THE LISP MACHINE: NOBLE EXPERIMENT OR FABULOUS FAILURE?

P. T. Withington
Symbolics, Inc.

The "Lisp Machine", a custom computer work-station designed specifically for the execution of Lisp, has been an important part of the Lisp tradition for 20 years. Recently, the Lisp Machine has been deprecated in view of the demise of many Lisp Machine vendors, the swing towards standardization, and the advances that reduced instruction set (RISC) architectures have brought. But rumors of its death are greatly exaggerated.

Unlike most commercial computer languages, Lisp has always been a language of ideals. Its roots are in the theory of lambda-calculus. Whereas other languages burden the programmer with implementational gaps in their abstractions, Lisp has always had the aim of supporting complete abstractions.[1] This idealistic bent of Lisp has led to it often being the language of choice for computer-oriented research in universities and industry. Removing the more mundane difficulties of computer programming allowed researchers to experiment with super-complex (at the time) technologies such as windowing, presentation managers, object-oriented programming, integrated programming environments, computer music, integrated-circuit design, and of course Artificial Intelligence (AI).

But, Lisp's purity did not come without a price. The choice by many languages to expose implementational limitations is often a choice of efficiency. The speed of the normal case is optimized at the risk of the abnormal case going undetected. Lisp, on the other hand, guarantees the unusual as well as the usual will be dealt with uniformly. It must always be on its guard: every operation must be checked for exceptions. As a consequence, Lisp on conventional machines has historically been ponderous to work with.

In the early 1970's several groups of researchers utilized two novel hardware technologies to improve the efficiency of Lisp: tagged architectures and micro-programming. Tagged architectures store type information with each data word, tracking dynamically changing types (a prominent feature of the Lisp language) with essentially no overhead. Micro-programming allows a simpler compiler by implementing complex instructions that more closely match the high-level semantics of the Lisp language. Experimenting with these techniques eventually led to commercial introductions of "Lisp Machines".

Other novel features of these machines include:

---

[1] For example, in 'C' the answer to (1-2/3)*3 is either 3 (!) or perhaps 0.99999994, in Lisp it is 1; in 'C' the answer to 2147483647+2147483647 is -2 (!), in Lisp it is 4294967294. Because 'C' is transparent to implementation detail, it may be that some 'C' implementations will give the right answer (or a different wrong answer) for these particular examples; nonetheless, the programmer's task is often more difficult because of this transparency. Lisp also has finite limits in its implementation, but they are usually large enough to not be of practical importance.

- The "stack cache"— Execution of Lisp is stack-oriented. By caching the top elements of the stack, efficiency approaching that of register-based execution is achieved without complex register allocation algorithms in the compiler. The stack cache also supports fast function call and return, as if the function arguments and values are passed in registers.

- Garbage-collection support— Lisp's storage management system is often implemented as a "garbage collector", which automatically reclaims unused objects. The Lisp Machine virtual memory hardware, in concert with the type tags, tracks object-reference loads and stores so the software can determine quickly which objects are in use and efficiently reclaim those that are not.

- Instruction emulation— When the Lisp Machine hardware encounters an exceptional situation (for example, an integer arithmetic operation that exceeds the hardware imposed implementation limit or an operation on a software-defined type) the hardware traps out to a software "emulator" subroutine. This subroutine implements the full semantics of the operation, as specified by Lisp, as though it is being handled by the machine instruction.

Early Lisp Machines implemented their micro-programmed architectures with a writable control store, which meant the instruction set, and to a certain extent other architectural features of the machine, could be changed by simply writing, compiling, and loading new micro-code. This flexibility led to further experimentation and evolution of the hardware support for Lisp.

In the late 70's and early 80's both AI and the Lisp Machine enjoyed a brief but heady vogue. Two small companies, Symbolics and Lisp Machines, Inc. (LMI), were founded to build Lisp Machines; Symbolics was eventually taken public. Xerox and Texas Instruments (TI) also entered the market. As late as 1986, Integrated Inference Machines (IIM) entered the Lisp Machine market. Computer science researchers, eager to speed their experiments, snapped up the machines. Companies bought them for their R&D labs in hopes of solving a wide array of problems from analyzing stock trades, to interpreting seismological data, to scheduling airlines, to evaluating loan applications. At the time, Lisp Machines were the only computer work-station of significant power and the only economic solution to efficiently developing and running Lisp programs.

The Lisp Machines produced by these companies went through several evolutionary generations: starting at the high price of $150,000 and implemented in TTL in a box rivalling the VAX/780 in size and power consumption[2] and eventually being delivered as 1- or 2-chip VLSI implementations on $10,000 add-in boards for the Apple MacIntosh. With this latest generation of Lisp Machines, Symbolics was satisfied that its instruction set had evolved sufficiently to commit it to mask-programmed ROM. However, the small size of the market for these machines has meant they are unable to take advantage of the cutting-edge hardware technology of commodity machines. They always lag by a few generations, where design and production costs are more reasonable.

Initially, the small companies received great exposure in the popular press. Because they represented a "pure play"[3] in AI, they were the darlings of Wall Street. But today, only

[2]It is amusing to realize that a machine of this size and price was ever considered a work-station.

[3]An investment term meaning the company is in a single market and can be expected to track that market's fortunes more closely than a diversified company might.

Symbolics remains in the market trying to sell Lisp Machine work-stations, at about 1/6 the size of its heyday. The popular opinion is that AI technology was oversold and lost its credibility; while at the same time, Unix was rising to meet the need for standards and RISC computers, with their simple instructions that can be executed at fantastic rates, obviated the need for a custom machine to implement Lisp. The press was as quick to damn the Lisp Machine companies as it had been to praise them, when they did not meet over-inflated expectations.

A more charitable interpretation of the same facts might be that the Lisp Machine market (by its esoteric nature) is small, and given normal start-up statistics, is doing surprisingly well. The Lisp Machine was a pioneer in the early days of work-station technology. It had the now standard high-resolution bit-mapped display, mouse pointing device, large virtual memory and local disk; it even had 16-bit digital stereo sound! Some of these innovations contributed to the evolution of Lisp and its associated technologies. Unfortunately, the similarity of these features to those eventually found on general-purpose work-stations led to confusion over what the market for the Lisp Machine was. The exotic features of the Lisp Machine that made it ideal for running Lisp and the dream-machine of many computer researchers had little or no value in the general-purpose market. The success it enjoyed as a Lisp engineering work-station faltered at attempts to market it as a computer-aided software engineering (CASE) work-station.

At the very least, one can say that the Lisp Machine was there for Lisp when it threatened to drown in its own idealism, due to the primitive power of the hardware architectures of its time. The attention the Lisp Machine companies drew may have added life to the Lisp market; it certainly added to the investment in Lisp research. In their heyday, the companies making Lisp Machines attracted top-notch talent and funded innovative research in many areas of computer science, both hardware and software.

However, the Lisp language did suffer in the commercial market from its association with what many customers have come to regard as the "snake oil" of AI. Because its only perceived redeeming value was to run Lisp, the Lisp Machine suffered doubly so. Despite the grim appearance of the future of the Lisp Machine, there remain small groups of zealots who will not part with their Lisp machines without a fight. They battle "MIS" departments and often end up closeting their Lisp Machines, so they won't be pestered about their non-approved equipment.

It has been argued that while Lisp may represent the ideal solution to a problem, the ideal solution is not always the economic solution, even more so when it requires a custom machine to run it. Today, there is a wide choice of general-purpose computer work-stations of similar power to the Lisp Machine, most with competitive Lisp implementations. Many of the software technologies that were considered research areas when the Lisp Machine was introduced have been codified to the point that they are amenable to implementations using commodity software and hardware. Others will follow. This trend has led many to believe there is no longer a need for the power of Lisp and even less a need for the Lisp Machine. What is often not considered, however, is whether these technologies would be where they are today (or would have been explored at all) in the absence of the Lisp Machine. Despite the trend toward commodity software and hardware, there continue to be super-complex and evolutionary problems where the Lisp Machine is the ideal solution and may well also be the most economic. The future is likely to bring more.

Compiler technology has evolved significantly since the birth of the Lisp Machine. Many of the problems that the micro-code of the Lisp Machine solved for the compiler writer can now be dealt with. The RISC revolution, which depends on a super-simple instruction set for its phenomenal execution rate, has forced the compiler writer to deal with similar problems, even in conventional languages. But the companies that market Lisp on RISC machines (perhaps begrudgingly) admit that despite their hardware technology lag, Lisp Machines are still competitive or surpass the latest RISC implementations in both Lisp benchmarks and, more often, high-end Lisp applications.

Today, most RISC hardware architectures support either large register sets or register windows, which bear great similarity to a stack cache. Close examination of some of the newest RISC architectures will reveal support for tagged data, to efficiently implement generic arithmetic operations. Architecture research papers continue to evaluate the merits of read and write "barriers", pioneered by Lisp Machine garbage collectors to track object references and speed automatic storage reclamation, and "fast traps" to allow expeditious handling of exceptional conditions in a manner similar to the Lisp Machine instruction emulation.

The current commercial versions of the Lisp machine have reached about the level of integration that floating-point and vector co-processors had in the early 80's. It remains to be seen if they will end up as simply a footnote in the history of Lisp, or if they will continue to evolve and their best ideas live on as an integral part of future commercial computer hardware.