# Design and Implementation Trade-offs in the Clipper C400 Architecture

**The C400 is the first complete redesign of the Clipper reduced instruction-set computing architecture since its introduction in 1985. The C400 delivers three times the performance of the C300, yet retains full-code compatibility with earlier Clippers. A combination of superscalar and superpipelining techniques effectively exploits instruction-level parallelism in code.**

***Howard G. Sachs***

***Harlan McGhan***

***Lee F. Hanson***

***Nathan A. Brookwood***

*Intergraph Corp.*

From the introduction of the first C100 architecture in October 1985 to the new C400 introduced in September 1990, three ambitions have motivated Clipper development. First, the microprocessor must be complete and highly functional. Second, it must achieve the highest possible performance. Third, it must accomplish the previous objectives at the lowest possible cost. Pursuit of these diverse goals meant facing a series of trade-offs: balancing the virtue of simplicity against the convenience of functionality, maximum functionality against the benefits of integration, the highest levels of integration against the requirement for performance, and ultimate performance against the value of cost-effective implementation.

Earlier Clippers, including both the C100 and the C300 (introduced in November 1987), emphasized a high degree of instruction-set functionality, together with highly integrated functional units in silicon. To attain these objectives with mid-1980s technology, we accepted various limitations on performance. The new C400 (see Figure 1) preserves the programming model of the earlier generation. But, since today's silicon technology forces far fewer perfonmance compromises on the implementation of this model, the C400 achieves far higher performance. Our performance simulations suggest the C400 will attain a SPEC ratio of 41 (see box).

The C400 combines two architectural approaches to attain its performance goals. The first approach, superscalar operation, allows the processor to begin the execution of more than one instruction during each clock cycle. A conventional, nonsuperscalar machine can start only one instruction per cycle. At 50 MHz, such a machine can start (or complete) at the most 50 million instructions each second. An $n$th order superscalar machine can start $n$ instructions during each clock cycle, and thus has a peak performance $n$ times that of the conventional approach. Johnson[1] provides an excellent overview of superscalar principles.

We characterize the C400 as moderately superscalar: it can dispatch two instructions per clock cycle, with each instruction coming from one of two broad classes. One class contains all load and store operations (such as memory reference instructions), as well as control, logical, and fixed-point arithmetic instructions. The other class includes floating-point instructions. This structure fits easily into the original Clipper architecture and yields a significant performance boost in the floating-point-intensive applications that predominate in technical markets. The C400 is also opportunistically superscalar; it does not replicate function units to minimize class conflicts, and it issues only one instruction per clock cycle when sequential instructions fall into the same class.

In addition to superscalar operation, the C400 embodies the design concept of superpipelining,



0272-1732/91/0600-0018$01.00 © 1991 IEEE

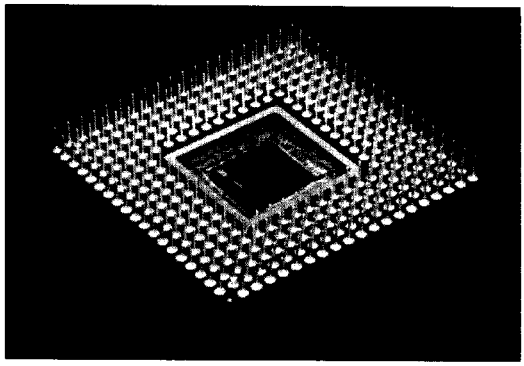

Figure 1. C400 CPU chip, mounted in a 299-PGA package.

We use this relatively new term, coined by Jouppi,[4] to describe an approach that emphasizes high clock rates and deep execution pipelines in attaining high computational performance. For example, the C400 pipelines access the cache subsystem, enabling the CPU (see Figure 2) to issue a steady

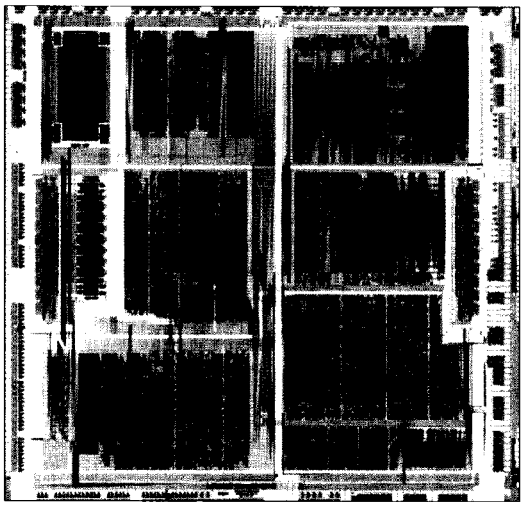

Figure 2. The C411 CPU.

## SPEC benchmarks

Computer system vendors have long struggled with the problem of measuring and comparing the performance of the systems they sell with the performance of other systems. Early solutions to this problem, such as characterizing systems in terms of MIPS (million instructions per second) or Mflops (million floating operations per second) became increasingly meaningless due to the diversity of computer architectures. This diversity caused different machines to require the execution of vastly different numbers of instructions to calculate the same results. Also, the programs that measured these values were small, and vendors could often "tune" their systems to perform better on these benchmarks than on the actual programs.

Finally, in November 1988, four vendors (Apollo Computer, Hewlett-Packard, Mips Computer Systems, and Sun Microsystems) and *Electronic Engineering Times* (an industry journal) formed the Systems Performance Evaluation Cooperative (SPEC). SPEC's charter was to "establish a suite of standard performance benchmarks for the measurement and characterization of high-performance computers."

In 1989, SPEC issued its benchmark suite—a set of 10 programs that represent the types of tasks technical computer users often encounter. Unlike the earlier programs that measured MIPS and Mflops, the programs in the SPEC suite were designed to more accurately represent the work load users will place on the machines they purchase. This increased accuracy should make it more likely that the performance reported by the benchmark will correlate with the performance observed in actual use. The suite includes four programs that exercise the system's capabilities in logical and fixed-point operations and six numerically intensive programs that measure floating-point performance.

Each program's performance, referred to as its SPEC ratio, is reported as the ratio between the execution time on the system in question and the corresponding execution time on a Digital Equipment Corp. VAX-11/780 computer. SPEC coined the term *Specmark* to refer to the geometric mean of all 10 SPEC ratios. A machine rated at 40 Specmarks can (on average) be considered to run 40 times faster than the reference VAX system. Some programs will likely exceed this ratio, while others will fall short of it.

The concept of using Specmarks to compare system performance is being explored within the computer industry. Many vendors have ceased entirely using the metrics of MIPS and Mflops. The SPEC consortium continues to evolve its suite of test programs and expects to release a new revision later this year that measures a larger set of system capabilities.

stream of load instructions without causing stalls while the cache accesses data. Similarly, the CPU can issue successive floating-point multiply (or add) instructions on every clock cycle without stalling the execution pipeline. This capability provides a level of floating-point performance in vector codes that previously required the use of digital signal processing arrays.

## Programming model

Clipper's programming model makes it one of the more complex reduced instruction-set computing designs. RISCs often include fewer than 100 instructions, but Clipper provides more than 160 instructions. Most RISCs use fixed-length, 32-bit instructions, but Clipper uses a Cray-like, variable-length scheme. Simple, register-oriented instructions occupy only 16 bits. More complex instructions can require 32, 48, or 64 bits. Most RISCs provide only two or three different ways to address memory, but Clipper provides nine different addressing modes, including absolute, relative, PC-relative, and indexed addressing.[4-6]

The combination of a Cray-like, instruction-set encoding and a multiplicity of addressing modes results in a more compact object code on the Clipper than on other RISCs. In an unpublished report, Alan J. Smith of University of California, Berkeley's Computer Science Department analyzed the size of emitted code for standard benchmarks in several architectures. Smith measured the text portion of the object files and normalized his results to the code size produced on a VAX running BSD 4.3 Unix. A summary of his results appears in Table 1.

Every 16- or 32-bit instruction on Clipper saves 16 bits over the equivalent 32-bit instruction or pair of 32-bit instructions on another RISC machine. Even in this era of relatively inexpensive dynamic RAM main memories, code density still plays a role in system cost and performance. High-density code improves both the effectiveness of the instruction cache and the available bus bandwidth.

**Table 1. Size of emitted code for standard benchmarks in several architectures.**

| | Architectures (hardware/operating system) | | | |
|---|---|---|---|---|
| Standard benchmarks | Clipper/ CLIX 3.1 | Sun-4/200/ Sun OS 4.0 | Mips R2000/ N/A | HP9000/ HP-UX |
| No optimization | 1.12 | 1.94 | 2.15 | 1.63 |
| Optimization O1 | 0.99 | 1.73 | 1.65 | 1.05 |
| Optimization O2 | 0.98 | 1.41 | 1.28 | N/A |

## Early hardware implementations

We weighed the advantages of Clipper's high-code density against the costs paid for this attribute on the C100 and C300. Neither machine included a delayed branch instruction, largely because of difficulties associated with parsing variable-length instructions. Consequently, every branch caused a pipeline stall and adversely affected performance (especially in tight loops). The variety of addressing modes complicated effective address generation for loads and stores. Since the C100 and C300 employed the integer ALU for address generation, use of the complex addressing modes on these machines tied up the ALU and slowed the flow of integer instructions through the execution pipeline.

Processor logic in the C100 and C300 was partitioned into three devices. The CPU chip both handled integer and floating-point operations. We provided separate cache and memory management chips (CAMMUs), one each for instructions and data. The integration of CPU and FPU functions on one chip required a large die—more than 500 mils on a side. Even with a die this large, space was still very tight at the 1.5-μm geometries used in the C300, forcing critical design compromises. In particular, of the two main execution units, we only pipelined the integer unit; the FPU was left as a standard, unsegmented flow-through unit. The single-threaded design of the floating-point logic caused operations with short latencies (like adds) to back up behind ones with very long latencies (like divides), and thus slowed performance. The chip's 32-bit data paths limited the rate at which operands could be loaded into the 64-bit register file; the data paths also reduced double-precision performance.

Combining cache and MMU functions on one chip imposed a serious compromise in cache size; a handicap that was only partly offset by good code density and a two-way, set-associative cache design. On large programs, a small cache usually meant a high miss rate. Having the CPU and cache on separate chips meant accesses to the cache took multiple clocks. Since limited silicon resources precluded pipelining cache accesses, back-to-back loads or stores introduced wait states, even for cache hits.

Of course, it is necessary to maintain perspective. In the 1990s, it is easy to regard the instruction cache used in the C100 and C300 as small and slow. It is important to remember that microprocessors have evolved rapidly over the past decade. The first commercial 32-bit microprocessor was not introduced until 1982. At the time of its introduction in 1985, the C100's 4-Kbyte, three-clock cache was relatively large and fast; it was a key to the performance attained by this machine.

The C100's hardware strengths unexpectedly led to a weakness related to software. Its compilers paid little attention to instruction sched-

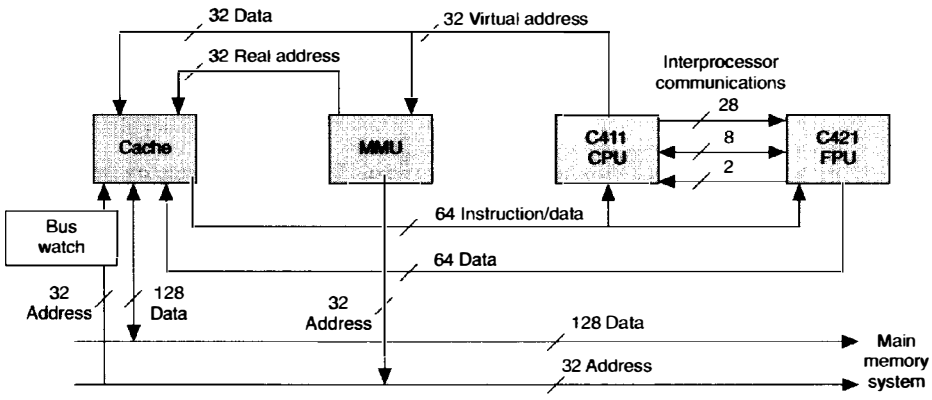


**Figure 3. Diagram of the C400 system processor.**

uling or the sequencing of calculations, because the CPU's hardware scoreboard could stall the CPU when the result of a previous operation was not yet available. This inattention to code scheduling did not affect the accuracy or the ability to repeat computations, but it did affect execution time, degrading performance in both benchmarks and real applications.

(Fortunately, in the summer of 1988, coinciding with the start of the C400 hardware redesign, Intergraph launched intensive software efforts to produce a new set of optimizing compilers and a performance-tuned operating system kernel for Clipper. Baxter and Arnold[7] present some of the results of this ongoing software effort in the area of compilers.)

## C400 project overview

We recognized that no amount of improvement in process or software technology alone could compensate for the performance constraints forced on these earlier products. In the fall of 1988 the Clipper design team set out to reimplement the hardware architecture completely. Clipper designers wanted to preserve binary software compatibility with the millions of lines of application code compiled for the C100/C300 since 1985. The goals for the C400 program were to:

- attain a performance level in excess of 40 Specmarks. This goal implied improvements of a factor of three in integer performance, and a factor of six in floating-point performance, relative to the earlier C300;
- minimize development time and risks, producing first silicon by the start of 1991;
- keep volume production costs low; and
- provide an upwardly compatible binary environment, enabling customers to migrate at their own pace to systems based on the new chip set.

The C400 system addresses the performance limitations of the earlier Clippers, and accomplishes the goals outlined above. The processor, as illustrated in Figure 3, includes four major elements:

- The C411 integer unit (CPU) decodes and issues all instructions, and executes integer operations. Packaged in a 299-pin ceramic pin grid array, the unit contains approximately 160,000 transistors on a die measuring 253,000 square mils.
- The C421 FPU incorporates the floating-point register file and the floating-point execution pipelines. Packaged identically to the CPU, the FPU contains approximately 140,000 transistors.
- The MMU handles virtual-to-physical address translation, using translation look-aside buffers stored in discrete static RAMs.
- The cache unit provides a high-speed (one clock cycle), 128-Kbyte, direct-mapped cache that supports the CPU's bandwidth requirements for instructions and data. Discrete SRAMs store cache data and tags.

The system uses 64-bit data paths to link the CPU, FPU, and cache. One transfer from the cache to the CPU's instruction buffer contains up to four variable-length instructions. Double-precision data move from the cache to the FPU's register file in one clock cycle; in general, there is only a minor performance penalty associated with the use of double-precision arithmetic, compared with single-precision timings.

The C400's superpipelined load and store operations permit the CPU to execute sequential load operations without pipeline stalls on a sustained basis. However, this execution occurs only as long as the system cache contains the contents of the

*continued on p. 74*

## Clipper C400

*continued from p. 21*

memory locations referenced. The CPU pipelines all accesses to the cache and main memory system. During a given clock cycle, the CPU generates a new virtual address, the MMU translates the previous virtual address, and the cache accesses the physical address calculated in the previous clock cycle. Although the CPU coordinates all transfers between the cache and CPU or FPU, floating-point data moves directly between the cache and the FPU register file; it does not pass through the CPU.

The initial version of the C400 implements MMU and cache functions using discrete elements, but plans call for a future version to incorporate a custom VLSI cache/MMU mechanism. We will package these forthcoming CAMMU chips with the CPU and FPU chips on a multichip module that behaves in most regards like a large single chip. The controlled impedance of the multichip module, along with the shorter signal paths possible in this configuration, will permit operation at clock speeds far in excess of the 50-MHz level planned for the discrete cache version. This multichip module product will possess the same high level of functional integration as the current C300 module, but with far greater performance in a much smaller package.

### C400 performance

The Clipper design team used two major strategies to maximize performance and attain its performance goal of 40 or more Specmarks. First, it targeted high clock rates (for a CMOS-based device) of 50 MHz and beyond, with the expectation that this performance would benefit all classes of applications. Second, the team planned to achieve a low-average-clocks-per-instruction metric via a combination of superscalar dispatch, improved cache bandwidth, and a multiplicity of execution pipelines for integer and floating-point operations.

The design team observed that instruction runs (that is, sequential instructions terminated by a transfer of control operation) tend to be short for integer-oriented codes (like compilers). They are longer for the scalar floating-point codes that characterize many electronic design automation applications, and longer still for the vectorizable codes used in many mechanical CAD applications. Clipper's principal customer, Intergraph, emphasizes these latter applications. We decided to optimize vectorizable code via a highly pipelined FPU coupled to a high-bandwidth pipelined MMU and cache subsystem. As the subsequent discussion of the Linpack Daxpy routine shows, the C400 achieves a more-than-respectable result on this class of code.

Extensive simulation of the C400 chip set and an associated 128-Kbyte, direct-mapped, unified cache suggests the product will attain an overall SPEC ratio of 41. As shown in Figure 4, C400 tends to excel in applications conducive to pipelining techniques. However, it turns in a respectable level of performance even in applications where short instruction runs force pipeline stalls. There remains the possibility that actual performance may not match the simulated results shown in Figure 4, but we have attempted to be conservative in the assumptions that drove our simulations.

The Linpack benchmark provides a good illustration of the

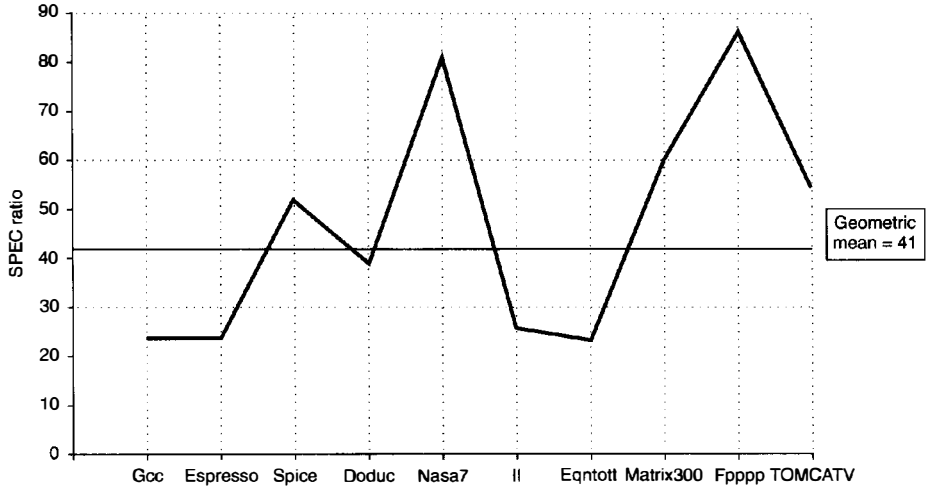


Figure 4. Graph of SPEC ratios for the 50-MHz C400 based on simulation studies.

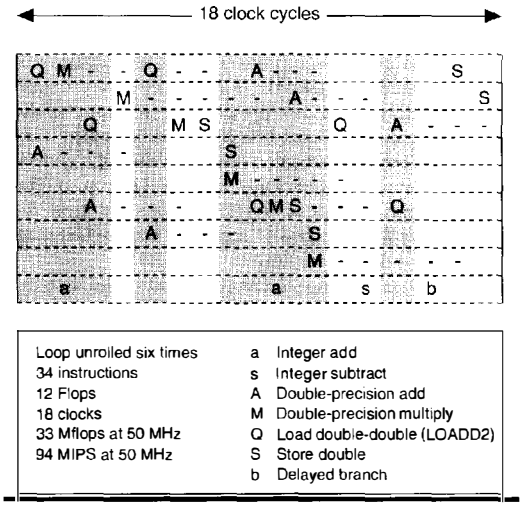


**Figure 5. Sequence of C400 code scheduling for the Linpack Daxpy. Shaded areas indicate the clock cycles in which multiple instructions issue.**

C400's vector performance. Daxpy, its well-known inner loop, includes a sequence of five operations: load, load, multiply, add, and store. Loads and stores dominate this loop, comprising 60 percent of the operations. Unrolling the loop increases the considerable instruction-level parallelism present in this algorithm even more. A hand-scheduled code for the C400, which has 34 instructions, unrolls the loop six times.[8]

The superscalar feature of the C400 exploits the parallelism inherent in this code segment by issuing a load or store instruction concurrently with each floating-point multiply or add. A new instruction in the C400, LOADD2 (load double floating double), plays a key role in increasing instruction issue bandwidth by combining two sequential 64-bit load operations in one instruction. The result is that the C400 can execute these 34 instructions in only 18 clocks. Therefore, the clocks-per-instruction rate for this piece of code is 0.53 with a corresponding rating of 94 native MIPS and 33.3 Mflops.

As the Daxpy case shows, the capability of issuing floating-point loads and floating-point operations in the same clock cycle makes the C400's superscalar capabilities far more effective than they might at first appear. Figure 5 details the code scheduling for this vector loop. In essence, the C400 calculates and stores six array elements, based on 12 input values (a total of 18 memory accesses) in just 18 clock cycles.

The execution of the arithmetic and loop control instructions (including the updating of address pointers and testing for end conditions) overlaps completely with the data loads and stores. The time needed to execute the former is completely hidden by the time used for the loads and stores; it adds nothing to the overall time required to execute the program. It is impossible to surpass the performance attained within this inner loop in a uniprocessor design with one 64-bit path between the CPU and cache. Subsequent performance improvements can only be attained via higher clock rates, wider paths between cache and CPU, or multiprocessor techniques.

## Integer unit design

The design of the C400 processor includes nine distinct pipelines that handle loads, stores, branches, and a variety of integer and floating-point execution elements (see Figure 6). To support the high degree of concurrency within the CPU, the chip incorporates a 32 × 32-bit register file with three read and two write ports. This custom-designed file uses an advanced circuit design that provides a 6-ns access time in a 1-μm-CMOS process technology.[9]

The integer unit fetches, decodes, and issues all instruc-

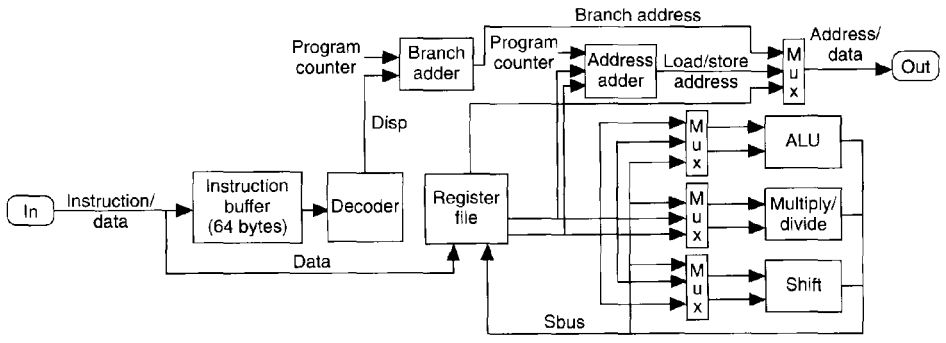


**Figure 6. Diagram of the C400 integer unit.**

tions, and manages all but the floating-point execution pipelines. The CPU also supervises the execution of floating-point loads and stores. It signals the FPU regarding the appropriate register to load or store, but treats the operation like an integer operation in all other regards. The C400's superscalar dispatch logic allows the CPU to start a floating-point load (or store) and a floating-point operations instruction in the same clock cycle, if there are no data dependencies in the instructions to be issued.

## Load/store pipeline

The load pipeline normally requires one clock cycle to generate an effective address, another to translate this address, and a third cycle to access the cache and present the cached data to the CPU or FPU. To eliminate the bottleneck in the C100/C300 caused by using a common ALU for both address generation and all integer operations, the C400 design uses a dedicated address adder in the load and store pipelines. Upon arrival at the CPU or FPU, the data can write to the register file or bypass to the appropriate functional unit. The overall flow appears in Figure 7.

Just as the dedicated address adder overcomes the delays caused in the C100/300 by Clipper's complex addressing modes, so the new branch pipeline adds a delayed branch instruction. This new instruction represents a major enhancement to the Clipper instruction-set architecture, and is also one of the few changes visible to application programs. While it is therefore necessary to recompile programs to take advantage of this performance enhancement, recompilation is not required simply for the sake of continued correct execution of existing binaries. C400 preserves the semantics of the old branch and conditional branch instructions. The new compare-and-branch instruction tests the value of a general register and branches accordingly with two delay slots. This approach avoids the use of condition codes and gives the compiler more flexible choices regarding code scheduling. Figure 8 shows the overall flow.

The integer unit contains an ALU, a 32-bit barrel shifter, a Wallace-tree multiplier, and an integer divider. Most arithmetic and logical operations execute in one clock cycle, and thus do not present any special problems regarding resource management or instruction scheduling. Multiplication and division operations, though, require a variable number of cycles to complete based on the values of the operands. Because these integer operations are not superpipelined, the instruction issue logic cannot issue a second instruction of the same type until the first one completes. To minimize lost cycles, the issue logic and the multiply/divide function unit communicate via a simple protocol that lets the function unit examine the arguments and inform the issue logic how long the operation will take. For most integer and logical operations, the basic flow follows the classic RISC decode/execute/write-back model, and the unit can sustain a clocks-per-instruction rate close to 1.0 once the program and data reside in the system's cache.

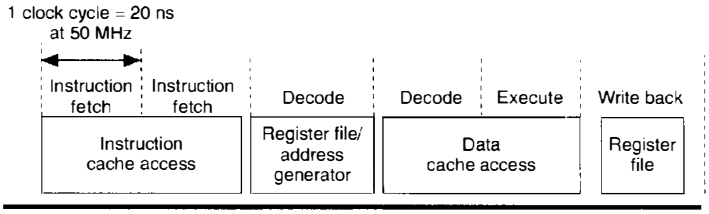


Figure 7. Example of the C400 load pipeline.

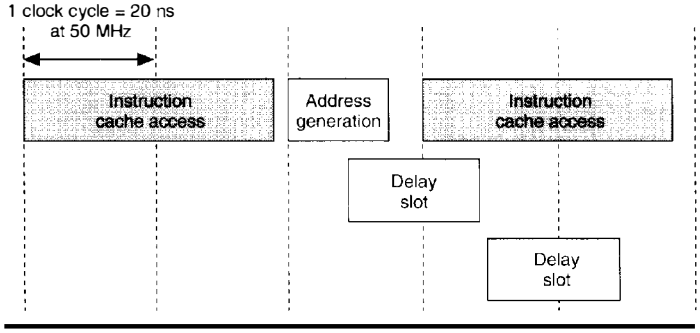


Figure 8. C400 delayed branch pipeline.

## FPU design

The FPU contains separate execution units for addition/subtraction, multiplication, and division. A 16 × 64-bit register file with three read and two write ports holds floating-point data. Like the CPU's register file, the register file can be accessed in less than 6 ns. The integer unit decodes all floating-point instructions and handles all the address calculations and memory operations involved in floating-point loads and stores. Thus, most of the real estate on the FPU chip itself is free for the three execution units that handle floating-point addition/subtraction, multiplication, and division.

The FPU's superpipelined design trades off the number of pipeline stages against the number of logic levels within each stage. This design allows the processor to run at a much higher clock rate than otherwise possible, given typical 1-

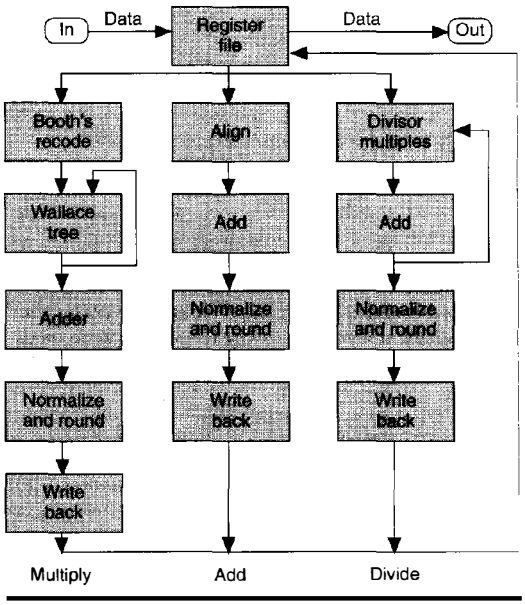


Figure 9. Diagram of the C400 floating-point unit.

μm-CMOS gate delays. The higher clock rate creates more opportunities to issue instructions and improves performance.

The key to the FPU's performance is its capability of issuing instructions to the same function unit on every clock cycle. (Double-precision multiplication operations require a one-clock interval between back-to-back instructions; a one-clock stall occurs when the compiler places two 64-bit floating-point multiplies in adjacent issue slots.) Floating-point division does not occur with sufficient frequency to justify the expense of a pipelined divider; the issue logic will stall the processor when the compiler schedules a second divide operation prior to the completion of an earlier one (see Figure 9).

The C400 instruction issue logic does not issue instructions out of order, and it will delay the issue of any instruction until all the resources needed for its execution are available. But, given the wide range of execution times for floating-point instructions (a 32-bit add takes four clock cycles and a 64-bit divide takes 30), concurrently executing instructions can finish out of order. This situation creates an interesting problem for the compilers that must track the sequence in which computations complete, if they are to provide optimal code scheduling. It creates an even more challenging problem for the handlers that deal with the variety of IEEE Standard 754 floating-point exception traps that can occur during normal program execution.

For example, suppose a program starts a 64-bit division (a 30-clock operation), proceeds with several additions and multiplications, and then discovers a floating underflow condition on the division. The trap handler will sort this out, match the error to the offending instruction, and present the results to the application programmer in a manner consistent with in-order instruction issue and in-order instruction completion. Some IEEE implementations do this simply by disabling out-of-order completion; for example, they serialize instruction issue and give up any opportunities to exploit parallelism within the code. Other implementations examine the operands at the start of each operation and signal a trap when there is a possibility a trap might occur. This approach increases overhead as the software processes these potential problems.

The C400 FPU achieves IEEE compatibility without sacrificing performance or accuracy. At the start of each floating operation, the FPU stores the program counter and source operands in a set of storage registers known as the floating trap queue. As operations complete, their entries are removed from the queue. When the hardware encounters an IEEE exception, the contents of the floating-point trap queue are frozen and an operating system routine untangles the situation. This solution is both easy and unintrusive to implement in hardware, since the floating-point trap queue largely replicates the floating register file and occupies less than 3 percent of the die. The queue is also relatively painless in performance, since when traps do not occur, there is no penalty, and when they do, the penalty is low.

## Superscalar/superpipelined architecture

The two strategies of providing superpipelined execution units and superscalar dispatch of instructions to multiple functional units are often viewed as redundant in principle since both exploit instruction-level parallelism in code.[10] In practice, the two capabilities are highly complementary. In vector codes characterized by a high degree of instruction-level parallelism, the combination of floating-point pipelines that can execute one instruction per cycle, and superscalar issue of two instructions in the same clock, is extremely effective (as the Daxpy example described earlier illustrates).

Combining the two confers the capability of exploiting instruction-level parallelism at little expense. Returning to the example of Daxpy, to attain the same 33 Mflops of performance on this code as the superpipelined/superscalar 50-MHz C400, a single-issue superpipelined machine would have to run at 100 MHz. Alternately, a pure superscalar machine operating at 25 MHz would have to issue four instructions in a one clock cycle, including two loads or two stores.

Following either of these pure routes is difficult. The extremely deep pipelines needed for 100-MHz operation introduce long latencies everywhere in integer and floating-point code alike. Since integer code generally has less instruction-level parallelism than vector floating-point code, the cost of superpipelining the integer execution units will generally go

**Table 2. Superpipelined execution unit latency in the C400.**

| Instruction | Issue rate (single/double) | Execution latency (single/double) |
|---|---|---|
| Floating-point unit adder/subtracter | 1/1 | 4/4 |
| FPU multipler | 1/2 | 5/6 |
| Load pipeline | 1/1 | 2/2 |

unrewarded. Higher clock rates complicate circuit design issues, magnify the effect of clock skew, and increase heat dissipation.

The alternate pure path, namely, building a superscalar microprocessor that can issue four instructions in one cycle, requires the construction of dual load/store pipes. Multiple paths to and from memory are both expensive and pin-intensive. Also, the decode circuitry required to parse four instructions in parallel may lengthen the instruction decoding stage and thus limit the processor clock rate. This forces the entire machine is then forced to operate at lower (less efficient) frequencies than otherwise possible.

By contrast, the C400 avoids these problems. Its integer ALU sets the clock rate by the speed of its fastest operations. Consequently, integer operations have a one-cycle issue rate and a one-cycle latency. Slower functional units in the critical path of instruction issue, including the load pipe, the floating-point adder/subtracter and multiplier, are superpipelined to match this clock rate (see Table 2) The 50-MHz clock rate does not tax contemporary CMOS technology and leaves room for clock-rate scaling with future improvements in process technology and smaller geometries.

The superscalar dispatch employed by the C400 is strictly opportunistic; instructions are issued simultaneously only to different types of functional units. No functional units are replicated solely to allow the simultaneous issue of instructions of the same type. Such replications increase cost and complexity for more than performance, particularly since the majority of cycles provide no opportunity to use these replicated execution units.

Decoding two instructions during one cycle poses no timing problem, fits well with the 64-bit buses needed (anyway) to handle double-precision data, and does not represent a real or potential hazard to the maximum clock rate sustainable by the C400. Johnson[1] discusses the trade-offs between two-instruction and four-instruction decoders, and concludes that the latter yields a 20-25 percent performance improvement at any given clock rate. The C400's use of a two-way decoder and a fast clock boosts performance by far more than 25 percent, and avoids the instruction-fetch and branch prediction problems inherent in the more complex order superscalar implementations.

## Circuit design

We designed C400 VLSI components using a standard cell method with a custom-designed register file and I/O pad ring, and a 1-µm-CMOS process with two layers of metal. The short design cycle, combined with the desire to minimize both risks and costs during the development phase, mandated the use of a standard cell approach for most of the logic design. The high bandwidth requirements needed to support concurrent operations by the CPU and FPU dictated the use of a custom memory design for the register file. Since the Intergraph Advanced Processor Division depends on external foundries for wafer production, the choice of a 1-µm process seemed a conservative one for the 1990-91 time frame, when we planned to go into production with the C400. Surprisingly, we found ourselves closer to the state of the art in this regard than we expected.

Our desire to spread the design over several moderately sized chips rather than pack it all on one die, led to the need for innovation with regard to interconnection circuitry. Along with interconnection delays, power dissipation had to be minimized. Dissipated power can be calculated by the formula: $P = CV^2f$ where $P$ is the dissipated power, $C$ is the capacitance of the circuit, $V$ is the voltage swing, and $f$ is the driving frequency. This formula reveals that much of the power dissipated by CMOS devices drives the I/O pads.

With many I/O pins on each device and anticipated frequencies in excess of 50 MHz, heat buildup was a concern. We reviewed the use of a BiCMOS process to obtain low-voltage I/O and 5-volt on-chip operation, but no BiCMOS vendor could support our requirements. Instead, we turned to a unique circuit design that allows us to use low-voltage swings (approximately 1V) for most high-frequency I/O lines. A low-voltage signal supplied to the chip serves as both a reference voltage for input signals and as a voltage source for outputs. This one-volt signaling method reduces the power used to drive the I/O lines by a factor of 25, and keeps total power requirements under 7 watts.

## Chip sets versus megachips

The decision to implement the C400 as a chip set, ultimately integrated on a multichip module, runs counter to the prevailing trend to integrate more and more processor functions on a one multimillion-transistor "megachip" (such as the 860, 960, 486, 68040, and so on). We believe the multichip approach has several important advantages that more than outweigh its one notable drawback. On the plus side, a chip set provides ample room for whatever silicon may be needed for the sake of performance or functionality. Thus, the Intel 860, with its 1.2 million transistors squeezed onto one chip, compares unfavorably in both performance and functionality

with IBM's RS/6000, which spreads out nearly 7 million transistors over nine chips.

Semiconductor economics also favor the use of several moderately sized chips over one very large device. Semiconductor production yield falls exponentially with increasing die area. One large die costs significantly more than two smaller die with the same aggregate area. At a ratio of 1.5 defects per square centimeter, a six-inch wafer yields only about three good 2.0 sq cm die, versus about 33 good 1.0 sq cm dies. Obviously, the economics strongly favor the smaller size. Independent of the cost of a wafer, the bigger die will be 5.5 times more expensive than the smaller, per unit area of (working) silicon.[11]

Of course, as manufacturing moves down the learning curve, defect densities decrease, yields increase, and the difference in cost between the one larger and the two smaller dies will decrease. Nonetheless, it never dwindles to the point of irrelevance. Our silicon suppliers estimate that it takes four years to cut the defect ratio roughly in half, that is, to move from 1.5 to 0.8 defects per square centimeter. But even at this second figure—typical for a mature product on a stable process—a 6-inch wafer yields only about 14 good 2.0 sq cm dies, versus about 66 good 1.0 $cm^2$ dies. The cost difference per unit area of (working) silicon still favors the two smaller dies by considerably more than two to one.

It is tempting to argue that these cost differences sound more important than they really are, since the CPU comprises an increasingly small fraction of the cost of building a system. One might argue that in a $10,000 system, it hardly matters whether the CPU costs $100 or $250. This argument assumes the economics of low volumes. As unit volumes increase, small cost savings get magnified greatly. As the RISC market grows, product shipments for workstations will be measured in the hundreds of thousands of units. At only 100,000 units, a savings of $150 a unit amounts to $15 million—a sum that falls straight to the bottom line.

The major disadvantage of chip sets, compared to megachip processors, is that communication between functional units located on different chips involves greater delays than communicating between units located on the same chip. It takes time to build up the current required to drive from chip to chip, and still more time for current to travel the extra distances involved. The result is that chip sets usually run somewhat slower than single chips. But this disadvantage can be minimized. Off-chip accesses can be pipelined, effectively bringing them down to one clock cycle (assuming sufficient instruction-level parallelism exists to keep the pipeline full). Even without pipelining, interconnection delays, when packaged on a multichip module, amount to no more than 2 or 3 ns. At 100 MHz, this time represents no more than 20 to 30 percent in overhead penalty (2 or 3 ns added to a 10-ns clock).

A 20 or 30 percent penalty in CPU performance seems a modest price to pay for a savings of more than 60 percent in CPU cost. We are in an era when margins are under increasing pressure for all workstation manufacturers. One key to survival in the coming decade will be cost-effective manufacturing. Cost-effective manufacturing begins with attention to small details and a willingness to make sensible product design trade-offs based on economics.

THE C400 DEVELOPMENT PROGRAM accomplished a complete hardware redesign of the Clipper processor in slightly less than two years, approximately half of the elapsed time required to implement the original C100. The speed with which we performed this task demonstrates two positive aspects of our industry and Clipper architecture.

First, VLSI development tools have improved vastly over the last five years, especially in the areas of chip layout and simulation.

Second, the design team incorporated a variety of new ideas in the C400, including superscalar instruction dispatch, superpipelined execution units, and even new delay slot instructions, while maintaining software compatibility with existing applications. This flexibility and compatibility demonstrates the robustness of the architecture and validates many decisions made in the Clipper's original definition, such as hiding the precise operation of the execution pipeline from application-level code. The performance gains from our limited opportunistic approach to superscalar instruction dispatch were truly gratifying.

This new Clipper delivers a level of performance unachievable just a few years ago. Nonetheless, the C400 does not represent a limit for future microprocessor development, any more than the original C100 did in 1985. We are enthusiastic about our accomplishments with the C400, based on 1990 architectural concepts and implementation capabilities. However, partly as a consequence of our experiences in designing and implementing this newest addition to the oldest line of commercial RISC microprocessors, we are also inspired with even newer architectural ideas and implementation possibilities. Today we are hard at work on our fourth generation Clipper—a machine targeted to provide three to four times the performance of the C400.

## References


1. W.M. Johnson, "Super-Scalar Processor Design," Stanford University Tech. Report CSL-TR-89-383, Stanford, Calif., June 1989, p. 52.
2. N. Jouppi, "The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance," *IEEE Trans. Computers*, Vol. 38, No. 12, Dec. 1989, p. 1,648ff.
3. N. Jouppi and D. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," *Proc. Third Int'l Conf.*

*Architectural Support for Programming Languages and Operating Sys.*, IEEE CS Press, Los Alamitos, Calif., Apr. 1989, p. 275ff.
4. *Introduction to the Clipper Architecture*, Intergraph Advanced Processor Division, Palo Alto, Calif., Jan. 1989.
5. "Clipper C300 Data Sheet," Intergraph Advanced Processor Division, Sept. 1988.
6. W. Hollingsworth, H. Sachs, and A.J. Smith, "The Clipper Processor: Instruction Set Architecture and Implementation," *Comm. ACM*, Vol. 32, No. 2, Feb. 1989, pp. 200--219.
7. W. Baxter and R. Arnold, "Code Restructuring for Enhanced Performance on a Pipelined Processor," *Proc. Compcon*, IEEE CS Press, 1991, pp. 252-260.
8. H. Sachs and H. McGhan, "The C400 Chipset Architecture," *Intergraph Advanced Processor Division White Paper*, Sept., 1990.
9. L. Hanson and N. Brookwood, "The C400 Superscalar/ Superpipelined RISC Design," *Proc. Compcon*, IEEE CS Press, 1991, pp. 247-251.
10. J. Wilson, "A Review of Superscalar Literature," Univ. of California at Berkeley Computer Science Division Report, Berkeley, Calif., Apr. 23, 1990, p. 10.
11. H. Sachs and H. McGhan, "Future Directions in Clipper Processors," *Proc. Compcon*, IEEE CS Press, 1991, pp. 240-246.

**Howard G. Sachs** is the executive vice president and general manager for Intergraph's Advanced Processor Division in Palo Alto, Calif. He was the chief architect of the C100, the original Clipper designed at Fairchild Semiconductor. Before joining Fairchild, Sachs was a vice president at the Boulder Division of Cray Research and directed engineering activities at National Advanced Systems, TRW, Xerox, and the California Institute of Technology.

Sachs has a BS in engineering from California State University at Los Angeles and an MSEE from the University of Southern California. He is a member of the IEEE.

**Harlan McGhan** is the manager of software engineering applications and analysis at the same division. Before joining Intergraph, he served as the software engineering manager and product line manager for National Semiconductor's 32000 series of microprocessors. He has also worked in engineering and engineering management, technical marketing, and technical writing.

McGhan received a BA in philosophy from Michigan State University and an MA and ABD in logic and history of science from Princeton University. He is a member of the American Philosophical Association.

Address questions concerning this article to Nathan A. Brookwood, Intergraph Corp., 2400 Geng Road, Palo Alto, CA 94303.

**Lee F. Hanson** is the director of chip development for the division, where he has managed the design effort for the C4 microprocessor for most of the project's lifetime. Before joining Intergraph, he managed the design of super-minicomputers at Gould's Computer Systems Division and held a variety of engineering and engineering management positions at Amdahl, STC Computer Research, National Advanced Systems, NCR, and Datum Corp.

Hanson earned a BS in electrical engineering from North Dakota State University and an MSEE from San Diego State University. He is a member of the IEEE and the Computer Society.

**Nathan A. Brookwood** is the director of marketing for the division. His responsibilities include planning and marketing for a range of Clipper-based chips, boards, and system-level products. Before joining Intergraph, Brookwood researched high-performance systems architecture for D.H. Brown Associates, a market research firm. He has also worked at DEC, Prime Computer, and Convergent Technologies.

Brookwood is a graduate of the Massachusetts Institute of Technology and the Harvard Business School.