Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution

YALE PATT, FELLOW, IEEE

Invited Paper

The first microprocessor, the Intel 4004, showed up in 1971. It contained 2300 transistors and operated at a clock frequency of 108 kHz. Today, 30 years later, the microprocessor contains almost 200 million transistors, operating at a frequency of more than 1 GHz. In five years, those numbers are expected to grow to more than a billion transistors on a single chip, operating at a clock frequency of from 6 to 10 GHz.

The evolution of the microprocessor, from where it started in 1971 to where it is today and where it is likely to be in five years, has come about because of several contributing forces. Our position is that this evolution did not just happen, that each step forward came as a result of one of three things, and always within the context of a computer architect making tradeoffs. The three things are: 1) new requirements; 2) bottlenecks; and 3) good fortune. I call them collectively agents for evolution.

This article attempts to do three things: describe a basic framework for the field of microprocessors, show some of the important developments that have come along in the 30 years since the arrival of the first microprocessor, and finally, suggest some of the new things you can expect to see in a high-performance microprocessor in the next five years.

Keywords*—*Computer architecture, microarchitecture, microprocessor, microprocessor design, microprocessor evolution.

I. Basic Framework

A. Computer Architecture: A Science of Tradeoffs

Computer architecture is far more "art" than "science." Our capabilities and insights improve as we experience more cases. Computer architects draw on their experience with previous designs in making decisions on current projects. If computer architecture is a science at all, it is a science of tradeoffs. Computer architects over the past half century have continued to develop a foundation of knowledge to help them practice their craft. Almost always the job of the computer architect requires using that fundamental knowledge to make tradeoffs. This has been especially true throughout the evolution of the microprocessor.

Manuscript received May 29, 2001; revised August 6, 2001.

The author is with the University of Texas at Austin, Austin, TX 78712-1084 USA (e-mail: patt@ece.utexas.edu).

Publisher Item Identifier S 0018-9219(01)09681-5.

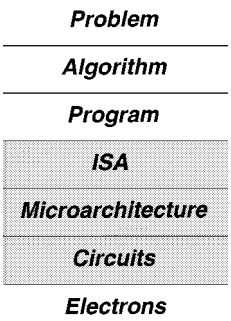

Fig. 1. The microprocessor today.

B. Levels of Transformation

Numbers of transistors and their switching times are resources provided by process technology. What we use those resources for depends on the demands of the marketplace. How we use those resources is what the microprocessor is all about. Fig. 1 shows the levels of transformation that a problem, stated in some natural language like English, must go through to be solved. In a real sense, it is the electrons that actually do the work and solve the problem. However, since we do not speak "electron" and electrons do not speak any natural language, the best we can do is systematically transform the problem through the levels shown on Fig. 1 until we reach the electron (or device) level, that is, the 200 million transistor, 1-GHz chip.

Along the way, the problem solution is first formulated as an algorithm to remove the unacceptable characteristics of natural language, such as ambiguity. It is then encoded in a mechanical language and compiled to the instruction set architecture (ISA) of the particular microprocessor. The ISA is the agreed upon interface that: 1) the compiled program uses to tell the microprocessor what it (the program) needs done and 2) the microprocessor uses to know what it must carry out

in behalf of the program. The ISA is implemented by a set of hardware structures collectively referred to as the microprocessor's microarchitecture. Each hardware structure and its interconnections are made of electronic digital circuits, which in turn are made of electronic devices.

When we say "microprocessor" today, we generally mean the shaded region of Fig. 1. That is, each microprocessor consists of circuits which implement hardware structures (collectively called the microarchitecture) which provide an interface (called the ISA) to the software. In the case of the personal computer, the ISA is the IA-32, and the microarchitecture is Intel's Pentium IV, or in earlier days, the Pentium III, Pentium II, Pentium Pro, 486, etc., or AMD's K-8, or in earlier days, K-7, K-6, etc.

There are other ISAs; for example, SPARC (from Sun Microsystems), Alpha (from Compaq), and Power-PC (from Motorola and IBM). Each has its own idiosyncrasies that makes it a better or worse interface for what a compiler can deliver, or how the microarchitecture can carry out the work. For each ISA, there are multiple distinct microarchitectures. We have mentioned several for the IA-32. For the Alpha, there are, for example, the 21064, 21164, and 21 264.

At each step in the hierarchy, from choice of algorithm, to language, to ISA to microarchitecture, to circuits, there are choices and therefore tradeoffs.

Often, but not always, the choice is between higher performance and lower cost. An analogy to the automobile is instructive. One can build a high-performance sports car that can go from 0 to 100 mph in practically 0 seconds. But, it will be very expensive. Or, one can build a very inexpensive automobile that could never get to 100 mph, but gets 100 miles to a gallon of gasoline. One does not get performance and economy. That is the tradeoff.

C. Design Points

The design of a microprocessor is about making relevant tradeoffs. We refer to the set of considerations, along with the relevant importance of each, as the "design point" for the microprocessor—that is, the characteristics that are most important to the use of the microprocessor, such that one is willing to be less concerned about other characteristics. Performance, cost, heat dissipation, and power consumption are examples of characteristics that strongly affect a design point. Another is "high availability"—one can design a microprocessor where the most important consideration is the requirement that the microprocessor never fail. Some customers are willing to accept lower performance or higher cost if they can be assured that the microprocessor will never fail. We call such a processor "fault tolerant," or highly available.

Other customers are willing to sacrifice a little performance, if it is combined with substantial savings in energy requirements. This design point has become more and more important as the power and energy requirements of the highest performance chips have become unacceptably larger and larger. Again, there is a tradeoff: highest performance or power awareness.

It is worth noting that "power awareness" is different from another important design point, "low power." There are many applications where the overriding consideration is that the microprocessor operate for a long period of time using a very small energy source.

In each case, it is usually the problem we are addressing (see again Fig. 1) which dictates the design point for the microprocessor, and the resulting tradeoffs that must be made.

D. Application Space

The word "Problem" in Fig. 1 is a catch-all for the Application Space, that is, the set of applications for which we wish to use microprocessors. This set is increasing at a phenomenal rate, and is expected to continue to do so. In fact, as long as people dream up more uses for computers, the need for microprocessors and the tradeoffs that each will make will continue to expand. That is, the application space (or, rather, the applications of central importance) drive the design point. We have already mentioned high-availability processors where the applications demand that the microprocessor never fails. And low power processors where the applications must be able to run for a long time on a small amount of energy.

Other examples of the application space that continue to drive the need for unique design points are the following:

1) scientific applications such as those whose computations control nuclear power plants, determine where to drill for oil, and predict the weather;
2) transaction-based applications such as those that handle ATM transfers and e-commerce business;
3) business data processing applications, such as those that handle inventory control, payrolls, IRS activity, and various personnel record keeping, whether the personnel are employees, students, or voters;
4) network applications, such as high-speed routing of Internet packets, that enable the connection of your home system to take advantage of the Internet;
5) guaranteed delivery (a.k.a. real time) applications that require the result of a computation by a certain critical deadline;
6) embedded applications, where the processor is a component of a larger system that is used to solve the (usually) dedicated application;
7) media applications such as those that decode video and audio files;
8) random software packages that desktop users would like to run on their PCs.

Each of these application areas has a very different set of characteristics. Each application area demands a different set of tradeoffs to be made in specifying the microprocessor to do the job.

E. The Basics of Processing

Simply put, a microprocessor processes instructions. To do this, it has to do three things: 1) supply instructions to the core of the processor where each instruction can do its job; 2)

supply data needed by each instruction; and 3) perform the operations required by each instruction.

F. Instruction Supply

In the early days of supplying instructions, one instruction was fetched at a time, decoded, and sent to the core for processing. As time has passed, the number that can be fetched at one time has grown from one to four, and shows signs of soon growing to six or eight. Three things can get in the way of fully supplying the core with instructions to process: instruction cache misses, fetch breaks, and conditional branch mispredictions. When an access to the instruction cache fails, the supply of instructions drops to zero until the cache miss is serviced. A fetch break occurs when an instruction being fetched is a taken branch, rendering useless all the subsequent instructions fetched in the same cycle, independent of the issue width. A conditional branch misprediction means that all instructions fetched since the mispredicted branch represent wasted effort, and must be thrown away before proceeding along the correct instruction path.

G. Data Supply

To supply data needed by an instruction, one needs the ability to have available an infinite supply of needed data, to supply it in zero time, and at reasonable cost. Real data storage can not accommodate these three requirements. The best we can do is a storage hierarchy, where a small amount of data can be accessed (on-chip) in one to three cycles, a lot more data can be accessed (also, on-chip) in ten to 16 cycles, and still more data can be accessed (off chip) in hundreds of cycles. The result is that real data storage suffers from latency to obtain a particular data element and the bandwidth necessary to move that data element from its location in the storage hierarchy to the core of the processor where it is needed.

As bad as this off-chip latency is today, it is getting worse all the time. Improvements in processor cycle time continue to grow at a much faster rate than memory cycle time. In a few years we expect to see off-chip data accesses to memory take thousands of processor cycles.

H. Instruction Processing

To perform the operations required by these instructions, one needs a sufficient number of functional units to process the data as soon as the data is available, and sufficient interconnections to instantly supply a result produced by one functional unit to the functional unit that needs it as a source. However, sufficient interconnections are not enough. As on-chip cycle times decrease, the latency required to forward results produced by functional units in one part the chip to functional units in other parts of the chip where these results are needed as source operands gets worse.

II. Agents for Evolution

Many things have aided the development of the microprocessor: The willingness of the buying public to scoop up what the vendors produce—without a market, we would have all gone home long ago. The creativity of engineers to come up with answers where there were problems—without solutions, there would be no evolution.

I submit that these things come second, and that the forcing functions (which I have called Agents for Evolution) have been new requirements, bottlenecks, and good fortune.

A. Agent I: New Requirements

Early microprocessors limited processing to what one could achieve by fetching one instruction each cycle, decoding that instruction and forwarding it and its data to the functional units in the core for processing. The demand for higher performance dictated that fetching one instruction each cycle was insufficient. The result is the wide-issue microprocessor, where the fetch mechanism allows multiple instructions to be fetched, decoded and issued to the execution core each cycle.

Another example, also due to requirements for high performance, was the need for more than one instruction to be processed at the same time. One can do only one ADD at a time if one has only one ALU. The result was the inclusion of multiple functional units in the execution core.

Today the prevailing new requirement involves power consumption, or what is being referred to as power-aware computing. The requirement is to provide the same level of computer performance as a previous design, while consuming a fraction of the power required for that previous design. Note that this is different from the low-power requirement of embedded processors, which has been an important design point for some time.

There is a good deal of sentiment that tomorrow's new requirement will involve the human interface, which is demanding more and more attention as computer/human interaction becomes more and more pervasive.

B. Agent II: Bottlenecks

We have identified above the three components of instruction processing (instruction supply, data supply, and carrying out the operations of the instruction), and what each entails. By far, most of the improvements to the microprocessor have come about due to attempts to eliminate bottlenecks that prevent these three components from doing their jobs.

For example, instruction supply requires fetching some number—today four—instructions each cycle. If these instructions were stored in memory, the time to fetch would be too long. The bottleneck is the slow memory. Thus, the instruction cache was invented.

If the hardware has the capability to fetch four instructions, but the second instruction is a conditional branch, only two—not four—instructions would be fetched. The bottleneck, which is caused by the conditional branch, is the arrangement of instructions in the order produced by the compiler, rather than the (dynamic) order in which the instructions are executed. The new feature, first added recently to the Pentium IV, is the Trace Cache, which stores instructions in the order in which they have recently been executed, not in the (static) order designated by the compiler.

Finally, if instructions are to be supplied every cycle, one has a problem when encountering a branch in that the condition that determines whether or not the branch should be taken is not yet known. One could wait for that condition to be resolved, temporarily halting the fetch of instructions until this resolution occurs. That bottleneck was alleviated by the introduction of branch predictors, which guess whether the branch should be taken or not, and immediately fetch according to this guess.

C. Agent III: Good Fortune

Good fortune happens when something causes a windfall which can then be used to provide additional features to the microprocessor. A good example of this is the technology shrink that allows a next implementation of a microprocessor to take up less space on the chip than the previous implementation did. With less space required by the old design, more space is available for doing other things. Two examples of other things that were introduced to the microprocessor in this way were the on-chip floating point accelerator in the mid 1980s and the multimedia instruction extension capability added on-chip in the late 1990s.

III. Evolution: From 1971 to Today

The microprocessor has evolved dramatically from the simple 2300 transistors of the Intel 4004 to what it is today. As suggested above, that evolution was due to several things. The result is that the Pentium IV of today bears little resemblance to the Intel 4004 of 1971.

Some examples of that evolution are the following.

A. Pipelining

Early microprocessors processed one instruction from fetch to retirement before starting on the next instruction. Pipelining, which had been around at least since the 1960s in mainframe computers, was an obvious solution to that performance bottleneck. Commercially viable microprocessors such as the Intel 8086 introduced the first step toward pipelining in the late 1970s by prefetching the next instruction while the current instruction was being executed.

B. On-Chip Caches

On-chip caches did not show up in microprocessors until a few years later. The latency to get instructions and data from off-chip memory to the on-chip processing elements was too long. The result: an on-chip cache. The first commercially viable microprocessor to exhibit an on-chip cache was the Motorola MC68020, in 1984. In a pipelined processor, it is useful to be able to fetch an instruction and fetch data in the same cycle without the bottleneck of contending for the one port to the cache. Once it became possible to put caches on the chip, the next step was to cache separately instructions and data. Among the first microprocessors to include separate on-chip instruction and data caches was Motorola's MC68030, in 1986.

A cache can either be fast or large, not both. Since the cache had to be fast, it had to also be small, resulting in too large a cache miss ratio. The problem with cache misses was the delay to go off-chip to satisfy the miss was too large. The result: two levels of cache on-chip, so that a miss in the fast, small first level cache could be satisfied by a larger, slower second level cache that was still a lot faster than going off-chip. This feature did not show up on microprocessors until the Alpha 21164 around 1994. Today, almost all high-performance microprocessors have two levels of cache.

C. Branch Prediction

The benefits of pipelining are lost if conditional branches produce pipeline stalls waiting for the condition on which the branch is based to be resolved. Hardware (run-time) branch predictors did not show up on the microprocessor chip until the early 1990s. Some of the early microprocessors to introduce run-time branch predictors were Motorola's MC88110, Digital's Alpha 21064, and Intel's Pentium.

D. On-Chip Floating Point Unit

Early microprocessors had a separate chip to handle floating point operations. As transistors got smaller and chips got larger, the transistor count reached the point where the floating point unit could be placed on the same chip with the main processing unit, utilizing "new" spare capacity and saving unnecessary off-chip communication. The Motorola MC88100 and the Intel 486 were two early chips to incorporate the floating point unit on the main processor chip in the late 1980s.

E. Additional Specialized Functional Units

Early microprocessors had one or very few functional units. As the number of transistors on a chip grew, so also the recognition that concurrent execution could be exploited with multiple functional units. First such things as separate address ALUs were added. Then a more sophisticated load/store functional unit containing structures like write buffers, a miss pending queue, and a mechanism for handling memory disambiguation became a part of the general microprocessor chip in the 1990s. Intel's i860, in 1986, was one of the first to have multiple specialized functional units, one to assist graphical processing, in addition to the floating point add and multiply units.

F. Out-of-Order Processing

The contract between the programmer/compiler and the microarchitecture requires that instructions must be carried out in the order specified by the translated program. This produces a bottleneck each time an instruction that can not be carried out prevents a subsequent instruction from being executed if the subsequent instruction has all that it needs to begin execution. The mechanism to get around this bottleneck, out-of-order processing, had been known since the mid-1960s on the IBM 360/91, for example. However, the mechanism was restricted to high-performance scientific computation, where it was claimed that being able to handle precise exceptions was not a critical requirement. Current acceptance of the IEEE Floating Point Standard by just

about all manufacturers suggests otherwise. Nonetheless, although out-of-order execution had been used on mainframes for 35 years, its use in combination with precise exception handling first showed up on microprocessors in the mid-1990s.

To accommodate out-of-order execution, the microprocessor adopted the register aliasing and reservation stations that had been used on earlier mainframes. To do so with precise exceptions, the microprocessor had to add the distinction between instruction execution and instruction retirement. Instructions were allowed to execute whenever their resources (data and functional units) became available, independent of their order in the program, but were forced to retire in the same order that they occurred in the executing program. That is, the internal microarchitecture could execute instructions out of order, but had to report results (i.e., change the permanent state of the computation) in the order the instructions occurred in the executing program. Doing this required a structure for restoring the state in the case of an exception. This state restoring mechanism is commonly manifested as a Reorder Buffer on most microprocessors today, and as a checkpoint retirement structure on a few. Though other microprocessors showed the beginnings of out-of-order execution earlier, the first to fully exploit the concept was the Pentium Pro, in 1995.

G. Clusters

Single die size continues to increase, feature size continues to decrease, and on-chip frequencies continue to increase. The result is that a value produced by a functional unit at one corner of the chip can not traverse the chip and be available as a source to a functional unit at the opposite corner of the chip in the next cycle. The result—partition the execution core into clusters so that most of the time, results produced by a functional unit in one cluster will be used by another functional unit in the same cluster. One still has the problem of knowing which cluster to steer a particular instruction to, but if successful, the inordinate multiple cycle delay caused by a result having to traverse a major part of the chip goes away. This feature first appeared on the Alpha 21264 in the late 1990s.

H. Chip Multiprocessor

An alternative use of the increasing richness of the die (many more transistors, combined with faster operating frequency) is to partition the chip into regions, with an identical processor occupying each region. The paradigm is referred to as CMP, for chip multiprocessor. For tasks that are easily partitionable into self-contained instruction streams, where substantial communication between the instruction streams is required, the CMP is a useful paradigm. It provides the added benefit of interprocessor communication occurring on-chip, where such communication is much faster than off-chip. IBM introduced this feature in 2000, with two processors on its G4 chip.

I. Simultaneous Multithreading

Instruction supply suffers when the instruction cache access results in a cache miss. A lot of capacity is wasted while waiting for the cache miss to be satisfied. Burton Smith in 1978 [3] suggested using that spare capacity to fetch from other instruction streams. The concept was first implemented on his Donelcor HEP. The concept did not show up in the microprocessor world until the 1990s, where it was expanded to allow fetching from alternate individual instruction streams in alternate cycles, but executing from all instruction streams concurrently in the same cycle, based upon the availability of the required data. The first microprocessor to implement this feature was the Pentium IV in 2000.

J. Fast Cores

On compute intensive tasks, the flow dependencies of source operands waiting for the results produced by earlier instructions can be a significant bottleneck. A solution—run the execution core at a frequency much faster than the rest of the microprocessor. The Pentium IV chip, introduced in 2000, has an operating frequency of 1.7 GHz, but an ALU that operates at 3.4 GHz.

IV. The One-Billion-Transistor-Chip Future

As we have said, within the current decade, process technology is promising one billion transistors on a single die, operating at a frequency of from 6 to 10 GHz. What will we do with all that capability?

Computer architects today do not agree on the answer. Some argue for extending the CMP idea that we described above. The argument is that with one billion transistors, we could put 100 microprocessors on a single chip, consisting of 10 million transistors each. The argument further states that a 10 million transistor processor is still very substantial, and building anything larger than that would just incur greater diminishing returns.

Others suggest an expanded use of simultaneous multithreading. They argue that many of the resources required for a CMP could be shared on a single processor SMT chip, freeing up the saved resources for other functionality such as larger caches, better branch predictors, more functional units, etc.

Some, including this author, note that while SMT is certainly an improvement over CMP with respect to shared resources, they both fall woefully short with respect to speeding up most of the important nonscientific benchmarks. The reason: Most of these benchmarks have the disappointing characteristic, re: SMT, of consisting of a single instruction stream. That and the notion that a very expensive chip ought to address problems not solvable by a multicomputer network made up of lots of smaller cheaper chips argue for using all billion transistors to produce a very high-powered uniprocessor.

Still others complain that since CAD tools are already unequal to the task of accurately validating our current chips, it is irresponsible to design even more complex ones. They

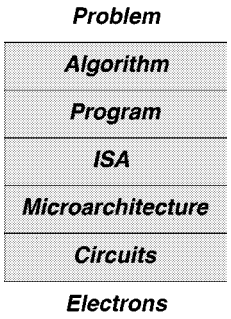

Fig. 2. The microprocessor tomorrow.

advocate a very simple core processor, combined with enormous on-chip caches.

Those that are cost-centered (a very different design point) recognize that higher levels of integration produce cheaper products, and suggest using the one billion transistors that will be available to put the entire nonaggressive system on the one chip.

These are the alternative suggestions, and I suspect each of them will show up in some product during the next ten years. My own preference is to use the billion transistors to evolve the highest performance uniprocessor that can process single-instruction-stream applications. Agents to catalyze this evolution remain as before: new requirements, bottlenecks and good fortune. Some ideas we are likely to see evolve are the following.

A. The New Microprocessor

Up to now, the microprocessor has been treated as shown in Fig. 1. But why so? If we take our levels of transformation and include the algorithm and language in the microprocessor, the microprocessor then becomes the thing that uses device technology to solve problems. See Fig. 2. Why is it not reasonable for someone wishing to design a microprocessor that addresses some point in the application space to take into account the special purpose algorithm needed to solve that problem, and embed that algorithm in the microprocessor? We do that today in low-cost embedded applications. Why not in high-performance requirements where we are willing to tolerate high costs?

This tailoring may take the form of reconfigurable logic, discussed below, special dedicated instructions in the ISA, or an integrated functional unit (like a DSP engine) provided on the chip.

B. A New Data Path

On-chip frequencies are expected to be so high that serious consideration must be given to the wire length of any signal on the chip. Some signals will require multiple cycles to traverse the chip, and one must examine carefully which signals will be allowed to do that. Most signals will probably not be allowed to. Therein lies the challenge: to redesign the data path in light of the new constraint of wire length.

C. Internal Fault-Tolerance

Another consequence of the increasing on-chip frequencies will be the susceptibility to soft errors—errors that will be created intermittently and infrequently due to the physical nature of the materials operating at the expected clock frequencies. Future microprocessors will have to provide functionality to check for and correct these soft errors as they occur.

D. Asynchronous and Synchronous Units Coexisting

Already, clock skew is a serious problem. At 6 GHz it is much worse. Admittedly, asynchronous structures are tougher to design, but they do get around the problem of a global clock that everything synchronizes on. And that is sufficiently important that the challenge is worth addressing. My expectation is that we will see structures that operate asynchronously for some fixed period of time (measured in clock cycles), after which they synchronize with the global clock. Different structures will require different amounts of time in which they need to operate asynchronously to get around their unique skew problems.

E. Different Cycle Times for Different Functions

For those structures that operate synchronously, it is not necessary that they all run at the rated frequency of the chip. Parts that don't need to go fast could be designed to go slow and save on power, for example. The future transistor budget can provide enormous flexibility to properly take advantage of the variability of the on-chip needs.

An ALU running at twice the frequency of the rest of the core is just the tip of the iceberg. The future microprocessor could use the clock intelligently, greater speed where needed, less speed where not needed, and very slow where speed is not in the critical path at all.

F. New Materials

I have no clue where these materials will come from, but Moore's law continues to prevail despite the doomsayers that show up every five years or so to spell its demise. Critical materials are needed vis-a-vis on-chip conductivity, and even more importantly, vis-a-vis power requirements and heat dissipation. So, in the spirit of unadulterated wishful thinking, I wish for engineering ingenuity to prevail again.

G. Expanded Use of Microcode

Off-chip bandwidth is expensive, on-chip bandwidth is plentiful. My expectation: we will more effectively harness on-chip bandwidth. The expanded use of microcode is a way to do that. For example, microcoded routines could exploit the spare capacity of underutilized functional units in a subordinate role to the primary instruction stream. We have coined the term subordinate simultaneous microthreading (SSMT) to reflect its role in an SMT machine [4]. These microcoded routines could perform dynamic recompilation, compute some compound instruction, tune

the cache replacement policy, or in some other way, perform a calculation that allows the primary instruction stream to execute faster.

H. Reconfigurable Logic

Consistent with Fig. 2, I expect many future microprocessors to address the requirements of specific applications. One application could make good use of some logic function that would be useless to other applications, while another application could make good use of a different logic function that would be useless to the first application. Perhaps both applications could be handled effectively by a microprocessor that had the capability to do run-time tailoring. That is, I think an on-chip structure, perhaps a low granularity FPGA, but more likely a higher granularity reconfigurable logic structure, will be common to future microprocessors.

I. Potpourri

Finally, I offer a list of features I expect to see in the high-performance microprocessor of 2008 or 2009, or whenever it is that process technology finally provides us with one billion transistors on a single silicon die.

1) Expanded use of the trace cache, where dynamic instruction stream segments will consist of far more than 8 instructions per entry, probably pre-scheduled with the help of the compiler (akin to the Block-structured ISA or the rePLay mechanism), but tuned at run-time.
2) On-chip microcode for using the spare capacity of the execution core to tune the on-chip hardware structures.
3) Dynamic recompilation of the executing program, probably carried out by the fill unit, or on-chip microcode will be commonplace.
4) Multiple (at least three) levels of cache with corresponding ISA additions (multiple prefetch and post-store instructions) to move data closer to the core and further way from the core in response to the core's need for that data.
5) Aggressive value prediction hardware, probably with a procedure level granularity, and corresponding compiler optimizations to aid its effectiveness.
6) Performance monitoring hardware to allow tuning the hardware at run-time to more effectively match the needs of the executing program.
7) An on-chip structure for monitoring and affecting the energy usage of the chip.

V. Conclusion

The microprocessor has enjoyed an exciting journey since its invention in 1971. Few technologies can boast the enormous strides it has made. Unfortunately, there are those who would argue that the end of this golden era is just around the corner. But such naysayers have been here before. They said the MIPS R2000 was all the microprocessor anyone would ever need in 1986, and ten years later they said the Intel Pentium Pro was all the microprocessor that anyone would ever need. The industry continues to do better, and the users of that technology continue to make use of that "better."

This is not to say that things will not change, that new ingenuity is no longer needed. Downstream, we may need a radical paradigm shift such as quantum computing to bail us out, but we are hardly constrained right now.

Sure we need to develop better CAD tools. Current CAD tools have trouble verifying the microprocessors of today, let alone the suggestions in this article. And, sure we need to think more broadly in our concept of the microprocessor (Fig. 2, for example). But the bottom line is that Moore's Law is alive and well, and still providing plenty of opportunity.

Acknowledgment

This introduction has benefited from many interactions over many years with many former and current students and many former and current colleagues. The draft published here has benefited explicitly from comments and criticisms of S. J. Patel, R. Belgard, and R. Ronen. It should be noted that an excellent and more detailed treatment of many of the issues touched on here are provided in a paper by Ronen and his colleagues at Intel [5], which this author recommends to the reader.

References

[1] H. Mazor, "The history of the microcomputer-invention and evolution," *Proc. IEEE*, vol. 83, pp. 1601–1608, Dec. 1995.
[2] Intel web site [Online]. Available: http://www.intel.com/pressroom/kits/quickrefyr.htm#1971.
[3] B. Smith, "A pipelined, shared resource MIMD computer," in *Proc. 1978 Int. Conf. Parallel Processing*, Aug. 1978, pp. 6–8.
[4] R. S. Chappell, J. Stark, S. P. Kim, S. K. Reinhardt, and Y. N. Patt, "Simultaneous subordinate microthreading (SSMT)," in *Proc. 26th Annu. Int. Symp. Computer Architecture*, May 1999, pp. 186–195.
[5] R. Ronen, A. Mendelson, K. Lai, S.-L. Lu, F. Pollack, and J. P. Shen, "Coming challenges in microarchitecture and architecture," *Proc. IEEE*, vol. 89, pp. 325–340, Mar. 2001.

Yale Patt (Fellow, IEEE) received the B.S. degree from Northeastern University and the M.S. and Ph.D. degrees from Stanford University, all in electrical engineering.

He is Professor of Electrical and Computer Engineering and the Ernest Cockrell, Jr. Centennial Chair at The University of Texas at Austin. He directs the Ph.D. research of nine students on problems relating to the implementation of high-performance microprocessors. He has been an active consultant to the microprocessor industry for more than 30 years. His particular love is teaching—both the first required computing course for freshmen and the advanced graduate courses in microarchitecture. He recently co-authored with S. J. Patel a textbook, *Introduction to Computing Systems: From Bits and Gates to C and Beyond* (New York: McGraw-Hill, 2000) which is a major departure from the traditional freshman course. It has already been adopted by more than 50 colleges and universities.

Dr. Patt has been awarded the IEEE/ACM Eckert Mauchly Award (1996), the IEEE Emmanuel R. Piore medal (1995), the IEEE Wallace W. McDowell medal (1999), and the ACM Karl V. Karlstrom Outstanding Educator award (2000). He is a Fellow of the ACM.