High Performance Microprocessor Architectures

Randy H. Katz
Computer Science Division
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, CA 94720

John L. Hennessy
Computer Systems Laboratory
Electrical Engineering and Computer Science Department
Stanford University
Stanford, CA 95403

Abstract: Single chip processor performance has improved dramatically since the inception of the four-bit microprocessor in 1971. This is due in part to technological advances, (i.e., faster devices and greater device density), but also because of the adoption of architectural approaches well suited to the opportunities and limitations of VLSI. The most appropriate are those that effectively reduce off-chip memory accesses and admit of a regular pipelined implementation. The overriding goal of pipelining is to achieve "single cycle execution," i.e., instructions appear to execute in a single processor cycle. Today's RISC processors are close to realizing this goal, and the next generation will reduce the cycles per instruction even further. In this paper, we will review the design issues and the proposed architectures for high performance VLSI processors.

1. Introduction

Microprocessors have advanced rapidly from their early days as low-cost calculator building blocks. Today's 32-bit single-chip processors provide performance comparable to super-minicomputers, at an historically low price per MIPS (million instructions executed per second, an admittedly imprecise metric of processor performance). For example, the LSI Logic Sparc processor chip, a 20 Mhz part that is rated by its designers as achieving 12 MIPs, sells for $179 in 100 unit quantities. That is an astonishing $15/MIP! Much of this dramatic price/performance improvement has come about through the advances of VLSI technology. Myers [1] reports that in the fifteen years between the introduction of the Intel 4004 and the 80386, the circuit technology delivered a 100X increase in transistor density, an 8X improvement in speed-power product, and a 100X reduction in gate delays (see Table 1).

Advances in circuit technology, however, do not explain the whole story. A complex interplay between architectural invention and implementation constraints is also at work. For example, architectural features that reduce off-chip memory references or that lead to a regular pipelined implementation are well matched to the constraints of VLSI technology. Architectures chosen to reflect the underlying technological constraints will more quickly reap performance benefits as the circuit technology inevitably improves

Processor	Year	Devices	Technology	Physical Addr Range	Add Time (us)
4004	1971	2300	PMOS	4K	10.8 (4 bits)
8008	1972	3500	PMOS	16K	20.0 (8 bits)
8080	1974	5000	NMOS	64K	2.0 (8 bits)
8085	1976	6000	NMOS	64K	1.28 (8 bits)
8086	1978	29000	NMOS	1M	0.375 (16 bits)
80286	1982	130000	NMOS	16M	0.25 (16 bits)
80386	1985	275000	CMOS	32M	0.125 (32 bits)

Table 1: Trends in LSI/VLSI Processor Technology
The table is reproduced from [1], and shows the evolution of the Intel processor family over its first fourteen years. Note the 100x increase in transistor count (accompanied by an 8x increase in chip size). To match this fourteen year improvement in technology, mainframes required approximately thirty years.

over the next decade. Over the last five years, one such architecture has been the reduced instruction set computers, or RISCs. While [1] observed that "conventional" microprocessor architectures had been improving at the rate of 40% per year (a factor of two performance improvement every 2.25 years; a factor of ten every seven years), William Joy of Sun Microsystems predicated that RISC processor performance would double every year. Popularly known as Joy's Law, it can be expressed as

$$MIPS = 2^{Year - 1984}$$

Astonishingly, the designers of RISC microprocessors have kept pace with this audacious prediction. The remarkable implications of Joy's Law are represented in Figure 1.

This paper is an extension of [2], and is organized as follows. In the next section we present the key issues for computer architectures destined for single chip implementation. This provides a framework within which to describe the advantages that accrue to RISC architectures in Section 3. Section 4 discusses new developments in the architecture arena, in particular, superpipelined and superscalar architectures, that will play an increasingly important role in the next generation of high performance single chip processors. Section 5 is our summary and conclusions.

2. Architecture and Implementation Issues

The IBM System/360 family was the first computer to successfully decouple architecture from implementation. It was possible for the same machine language program to run on a continuum of implementations of the same instruction set architecture, yet representing different points in the cost-performance space. Because of the tight interplay between technology and limits of what can be implemented in a single chip, VLSI processor architectures have been more severely constrained, and it has not been possible to decouple architecture from implementation to the same extent. To a large extent, design decisions that influenced early microprocessor instruction sets live on in newer generations because of software compatibility considerations. As the technology delivered ever more transistors, the instruction set extensions became more complex.

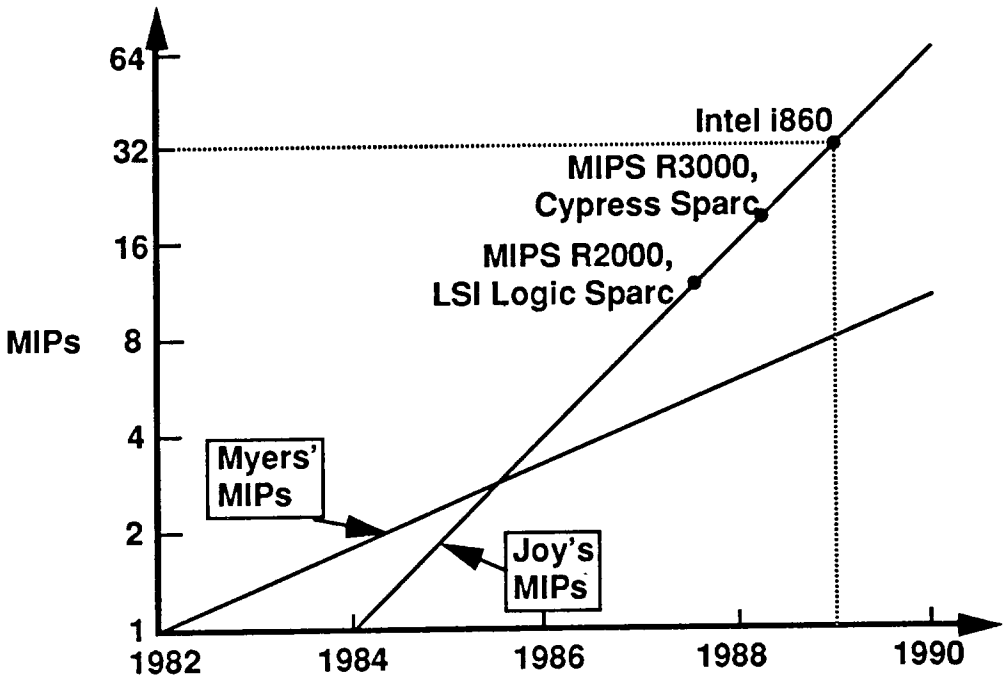

Figure 1: Myers' and Joy's MIPs
The Y-axis, Processor Mips, is logarithmic. The 40% per year extrapolation for conventional microprocessor architectures rapidly falls behind the predictions for RISC microprocessors. Despite being made in 1984, Joy's Law has so far been proven true by a number of available products, including Intel's recently announced i860.

By the early 1980's the time was ripe to reexamine the interplay between architecture and implementation for single chip processors.

VLSI technology imposes some important design considerations that cannot be ignored during the design of a new processor architecture. The first is *utilization of real estate.* Even with today's one million transistor devices, the allocation of chip area to function must be done carefully. Memory bits are cheap, regular, and transistor and design efficient; random logic and interconnect are not.

The second consideration is *time to market.* Because of intense competition coupled to a very rapid rate of innovation, a delayed product introduction will have serious problems in capturing market share: a late product is not likely to be performance competitive. Design correctness to achieve working silicon in a first pass becomes an overriding consideration. This has led several computer architects to pursue "simplicity" rather than ever increasing complexity in microprocessor architectures. An important secondary effect is that simpler architectures normally require less resources to implement them, and are thus the best candidates for implementation in the most aggressive new technologies.

The third is the *magnification of performance* when critical paths can be subsumed within a single chip, because of the considerable expense associated with crossing chip boundaries. Thus, a single chip processor can outperform a multichip ECL implementa-

tion, even though the latter technology is much higher performance at the gate level.

Finally, the *limits of VLSI technology* also impact design decisions. For example, local communication is faster than global communication, and the technology favors architectures that can exploit this. The allocation of the scarce transistor resources to processor functions, such as control, datapath, and on-chip memory, also plays a critical role.

These considerations led, in part, to the original proposals for single chip reduced instruction set computers (RISCs). Prior to RISCs, processor instruction sets had begun to contain an overwhelming collection of operations, operand accessing modes, and instruction formats. A number of studies indicated that most instructions in existing "complex" architectures were rarely executed. Thus, one way to achieve faster time to market was to simplify the processor's instruction set. It was proposed that the simplicity of RISC processors should yield shorter design cycles, while simultaneously yielding a faster machine with a better utilization of silicon resources. While it makes sense that a simpler processor should take less time to design, the latter two points are not nearly as self-evident. We shall examine these points in the next section.

3. Why RISC is a Good Idea

3.1. Pin Bandwidth and Instruction Set Design

The critical performance bottleneck for VLSI processors is represented by the limited bandwidth of the single chip processor's pins. The pin bandwidth limitation manifests itself in several ways. From a technological perspective, it is expensive in power and delay to cross the chip boundary. While we may expect that external busses will increase in size and speed in the near future, packaging and chip periphery restrictions will continue to severely limit the number of connections relative to the amount of logic on chip. Architecturally, the bandwidth limitation implies that main memory access will continue to be slow compared with the processor's cycle time, and that the speed gap will widen.

One possible solution is to design a highly encoded instruction set, with high "function" per instruction. A typical program could be represented by a shorter sequence of instructions, thus decreasing the number of instructions that need to cross the processor's pins to accomplish a given task. This approach has motivated the design of most of the conventional instruction sets.

Nevertheless, a careful analysis must be made of how the usage of chip area affects performance. A highly encoded instruction set requires a more complex instruction decoder, a step that often determines the overall cycle time of the machine. Microprocessors that have taken this approach typically dedicate more than 50% of their chip real estate to on-chip microcode store. Rather than being executed directly, a given complex instruction is actually interpreted by some sequence of microcode fetches.

The alternative is to employ streamlined or "reduced" instruction sets, which allow instructions to be executed by the hardware directly. Complex tasks are written as a sequence of the more primitive "reduced" instructions. This allows the chip real estate normally dedicated for microstore to be more effectively used for other forms of on-chip memory that can more directly improve program performance [3]. These include large register sets, instruction caches, and data caches. Registers allow operands to be accessed at processor rather than memory system speeds, and compilers have been developed that can be effective in allocating frequently accessed variables to registers. A register-oriented machine has the additional advantage that operands can be encoded in fewer instruction bits than if full addresses are needed. Instruction caches are efficient in capturing "loops" within programs: once an instruction is fetched from main memory, it can be "cached" in a fast memory within the processor. If it is referenced in the near future, as one would expect within a loop, the next time the instruction is accessed, it can be obtained much more rapidly if it is "hit" in the cache memory. Because instruction access exhibits a high degree of temporal reference locality, relatively small caches can obtain high hit ratios. Data caches also reduce off-chip references by catching accesses to recently referenced data, although they are not as effective as instruction caches, because data does not have as high temporal locality.

Instruction path length is defined as the number of instructions that must be executed to complete a task. The difference in instruction path length between RISC and more conventional machines is actually not as great as one might think. Recent studies indicate that compilers generate sequences with comparable instruction counts for the Motorola MC68020 as for the commercial RISC architectures, while RISC instruction counts are approximately 1.8 times as much for programs compiled for the Digital Equipment Corporation's VAX architecture [4].

In the (still) transistor constrained world of 1989, the technology tradeoffs favor instruction sets that require simple decode over more complex encoded instructions. This does not imply, however, that the same choices would be made in a less constrained world. Nevertheless, simple to decode instruction sets have additional advantages, which we discuss next.

3.2. Pipelining and Instruction Set Design

Perhaps the most significant reason why RISC architectures lead to faster machines is because they admit of a more efficient pipelined implementation. CPU performance can be expressed in terms of the clock rate divided by the product of the dynamic instruction count and the average number of processor cycles per instruction (CPI) [5]:

$$\text{CPU Performance} = \frac{\text{clock rate}}{\text{dynamic instruction count} \times \text{cycles per instruction}}$$

Performance can be improved by increasing the clock rate, or by reducing the instruction count or the cycles per instruction. Technology factors make possible a higher clock rate, while RISC concepts dramatically decrease the number of cycles to execute the average instruction at the expense of modestly increasing the number of instructions needed to complete a task.

Pipelining is the best known technique for reducing the effective clocks per instruction. In just about any processor implementation, every instruction must pass through the stages of instruction fetch, instruction decode, operand fetch, instruction execute, and result write. If these stages can be overlapped, then a separate instruction can be in each stage at the same time. An instruction's execution time, or *latency*, is the sum of its (five) stage times. Since an instruction exits (and under ideal conditions a new instruction enters) the pipeline every stage time, the effective time per instruction is that of a single stage. Hence the term "single cycle execution" often associated with pipelined instruction sets.

Consider Table 2, which shows the clock rate, cycles per instruction (CPI), and native MIPS for several popular machines. The CPI was determined by averaging the results for a standard benchmark set run for each of the machines reported in the figure. It clearly shows the importance of CPI in determining overall performance. The VAX 11/780 and the IBM RT-PC have comparable clock rates, 5 and 6 Mhz respectively, yet the advantageous CPI of the RISC machine allows it to execute its instructions at a four times faster rate. Since the VAX has a highly encoded instruction set relative to the RT-PC, it takes approximately twice as many of the latter's instructions as the former's to perform the same task. Thus, the RT-PC is still twice as fast at a comparable clock rate. The MC 68020 and the MIPS R2000 have the same clock rate, yet once again, the advantageous CPI for the RISC machines yields a four times improvement in performance. As we have noted before, dynamic instruction counts for the two architectures are very comparable.

Machine	Clock Rate	CPI	Native MIPs
VAX 11/780	5 Mhz	10	0.5
VAX 8550	22.2	9	2.5
MC 68020	16	7	2.3
Clipper	33	6	5.5
RT-PC	6	3	2.0
MIPS R2000	16	1.6	10.0
Fujitsu SPARC	16	2.1	7.6
88000	20	1.6	12.5
MIPS R3000	25	1.3	20.0

Table 2: Native MIPS
The CPI is based on averages observed for the execution of a standard benchmark set run on the indicated machine. Note that the clock rate alone does not predict true machine performance.

Machine	Pipeline Stages	CPI	Instructions in Execution
VAX 8700	5	6.8	0.7
MC 68020	2	6.3	0.3
MIPS R3000	5	1.2	4.2

Table 3: Pipeline Efficiency and CPI
Instructions in execution is derived by dividing the number of pipeline stages by the average CPI. A low results indicates that there is little instruction level parallelism being exploited. The optimal result would be a perfectly utilized pipeline, achieved by reaching a CPI of 1.

Cycles-per-instruction offers an important measure of the efficiency of a machine's pipelined implementation. The smaller the CPI, the better the processor is at exploiting *instruction level parallelism*, i.e., the overlapped execution of instructions in the sequential instruction stream. Despite a much faster clock rate, the VAX 8550 only barely improves on the clocks per instruction of its ancestor the VAX 11/780. This is in large part due to the difficulty in overlapping the interpretation and execution of the complex multi-format variable-operand instruction format of the VAX. Pipeline efficiency is illustrated in Table 3. The figure shows for several machines the number of pipeline stages, the CPI, and the derived instructions in execution. The RISC machine, with its very low CPI, yields the highest number of instructions in simultaneous execution, and hence a more highly utilized and thus efficient pipeline. This can be accomplished because of the streamlining of the instruction set: memory delays are more easily decoupled from the pipeline by separating load/store instructions from register-register instructions, delayed branches facilitate the smooth change of control flow, instruction decode and execution take a predictable number of processor cycles, etc. The degree of pipeline efficiency due to streamlined instructions is why a high end implementation of the IBM System 370 architecture requires 200,000 gates to achieve a CPI close to 3, while a single chip RISC processor, which can be implemented in 40,000 to 50,000 gates, achieves a CPI of approximately 1.

Thus, the advantages due to RISC architectures primarily come from the instruction set streamlining and its resulting pipeline efficiencies, leading to very low cycles per instruction. After normalizing for instruction set differences, the advantage is approximately a factor of five in terms of instruction execution rate.

3.3. RISC Performance

One might be left with the impression that all RISC architectures are the same and that there is little to differentiate the vendors. This is not the case. The RISC approach depends intimately on moving some of the pipeline scheduling complexity to compile time. For example, a compiler must be effective in finding useful instructions to insert into the delay slots following branch instructions. This requires a careful analysis of looping and IF-THEN-ELSE constructs in high level language programs. It is often helpful to obtain a profile of the program's execution patterns before compiler optimization. This can help identify the more common directions for conditional branches, i.e., are

they more frequently taken or not taken (see Figure 2 and the explanation below). Even though the architectures are similar at a high level, the quality of compiler technology varies from one manufacturer to another.

In addition, there are some architectural differences among the various RISC architectures that do have an effect on performance. These include (1) branch architectures, (2) the organization of on chip registers, and (3) the choice of what primitives to support in hardware. We will examine each of these in turn.

Every RISC machine exposes some aspects of its pipeline to the compiler. The most notable example is the delayed branch: to avoid pipeline hazards associated with the update of the program counter, the instruction following the branch is always executed. This guarantees that once an instruction enters the pipeline, even in the presence of conditional branches, it will complete its execution (assuming no traps or interrupts). This helps achieve a CPI of 1 by avoiding "holes" or "bubbles" in the pipeline due to switching the instruction stream fetch to a different memory location. However, as mentioned above, it is something of a challenge for compilers to find useful instructions to place in the delay slot. The compiler's job can be simplified by having *canceling branches* in the instruction set. Figure 2 shows two alternative code sequences for a repeat-until loop, one of which uses a canceling branch with the semantics that the instruction in the delay slot is canceled when the branch is *not* taken. Note how the canceling branch allows the delay slot to hold a useful instruction rather than an NOP. It should be noted that very good compilers can often rearrange the instructions within the loop to achieve the same effect with a conventional delayed branch instruction.

On chip register organizations also affect performance. Some RISC processors contain register windows (overlapped and non-overlapped), basically a sliding porthole to a large stack of registers, while others contain more conventional register files. Register windows accelerate subroutine call and return, by obviating the need for saving and restoring processor registers to and from memory. However, the decode time for the large register stack, in the range of 138 registers for most of today's machines that incorporate this feature, could be the critical path that lengthens the machines basic

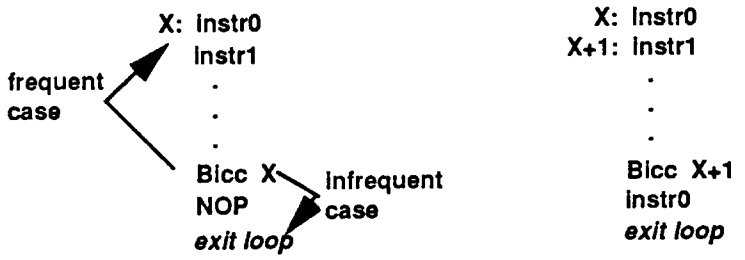

Figure 2: Canceling Branches
The code sequence on the left is for a straightforward implementation of an indexed do loop or repeat-until loop. **Bicc X** represents a conditional branch to address X. The right sequence shows the alternative assuming that the branch at the bottom cancels the following instruction when the branch back is *not* taken. With canceling branches, instr0 can be placed in the delay slot and only canceled for the final loop iteration in which the loop is exited.

cycle time. Also, more advanced compiler techniques for register allocation can make more efficient use of a smaller number of registers, even across procedure calls.

Finally, there is the issue of what primitives to include in the instruction set. While the goal is single cycle execution, some operation are likely to require multiple instructions, such as multiply and divide. Some processors provide a primitive multiply/divide step that must be iterated to perform full 32-bit operations, while others implement the full operations in hardware.

Yet these effects are actually rather small compared with variations in pipeline strategies, especially in terms of memory interfaces and coprocessors. For example, some RISC machines assume a split instruction and data cache off chip (a so-called "Harvard architecture"), and have organized their pipelines and memory interfaces to fetch an instruction and data word within the same cycle. This can be accomplished either by time multiplexing the address and data pins, or by providing a separate set of pins for instruction and data memory. Other architectures assume a unified cache, and can not perform both accesses at the same time. A similar problem exists for loads and stores: some machines can execute these operations in a single cycle if coupled to a fast off-chip memory, while others will require more than one cycle. Another place where architectures differ is in the area of floating point pipelines. If these are fully integrated into the integer pipeline, rather than decoupled through a coprocessor or multiple function unit interface, the pipelines tend towards a large number of stages. Deep pipelines can result in higher instruction latency for all instructions. If the pipeline depth exceeds the amount of available instruction level parallelism, i.e., it is not possible to find enough instructions before a branch to keep the pipeline flowing smoothly, this can lead to higher CPI and thus degraded performance.

Figure 3 shows a performance comparison between a MIPS R2000 and a Cray XMP uniprocessor for five different workloads: integer, LISP symbolic processing, non-vectorizable floating point, low-to-medium vectorizable floating point (the livermore loop kernels), and highly vectorizable floating point (linpack). The figure clearly shows that the current generation of RISC processors are already comparable or faster than the Cray for some tasks, although slower in raw floating point and vector processing performance. The gap between the RISC machines and the supercomputers are narrowing rapidly, especially given the observation that RISC machines are doubling in performance every 12 to 18 months. Justin Rattner of Intel has observed that single chip RISC processors surpassed conventional minicomputer performance in the late 1980's and predicted that high end mainframe performance could be eclipsed in the mid 1990's [6].

4. What Comes Next

4.1. Architectural Futures

A number of factors are driving towards faster single chip processors, irrespective of the architectural approach. Innovations in process technology will continue to produce

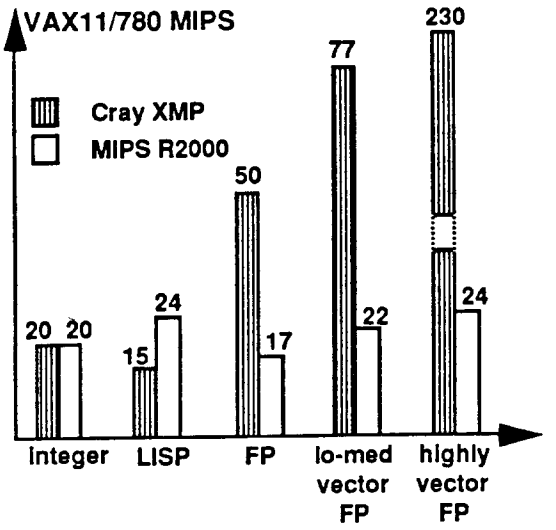

Figure 3: RISC vs. CRAY XMP Performance
The figure shows the relative performance, in terms of VAX 11/780 MIPs, for five benchmark workloads. The RISC machine is better at integer and symbolic processing than the Cray, although the Cray is superior in raw floating point and especially vector processing floating point. Note however that RISC machines have been doubling in performance every 12 to 18 months, while we have not seen a similar improvement in supercomputer performance. The gap between RISC processors and supercomputers is narrowing rapidly, and some experts expect the crossover to take place within the next five years, at least for programs with low to moderate levels of vectorization.

new integrated circuit technologies with ever higher clock rates. Better compilers and small improvements in instruction sets will obtain more efficient utilization of machine pipelines.

However, the major impetus to higher levels of performance will be new computer organizations that will drive CPI below one clock per instruction. To a large extent this is the secret of vector processing machines: a single instruction is issued yet the operation is applied to a large number of operands, in effect achieving more than one operation per instruction.

However a new collection of ideas have begun to emerge, based on incorporating multiple functional units within the processor to permit more than one instruction to be issued per clock. These machines are called *superscalars* [7]. Figure 4 compares pipeline diagrams for a generic RISC machine and a superscalar implementation of the same architecture. Each implements the same basic four stage pipeline: instruction fetch, decode, execute, and write results. However, this particular superscalar machine has the ability to issue two instructions at the same time, assuming that each does not require the results of the other (i.e., they can be executed in parallel without changing the results of the final computation). Performance is doubled, at an increased cost in memory and register files ports, as well as an additional ALU.

An early example superscalar was the original MIPS processor developed at Stanford University, which could combine two ALU-register operations or one LD/ST and one ALU-register operation within a single instruction [8]. However, they initiated a new instruction every other cycle, and so the net effect was a single cycle per instruction. The Intel i860 has a limited capability of issuing two 32-bit instructions simultaneously, as long as one is a floating point operation and the other is an integer operation.

An alternative to superscalars are the *very long instruction word* (VLIW) machines.

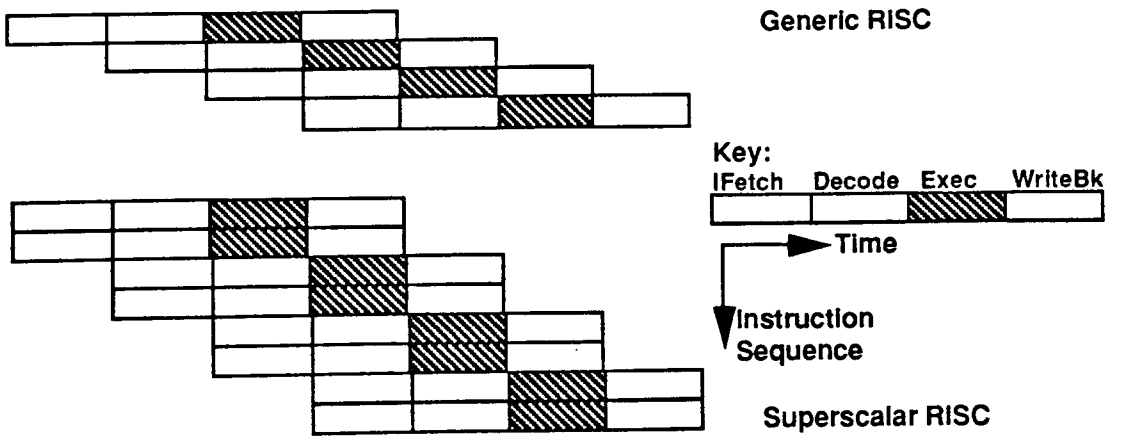

Figure 4: Pipeline Schemes for RISC and Superscalar Machines
The figure compares a generic 4-stage pipeline for a RISC machine (instruction fetch, instruction decode and read register operands, operation execution, and operation result register writeback) with the pipeline of a superscalar machine that can issue two instructions per clock cycle. The superscalar requires twice as many ports to the instruction memory and register file, as well as two independent functional units (i.e. ALUs). The figure is from [7].

VLIW machines [9] employ a large number of functional units within the CPU and use compiler techniques to rearrange the instruction stream to find operations that can be simultaneously executed. Figure 5 shows a pipeline diagram for this class of architectures. The approach depends on finding many operations that can be packed into a single instruction for simultaneous execution. There is some controversy about the amount of instruction level parallelism that can be extracted from real programs. A variety of studies place the limits from less than two all the way to many tens of instructions. The prevailing wisdom is that VLIW machines do well on the same kinds of programs as vector processors: those with simple control flow, such as applying the same operation to all elements of a matrix.

However, for non-vectorizable codes, such as most non-numeric programs, the available instruction level parallelism is actually relatively low, in the range of 2 to 4 instructions [10]. Thus superscalar machines appear to be a more attractive architecture, especially since they have a number of implementation advantages: the choice of which instruction(s) to issue can ultimately be made at run-time rather than determined

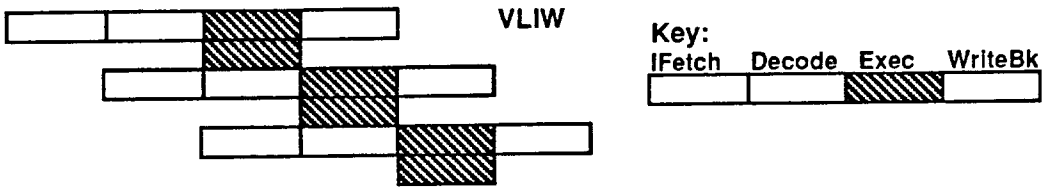

Figure 5: VLIW Pipeline Structure
A VLIW pipeline can be represented by multiple parallel tasks during the execution stage. The figure shows two simultaneous execute operations, but in a general VLIW, significantly more parallel operations are usually supported, limited only by the number of functional units within the processor. VLIW instructions are typically hundreds of bits long. The figure is adapted from [7].

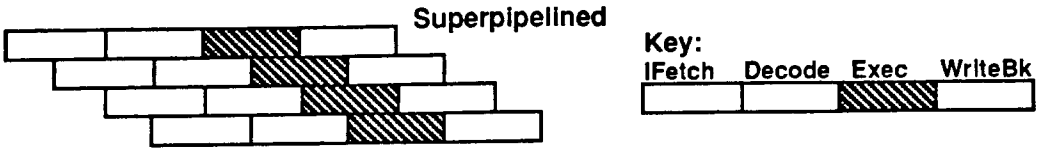

Figure 6: Superpipelined Pipeline Structure
The figure shows a pipeline structure in which the cycle time is one half that of the generic RISC machine of Figure 7. The functional unit is pipelined, so a new operation can start every cycle. The figure is adapted from [7].

statically at compile time (although the existing superscalars are static rather than dynamic), better code density will be possible especially if actual instruction level parallelism is low, and finally, object code compatibility with existing RISC instruction sets can be maintained, an important practical consideration.

Another alternative, which seeks to reduce the machine's cycle time rather than the number of instructions issued per clock, is called *superpipelining* [7]. In many current RISC machines, the pipeline is arranged so that the ALU operation time determines the machine's cycle time, even if instruction fetch takes less time. This assumption results in a simplified pipeline structure, requiring reduced control logic at the cost of potentially increasing the cycle time. In superpipelined machines, the functional units like the ALU are pipelined, requiring multiple pipeline stages to complete their operations. Such an organization allows the shorter instruction fetch time to determine the machine's cycle time. The idea is not a new one. The computers designed by Seymour Cray exhibit this kind of pipeline structure. The pipeline structure is shown in Figure 6.

The relative merits of RISC, superscalar, superpipelined, vector, and VLIW machines are summarized in Figure 7. Superscalar, VLIW, and vector architectures all seek to increase the number of instructions issued per clock cycle, the latter achieving this by increasing the number of operations per instruction for a special class of instructions. Superpipelining increases the achievable clock rate by dividing the functional units into finer pipeline stages, which is also used to advantage by vector processing. The most controversial point in the figure is the location of the VLIW architecture. Despite having the potential to issue a large number of instructions per cycle, the control of such a large number of functional units is likely to sacrifice clock rate, especially if the

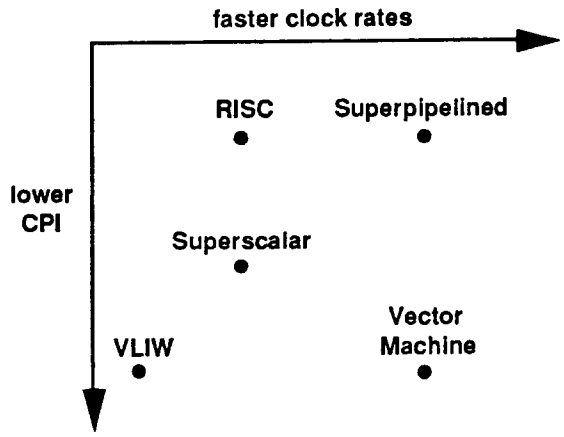

Figure 7: Architectural Futures
This figure shows the differences among RISC, Superpipelined, Superscalar, Vector, and VLIW architectures with respect to clock rate and CPI. Superpipelining allows RISCs to run at faster clock rates, with no effect on CPI. Superscalars improve CPI with no effect on clock rate. Vector machines are pipelined and support many operations per cycle, and are well matched to numeric codes. VLIW achieve very low clocks per instruction through large number of functional units, but the control circuitry appears to be very complex, and can actually reduce the achievable clock rate.

achievable instruction level parallelism is only very modest. A much more attractive alternative is to combine superscalar, superpipelining, and vector processing techniques, as has been described in [11].

The experience with RISC designs supports the adage that machines must be balanced. This is essentially Amdahl's famous law that states that the slowest part of computation will limit the available speedup. No design can afford to have one component be extremely high performance at the expense of the others. Good designs will have fast branch, memory, and arithmetic support. The combination of deeper pipelines and multiple instruction issue imply longer effective latencies for instructions. This could cause problems unless instructions can be found to keep the pipelines filled around branches. Compiler techniques will be critical in finding such instruction level parallelism. Eventually, there are limits to the amount of instruction level parallelism that can be found in a single instruction stream. Once this limit is reached, the next step to higher performance will require parallel processing, distributed memory architectures, and hardware support for switching among multiple threads of execution within a single processor ("multi-threaded" machines). Will RISC ideas run out of steam? We have not yet reached the limit in extracting the parallelism in the instruction stream, nor have we reached the end of the range of increased clock rates due to architectural innovations. At worst, performance improvements will return to technology driven growth in the range of 20-25% per year.

4.2. Technology Implications

Extrapolating from the raw technology trends, Myers [1] predicted that the processor of 1995 will be constructed from 0.25 micron CMOS technology, containing 5-10 million transistors on a 500 square mil die, with gate delays below 100 ps. More than one half of these transistors will be dedicated to cache functions. In fact, designing memory systems that can sustain the high memory access rates will be one of the most important challenges for the next generation of single chip processors. As the CPI approaches (and goes below) 1, a cache miss penalty becomes very severe. To avoid the pipeline stalls associated with memory access, which drive up the CPI, cache memory design becomes critical. Caches will go to two level, one level on chip (split for instructions and data) and one unified level off-chip, and they will be implemented in fast static RAM rather than dynamic RAM. By the mid-90's we can expect to see a 1Mbit SRAM cache within the CPU, and a 16 Mbit cache off-chip implemented with BiCMOS SRAMs. The processor will be heavily pipelined, incorporating many of the techniques traditionally used in mainframes and supercomputers.

Rattner [6] extends the predictions to the year 2000. He foresees 50-100 million transistors on a one inch square silicon die, incorporating four 250 Mhz processors, organized as a shared memory multiprocessor. The higher transistor count is expected because of the extensive use of memory cells. Each processor will be able to reach a peak integer processing rate of 750 MIPs and a 500 Mflop peak floating point processing rate. The multi-megabyte second level cache will be implemented on the die. Once again, the predicted technology will be CMOS or BiCMOS.

Design complexity, rather than transistor density, will continue to be the limiting factor in bringing these chips to market. We expect to see the incorporation of full floating point arithmetic and integer multiply/divide on chip, along with multiple scaler functional units. A worthwhile tradeoff exists between compiler technology and hardware complexity, and we expect to see a continued shift of decision making to compile-time to simplify the run-time hardware.

Binary compatibility with existing architectures will continue to be a key requirement, although RISC machines have demonstrated that it is possible to break with the past if there is sufficient performance gain by doing so. There will be less controversy in instruction sets with respect to RISC vs. CISC, although 64-bit instruction sets could lead to some some innovative ideas. These 64-bit instruction sets will need to have a compatibility mode with existing instruction sets. There will be a return to somewhat more complex instruction set formats, and while single cycle execution will be the goal, some instructions will still require multiple cycles. There will be a shift in emphasis from instruction sets to system support features, such as memory management, cache control, and multiprocessor support. Special purpose coprocessors for I/O processing, graphics, message passing, will be fabricated on the same die. [6] postulates two different futures for microprocessors: *high performance* and *high integration*. The former would be a four processor multiprocessor with on-chip support for graphics and interprocessor communications. The latter would be a "workstation on a chip", with integrated coprocessor support for human interfaces, such as speech recognition/synthesis and digital television.

The highest performance single-chip architectures will be 64-bit machines by 1995. There already exists a demand for address spaces that are larger than 32 bits. External busses significantly larger than 32-bit would ameliorate some of pin/bandwidth issues we have already raised. 64-128 bit wide processor buses running at 16-32 Mhz have already been predicted [1]. Further, a 64-bit internal architecture would result in a primitive superscalar architecture, yielding an ability to pack multiple independent simple instructions within the same instruction and also to use the datapath for at least two independent 32 bit operations at the same time.

We expect to see more extensive use of macrocell rather than full custom technology to implement these future processors. The SUN Microsystems/Fujitsu SPARC chip illustrated that macrocell and high-performance are not incompatible, and that reduced instruction set machines are a good match to this implementation technology. Although the densest and most aggressive chips will continue to be full custom, a class of high performance machines will get to market more rapidly by using this approach.

While we have described a positive future for high performance single-chip processors, there remain a number of technical issues that may yet limit their success. The most pressing problem will be how to construct very high clock rate CMOS and BiCMOS systems. The problems of dynamic power consumption will require packaging and cooling technologies adapted from those used in today's highest performance ma-

chines. It is virtually certain that a reduced power supply voltage will be used. The problems of clock distribution at these speeds across a 500 square mil die will be considerable, and besides the circuit issues, a non-synchronous architectural design style may be necessary. Off-chip delays will be considerable, and hybrid packages that allow the processor and its memory to be closely located and connected with low impedance paths will be required.

5. Summary and Conclusion

In this paper, we have briefly reviewed the developments in high performance VLSI processor architectures. The key attributes of an architecture that make it suitable for high performance VLSI implementation are its proficiency to minimize off processor references, through the incorporation of data and program memories on processor, and its ability to realize high performance through a pipelined implementation. The reason that these attributes are the most critical is the inherently limited bandwidth that is possible across the processor's pins and that performance depends critically on the average number of clock cycles necessary to execute an instruction. The interested reader is directed to [12] for a more thorough treatment of instruction set usage and pipelining issues than we can cover here.

While there has been a raging controversy over instruction set design, we believe that future processors will focus more on system features and advanced implementation techniques than instruction sets. Part of the motivation for the RISC/CISC debate has been the relatively limited number of transistors available on a processor chip. Even with the ability to place several million transistors on such a chip, some critical lessons of the last few years should not be forgotten. Namely, instruction sets that are streamlined lead to simpler, faster pipelined implementations than complex instructions with multiple formats and operand accessing modes; that software complexity can be traded-off to reduce hardware complexity, in particular, hardware design complexity; and that simpler pipelined implementations can take advantage of more rapid turnaround technology such as macrocell design to get to market faster.

RISC architectures are really about rational and well-designed pipelined implementations. It has been observed that successful pipelining requires that no single stage should dominate, which provides another rationale for reduced instruction sets with their simpler decoding formats. There are a whole host of aggressive mainframe and supercomputer pipelining techniques, such as branch prediction, multiple branch path following, register scoreboarding, etc. that are likely to be attempted to be incorporated into future single chip processors. In fact, the goal of "single cycle execution" can be achieved for just about any instruction set, given sufficient transistor resources, but at significant cost in design complexity with a potentially longer clock cycle than a more straightforward pipelined implementation. Also, the RISC concepts, perhaps not as critical in a less transistor constrained point in technology, still demonstrate the importance of compiler technology to assist in pipeline efficiency and to reduce hardware complexity. Simple techniques such as delayed branches and loads, easily supported by compiler technology, simplify the pipeline hardware and improve its efficiency by eliminating

potential pipeline stalls.

The ultimate limit of conventional pipelined machines, that of commencing the execution of a new instruction every clock cycle, will be expended by new techniques that seek to initiate *more than one* instruction per clock cycle. We have described superscalar methods oriented towards multiple instruction issue per clock. These techniques are likely to be incorporated into future single chip processors.

We expect to see high performance architectures based on a 64-bit internal organization, with multiple functional units and more aggressive pipelining than that found in today's machines. However, because of complexity considerations, we do not expect to see very complex pipelined implementations on a single chip. The challenge of high clock rate CMOS and BiCMOS systems will keep circuit designers and chip system architects busy for many years to come.

Acknowledgements: The authors wish to thank Tzi-cker Chuieh, Valerie King, and the editors of this journal for their helpful suggestions and comments.

6. Bibliography

[1] G. J. Myers, A. Y. C. Yu, D. L. House, "Microprocessor Technology Trends," *Proceedings of the I.E.E.E.*, Vol. 74, No. 12, (December 1986), pp. 1605-1622.

[2] R. Katz, "High Performance VLSI Architectures," Proc. Symp. on VLSI Circuits, Kyoto, Japan, (May 1989).

[3] D. A. Patterson, "Reduced Instruction Set Computers," *Communications of the ACM*, Vol. 28, No. 1, (January 1985), pp. 8-21.

[4] J. L. Hennessy, "RISC Architecture: A Perspective on the Past and the Future," Proceedings 10th Caltech Conf. on VLSI, Pasadena, CA, (March 1989).

[5] J. L. Hennessy, "VLSI Processor Architecture," *I.E.E.E. Transactions on Computers*, Vol. C-33, No. 12, (December 1984), pp. 1221-1246.

[6] J. Rattner, "MICRO 2000: Microprocessors in the Year 2000," Keynote Address, 2nd Annual VLSI Educators' Conference, Santa Clara, CA, (July 1989).

[7] N. Jouppi, D. W. Wall, "Available Instruction Level Parallelism for Superscalar and Superpipelined Machines," Proc. ACM/IEEE ASPLOS-III Conference, Boston, MA, (April 1989), pp. 272-282.

[8] J. L. Hennessy, et. al., "MIPS: A VLSI Processor Architecture," *Proc. CMU Conf. on VLSI Systems and Computations*, Computer Science Press, Rockville, MD, 1981, pp. 337-346.

[9] R.P. Colwell, et. al., "A VLIW Architecture for a Trace Scheduling Compiler," *IEEE Transactions on Computers*, Vol 37, No 8, (August 1988), pp. 967-979.

[10] M. D. Smith, M. Johnson, M. A. Horowitz, "Limits on Multiple Instruction Issue", Proc. ACM/IEEE ASPLOS-III Conference, Boston, MA, (April 1989), pp. 290-302.

[11] N. Jouppi, J. Bertoni, D. W. Wall, "A Unified Vector/Scalar Floating Point Architecture," Proc. ACM/IEEE ASPLOS-III Conf., Boston, MA, (April 1989), pp. 134-143.

[12] J. L. Hennessy, D. A. Patterson, *Quantitative Computer Architecture*, Morgan Kaufman Publishers,

Menlo Park, CA, 1990.