism of the processor. A library of assembly language routines for scalar mathematics is also available.

A multiprocessing version of Unix System V 4.0 is under development for the 80860. The project is a joint effort by AT&T, Unisys, Intel, Olivetti, Prime, Okidata, and others. The Intel programming tools will maintain high-level-language source code compatibility between the 80386, 80486, and 80860 microprocessors. A document is available that specifies an application's binary interface standard for the 80860 to allow portability of applications software across systems from different companies.

A Supercomputer on Your Desk?

The 80860 delivers balanced integer and floating-point performance that only a million-transistor processor can provide. Software developed for the 80860 can take full advantage of the architecture, bringing the power of supercomputers into the hands of desktop computer users.

But in addition to providing a new level of performance, the 80860 is the first of a new class of microprocessors. By breaking the one-operation-per-clock-cycle barrier, it has become the first commercial superscalar microprocessor. The advent of superscalar architectures will allow microprocessor performance to work alongside advances in semiconductor technology to bring even greater capabilities to desktop computers. ■

Neal Margulis is chief applications engineer for high-performance processors at Intel Corp. in Santa Clara, California. He can be reached on BIX c/o "editors."