

# Customized Instruction-Sets For Embedded Processors

Joseph A. Fisher  
Hewlett-Packard Laboratories Cambridge  
1 Main St.  
Cambridge, MA 02142

jfisher@hpl.hp.com

## ABSTRACT

It is generally believed that there will be little more variety in CPU architectures, and thus the design of Instruction-set Architectures (ISAs) will have no role in the future of embedded CPU design. Nonetheless, it is argued in this paper that architectural variety will soon again become an important topic, with the major motivation being increased performance due to the customization of CPUs to their intended use. Five major barriers that could hinder customization are described, including the problems of existing binaries, toolchain development and maintenance costs, lost savings/higher chip cost due to the lower volumes of customized processors, added hardware development costs, and some factors related to the product development cycle for embedded products. Each is discussed, along with potential, sometimes surprising, solutions.

## Keywords

Embedded processors, custom processors, instruction-level parallelism, VLIW, mass customization of toolchains

## 1. INTRODUCTION—Architecturally Visible Customization

Instruction-Set Architectures (ISAs) are the visible instructions of a processor, often referred to as the contract between the hardware and the software: the instruction set implemented in the hardware and for which the software generates code. It is conventional wisdom that in the general-purpose computing world there will be little more variety in CPU architectures, and thus the design of ISAs will have no role in future CPU design. In fact, John Hennessy once told me that when he and David Patterson were revising their popular architecture book, several people suggested that they leave out the material on ISA design for general-purpose systems—that it is basically a dead subject.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
DAC 99, New Orleans, Louisiana  
©1999 ACM 1-58113-092-9/99/0006..\$5.00

Nonetheless, I believe that architectural variety will soon again become an important topic, and this short paper is an overview of the architecturally visible variation of microprocessor architectures. In it, I'm especially addressing embedded processors that are custom-designed for their intended use, but will address nonembedded CPUs as well. I will cover:

1. The motivation for doing this,
2. The barriers that stand in the way of doing this, and
3. Technologies that I believe will overcome these barriers.

At HP Labs Cambridge, we address everything I'll talk about. (And we do a lot in this area I can't talk about.)

### 1.1 Performance Need Encourages Variety

The major motivation for breaking the ISA is that doing so can sometimes lead to performance or performance/price gains, because, for example:

- In an embedded design, the specific application or application space is always known in advance
- Even in a general-purpose processor some markets will have an expected usage pattern
- There may be strict area and/or power requirements (e.g. in hand-held appliances or palmtop computers), and only by customizing can we meet performance goals, or
- We might have general architecturally visible improvements we want to incorporate, perhaps motivated by computer science, religion or semiconductor technology

### 1.2 Visible Changes

The sorts of visible changes that can be desirable, and can lead to greater performance, include, for example, having multiple visible ALUs, changing the number of registers, using "register clusters", having specialized ALUs (floating vs. not, special ops, etc.), changing latencies, saving power through visible control, visible instruction compression, and much more.

In embedded processing in particular, the opportunity—and competitive pressure—to customize is huge.

**Table 1. Pentium II price and performance.** Perf/Price is simply the ratio of the performance measure and the price. Notice the very high premium paid for the small performance improvement in CPUs on the high end. Prices from: PC Broker Inc, Updated 10/23/98, <http://www.pcbroker1.com/pricing/cpu.htm> . Performance from: Tom's Hardware Guide, <http://www.tomshardware.com>, same date.

Core Speed	Bus Speed	Family	Price	Business Winstone	Quake II	Winstone Perf/Price	Quake Perf/Price
266 MHz	66 MHz	Klamath	\$245	31.0	47	0.127	0.192
300 MHz	66 MHz	Klamath	\$268	33.1	52	0.124	0.194
333 MHz	66 MHz	Deschutes	\$299	35.0	56	0.117	0.187
350 MHz	100 MHz	Deschutes	\$349	36.7	60	0.105	0.172
400 MHz	100 MHz	Deschutes	\$596	39.5	66	0.066	0.111
450 MHz	100 MHz	Deschutes	\$799	41.3	69	0.052	0.086

### 1.3 Do Embedded Processors Need This?

Embedded processors for which performance matters are everywhere. Indeed, when I first started investigating embedded processing, with an eye towards making faster embedded processors, I assumed that I would find a few areas in which performance was a limiting factor. I assumed that, by-and-large, other factors (e.g. mechanical aspects) would dominate performance. How wrong I was! Indeed, I quickly found that processor performance was a key limitation in, for example, cellphones, video, disk controllers, medical devices, network devices, digital cameras & scanners, printers, etc.

Some need performance because they often have huge data sets and/or do very large computations (what we might fairly call embedded supercomputing). Others, as mentioned above, have low power or chip area requirements, but need performance nonetheless, and, finally, some need performance comparable to what might be achieved in an ASIC but can't suffer the loss of flexibility and time to market ASICs may cause.

### 1.4 Small Performance Improvements Matter

Because processor performance can often carry a large incremental value to an entire product, there is a large premium paid for performance at the high end. For example, consider Table 1, which shows the prices and performance of Pentium II's on the open market.

### 1.5 Barriers to Visible Variety

So why not do this? I can think of five big barriers:

**Barrier 1: Existing binaries barrier.** A general-purpose processor must be compatible; this is even becoming more relevant to embedded processing.

**Barrier 2: Toolchain development and maintenance costs; user familiarity.** Visible ISAs imply a new software toolchain for each processor.

**Barrier 3: Lost savings/higher chip cost due to lower volumes.** You lose the ability to leverage sales to many different users.

**Barrier 4: Hardware development costs.** Each variant processor needs a new chip design.

**Barrier 5: The product development cycle for embedded products.** Users don't develop products in a way that is very friendly to customization.

Most of this paper is a discussion of these barriers and some solutions.

## 2. Barrier 1: The Existing Binaries Problem

Until recently, new companies, or companies that weren't previously selling computers, entered the general-purpose computer business by building and selling computers on which no existing binaries would run efficiently. This was true of IBM, DEC, HP, Prime, Data General, Univac, CDC and so on. Indeed, probably more than 100 companies made "significant dents" in the market this way.

Although that was the universal entry route in the computer industry for forty years, in the mid-1980's it hit a wall. In fact, in informally surveying companies and the year of introduction of their first (non-compatible) ISA, I could find an average of 2-3 new companies each year that followed this model from 1946 forward. But then, presumably because of the expectation of what a computer would do—caused by the PC revolution—it became virtually impossible to make a dent in the market this way. Indeed, two well-financed attempts, Next, and, very recently, BeBox, were unable to survive despite their tremendous financial resources.

In a telling 1995 article in The New York Times [3], John Markoff wrote:

A new PC demonstrated by former Apple Computer executive Jean-Louis Gasse was impressive enough to elicit a standing ovation from the typically skeptical crowd of some 500 technology executives at the Agenda conference. With its own software operating system, *Gassee's BeBox computer will be incompatible with everything now on the market, but the audience still seemed enthusiastic.* [italics mine]

But, by 1997, the company had to cease production of the BeBox and concentrate solely on software development.

## 2.1 Performance Drives "ISA Drift"

I believe that the constraint of total binary compatibility is too severe for anyone to live with. For example, in Intel's 32-bit architectures we have seen or will see the following ISA variants:

- Hardware floating-point vs. software floating point
- MMX vs. no MMX
- And, soon, "Katmai New Instructions"

The temptation to do this is often irresistible. As put in an Intel white paper, published on the web, "Intel engineers designed Katmai New Instructions to add compelling benefits in various software categories, some of which are described in this paper. We expect the benefits of Katmai New Instructions to extend beyond the categories described here and are still working on discovering all the ways that developers can utilize the technology."

Because binary compatibility is so important, Intel has provided programmers with mechanisms that allow them to envelope code with a software process that selects different versions of the code, depending upon what ISA is present. This is done out of necessity, and is something that I find quite clumsy. Using methods mentioned below, one will soon be able to make these variants appear truly binary compatible, changing the meaning of binary compatibility to something significantly broader than what we mean today.

Techniques are emerging that will alter binaries after they're distributed. This will happen because:

- As we just saw, even the most mainstream ISA's cannot resist small amounts of visible variety
- We're going to see binaries with very poor performance distributed over the web (e.g. as Java VM code)
- The advantages of altering binaries while they're loaded and while they're running are huge.

This can (and I believe certainly will) lead to the opportunity to customize, and will have the slow effect of causing what I call ISA Drift. I believe that, sometime in the future, architectures will become families of ISAs that are what we would today call mutually incompatible. Even the highly compatible IA-32 is today a family that displays this attribute to a very small degree. Given the presence of appropriate techniques, one can picture this phenomenon occurring to a dramatically greater extent.

## 2.2 The Solution: Making "ISA Drift" Acceptable On A Large Scale

The key to this happening is the maturity of and research into techniques that operate on binaries after they are distributed. These techniques have exploded in the past 3-4 years, examples are:

- Object code translation/Software migration (this has been done for years, but not in any way that produces high performance binaries)
- Code caching
- Dynamic compiling
- Dynamic optimization
- Statistical profiling
- Link/load/install time techniques
- ... and so on

The result of these technologies is that, slowly and incrementally, hardware designers will see the things already being done at run-time and will take advantage of them to customize and improve. This will lead, perhaps without designers even thinking in these terms, to families of compatible processors that would today be incompatible. These will go by a single name, but might have dramatic differences from model-to-model. Indeed, all of the things listed in Section 1.2 as areas of potential customization could be candidates in this paradigm.

In addition to allowing families of compatible processors that would today be incompatible, this will enable far simpler Instruction-level Parallelism (ILP). In about the chip area required for a RISC processor, we can build a 4-issue customized VLIW. No area is used to maintain the compatibility that the run-time techniques maintain. In addition to the area saved, critical paths in the hardware are far shorter, the cycle time faster, etc. But the ISA is unique, and, by today's standards, incompatible.

A short paper of mine discussing these ideas can be found in [1].

## 2.3 Even in the Embedded Market, Binary Compatibility Can Be Important

In general, embedded CPUs are placed on boards with entirely new systems, built for the product at hand. Recompile is a given, and, thus, binary compatibility shouldn't be an issue. However, when the ISA changes, certain 3rd party programs, especially real time operating systems, will change in places, or even require assembly code, to run properly

Most of the effect of this is upon the toolchain (discussed later), but there are other techniques that must be brought to bear. The trend today is for this to be an increasingly important effect, but I predict that where it is relevant, ISA drift will occur. We will see a lot of ISA variety, just as in the case of general-purpose processors.

It's worth commenting that by "embedded systems" I don't mean those found in hand-held or "palmtop" PCs. Right now, there is quite a bit of CPU chaos in the handheld and palmtop CPU

markets, with the market split mostly among the Hitachi SH3, MIPS, ARM, and the MC68000. But to me, these are really small laptops! I believe it is the case that we are in a small window of time before these become typical laptops, just a little scaled down, running standard processors, OS's and applications, but with different input and output. Even if that's not the case, given that they are capable of having varied programs distributed for them, how long will it be before the same pressures of standardization make this a one CPU family market?

To me, an "embedded processor" is buried silently is some device, mostly doing one thing over and over. This is a very fuzzy line, but I put printer CPUs on one side (not to mention clock radio CPUs), and handheld and palmtop PCs on the other.

### **3. Barrier 2: Software Toolchain Costs and Familiarity**

In the truly embedded CPU world, the above run-time techniques are not likely to be as important as in the general-purpose CPU world, though I predict they will have their place, for example in mitigating the problem of running a real-time operating systems on a new ISA family member.

A similar set of problems evolves from questions of software toolchain cost and familiarity. A new ISA implies a new toolchain. But users don't want to have to change toolchains, having grown familiar with the old ones, and the cost of a new toolchain is high.

#### **3.1 The Solution: "Mass Customization" of Toolchains**

The solution is to automate all aspects of the variation of ISAs. In other words, treat the automation itself as a science. This then follows a common pattern for highly automated manufacturing:

1. At first, everything that's built is one-of-a-kind
2. After manufacturing becomes highly automated, only a few different styles are manufactured
3. Later, automation is introduced within the manufacturing process itself, and a great deal of variety becomes possible

Henry Ford is often quoted as having said, "The Customer Can Have Any Color He Wants So Long As It's Black." Today, especially with the presence of the Web, there is a movement towards the automation of customization itself. This movement has been called "mass customization", where the irony of the oxymoron is clearly intended. As an example, from Fortune Magazine [4]:

A silent revolution is stirring in the way things are made and services are delivered. Companies with millions of customers are starting to build products designed just for you. You can, of course, buy a Dell computer assembled to your exact specifications. And you can buy a pair of Levi's cut to fit your body. But you can also buy pills with the exact blend of vitamins, minerals, and herbs that you like, glasses molded to fit your face precisely, CDs with music tracks that you choose, cosmetics mixed to match your skin tone, textbooks whose chapters are picked out by your professor, a loan structured to meet your financial

profile, or a night at a hotel where every employee knows your favorite wine. And if your child does not like any of Mattel's 125 different Barbie dolls, she will soon be able to design her own.

Welcome to the world of mass customization, where mass-market goods and services are uniquely tailored to the needs of the individuals who buy them. Companies as diverse as BMW, Dell Computer, Levi Strauss, Mattel, McGraw-Hill, Wells Fargo, and a slew of leading Web businesses are adopting mass customization to maintain or obtain a competitive edge. Many are just beginning to dabble, but the direction in which they are headed is clear. Mass customization is more than just a manufacturing process, logistics system, or marketing strategy. It could well be the organizing principle of business in the next century, just as mass production was the organizing principle in this one.

The key to doing this with toolchains is discipline. To quote from the Fisher, Faraboschi and Desoli paper in Micro-29 [2], the toolchain we use:

...generates code from table-driven architectural descriptions in the following sense: if you have a description of an architecture for which you are generating good code, you can change most of the "normal" architectural parameters to produce a new model, and continue to generate good code.

We thus are able to use it to explore a design space of architectures to fit one to a given application. By building scalable and customizable toolchains, we allow the user to do software development relative to toolchain, not the hardware. The software tool base doesn't vary with new ISAs—it presents a single family view to programmers. The most difficult part is a compiler backend that doesn't compromise ILP because of scalability.

This is easy to talk about, but hard to implement! It's especially hard to write a state-of-the-art compiler that finds ILP and does all other desirably optimizations, while maintaining this philosophy. In order to be successful at this, here's what you probably want to do:

1. All toolchain changes support all architectures in range
2. Testing methodology uses architectures as if they were test programs (thus NxM tests)
3. Preserve C semantics as best you can (even when you can't really, because you've customized in a way that doesn't work in C)
4. Fast and accurate simulation of everything (use direct execution simulation)

This requires enormous discipline. You're always faced with the choice of doing something quick that makes what you're building work on this architecture, or, alternatively, doing something painstaking to preserve the generality. In a commercial environment, the temptation to do the former, in the face of deadlines, product schedules, etc., is huge.

#### **4. Barrier 3: Can Low Volume Customized Processors Be Competitive?**

Suppose a product designer is choosing between:

1. A simple customized processor as I'm describing, or
2. A more complex, much larger mass-market, high performance embedded microprocessor

The mass-market processor gets a big volume advantage, because, if it's a successful chip, many other buyers will select the same chip. As a result, silicon processes can be tuned up, big runs can be done, with a dramatic effect on yield. Chips can be cranked out like jellybeans. If it had volume as small as the custom processor, the mass-market processor might cost twice as much or more than a simpler customized processor. But with its much larger volume it might cost less, and have as high performance.

##### **4.1 The Solution: System-On-Chip Changes the Equation**

However, there is a sea change occurring in the embedded processor market: nearly everyone is moving towards system-on-chip.

Economics are beginning to dictate a greater degree of integration, with the processor becoming simply a "processor core", found in one portion of the chip. This is mandated, because savings—due to reduced component count, board area, etc.—are greater than loss of mass-market volumes on the processor alone. Using this methodology, every chip is made for the anticipated use only, the experience curve is not at all as important, and thus customized processors are more price competitive.

#### **5. Barrier 4: The Hardware Design Cost Barrier**

Of these barriers, hardware design cost is the one I have the least to say about. This is partly because it's not something we have spent a lot of time on at Hewlett-Packard Laboratories Cambridge, but also because I am not in a position to talk about what we have done, for proprietary reasons.

However, there are some things to mention in this context:

1. Not having to pay much attention to binary compatibility allows simpler and more flexible design. It is far easier to change the number of registers, number and types of ALUs, etc. than it is to change a complex control unit designed to extract ILP while maintaining binary compatibility.
2. As with toolchains, a key is to have the "religion" of designing for families of ISAs. That is, one should always be cognizant of the fact that the design will be changed, hopefully at minimal cost.
3. I believe that reconfigurable hardware is an important research area. But it seems to be huge factors away from success—I don't see it making a dent for a long time. Given that so many commercial efforts have sprung up to exploit this technology, it stands to reason that I'm wrong, but who knows.

#### **6. Barrier 5: Product Development Cycles and Customization**

A significant problem, which I will not cover in detail in this paper, is that of the timing of code development vs. processor choice. For various reasons, hardware is tested before it's shipped to a far greater extent than software, or even "firmware". Some of the reasons for this are cultural, but some are due to the need to deal with factors like RFI, or the constraints of certifying agencies, or even the relative plasticity of software.

Given that processor choices are usually bound ½ to 1½ years ahead of first shipment, and that the software can change significantly in that period, it is often hard to know the application one is customizing for. How, then, can one customize for a specific use?

##### **6.1 The Solution: Tailor To an Application Area, Not an Application**

In my experience, most products are designed with a core of things that are compute-intensive that they must do well, and some other, less predictable, compute-intensive things that they may have to do well, directly or on a variant of. To design a processor for a such a product, the key is to balance the specificity that one can be sure of, against having enough slightly more general horsepower for the uncertain things. Even there, however, there are core capabilities that the application is likely to take advantage of, and key customizations that can be used to good effect.

#### **7. Summary**

In this paper I have outlined the factors that I believe will cause instruction-set architectures to become performance-driven families of what we would now call incompatible ISAs. One of the most important enablers of this will be the "mass customization" of software toolchains. With a suitably disciplined approach to this, many products will be built with CPUs that are customized their use, while the product development will be done in a familiar way, much as if only subsequent generations of a single ISA were involved.

#### **8. ACKNOWLEDGMENTS**

Many of the views in this talk have been influenced by the work of, and by conversations with, Paolo Faraboschi, Vasanth Bala, Stefan Freudenberger, Giuseppe Desoli, and Geoffrey Brown.

#### **9. REFERENCES**

- [1] Fisher, J. A. Walk-Time Techniques: Catalyst for Architectural Change. *Computer*, 30, 9 (September 1997), 40-42.
- [2] Fisher, J. A., Faraboschi, P., and Desoli, G. Custom-Fit Processors: Letting Applications Define Architectures. *International Symposium on Microarchitecture*, Micro-29, Paris, France, 1996, 324-335.
- [3] John Markoff, New Computer Dazzles a Jaded Industry Crowd. *The New York Times*, October 4, 1995, D6.
- [4] Erick Schonfeld, The Customized, Digitized, Have-It-Your-Way Economy. *Fortune Magazine*, 138, 6, September 28, 1998.