Lisp machines and the analysis of their high-level language computer architecture

Dominic Dabish and Evan Bradley
School of Engineering and Computer Science
Oakland University, 2200 N Squirrel Rd, Rochester, MI 48309
{dadabish, edbradley}@oakland.edu

***Abstract*—The high-level capabilities that Lisp affords programmers makes it very useful in certain applications, but also causes it to run slow on most computers, which were designed for different types of executions. To account for the incompatibility between Lisp and the hardware on which it is executed, computers called Lisp machines have been designed specifically to run Lisp programs by designing the processor hardware to accommodate the language features Lisp possesses. Achieving greater efficiency in running Lisp was accomplished using various methods, both in the hardware and in how data would be represented in the software. Hardware alterations largely consist of changes to the circuitry in the processor itself as well as the number and purpose of processors in the machine. Software changes mostly correspond to the representation of lists, the main data structure in Lisp. Lisp machines were largely deprecated in the 1990s, giving way to traditional computers which were fast enough to run Lisp with reasonable speed. Despite this, Lisp machines remain a popular subject for study and discussion amongst those who are familiar with Lisp.**

Keywords*—*Lisp, Lisp Machine, computer engineering, vintage computing, computer organization.

I. Introduction

The history of Lisp begins with Alonzo Church's introduction of lambda calculus in the 1930s, where Church attempted to reconcile mathematics and computation. Lisp takes many notational cues from lambda calculus, and in a sense serves as a practical implementation of the ideas Church presented. Lisp machines were created as a response to the decreasing costs and increasing complexity of computers in the 1970s, and were rendered virtually defunct in the 1990s by the same conditions that had created them.

A. Early history of Lisp

John McCarthy created Lisp while working to create a new computer language to replace languages like the Information Processing Language (IPL) and Fortran, which he saw as cumbersome and insufficient for programming artificial intelligence (AI) [1]. He began work on what was then called lisp LISP, or the LISt Processing language, in 1958 while working at the Massachusetts Institute of Technology (MIT), and published a Communications of ACM paper in April of 1960 that detailed the basic workings of the language he constructed [1,2]. By April 1958, Steve Russell had implemented some Lisp functionality in machine language on the IBM 704, much to the surprise of McCarthy, who believed his work to be entirely theoretical before seeing the implementation. Development of Lisp began to occur in a more concrete fashion after the actual implementation of Lisp code on the IBM 704. McCarthy and his team then began to focus on creating a usable version of Lisp in addition to creating something that works in theory [1].

In the late 1960s, Jon L. White published a paper describing MACLISP, a variation of Lisp that ran on the Digital Equipment Corporation's PDP-6 [1]. This was followed by the creation numerous other Lisp dialects by individuals and groups who were not directly related to McCarthy. Many of these were combined into Common Lisp in the 1980s [Wikipedia]. One of these dialects was Scheme, which, along with Common Lisp, continues to be one of the two major Lisp dialects to date. It was created in 1975 by Guy L. Steele and Gerald Jay Sussman at the MIT AI Lab, and on whole is much simpler than Common Lisp.

B. History of Lisp machines

Lisp machines were some of the first single-user workstations, which at the time were an exception to the common timesharing systems where multiple users would connect to a single machine through terminal stations. This change came in part due to Lisp's nature of consuming the entirety of the computer's resources, which made its use difficult on multiple-user systems [3]. Two main companies emerged from the increasing demand for Lisp Machines: Symbolics and Lisp Machines, Incorporated. Larger companies began to produce Lisp Machines as well, including Xerox and Texas Instruments.

These machines were often seen as valuable for not just their ability to run Lisp, but for the advanced features they offered in terms of graphical user interfaces (GUIs) and programmability due to the flexibility that Lisp afforded programmers. Despite their advances in usability, they were often behind in terms of hardware due to their niche audience, they sold to a much smaller market than general-purpose machines, which quickly raced past Lisp machines in terms of hardware capability

This paper was submitted for review on March 25, 2016. It was produced with help from Oakland University's subscription to publication sources with information on Lisp and Lisp machines.

Dominic and Evan are Computer Science majors at Oakland University in Rochester, MI, 48309.

[4]. Eventually general-purpose computers became faster at running Lisp despite the hardware incompatibilities, and in combination with the slow development of AI in the early 1990s, most Lisp Machine vendors either left the market or went bankrupt.

II. The Basics of Lisp

III. Identifiable Problems in the Execution of Lisp Code

In A 1987 survey of Lisp machines, Andrew R. Pleszkun and Matthew J. Thazhuthaveetil concluded that there were four main problems that engineers had to address when designing a Lisp machine: function calls, environment maintenance, efficient list representation, and heap performance [6].

IV. A Proposed Lisp-Based Processor by Steele and Sussman

Steele and Sussman, creators of the popular Lisp dialect Scheme, proposed in March 1979 a processor that they suggested would allow for the efficient execution of Lisp programs under a dialect of Lisp they had created [5]. While it is uncertain whether their ideas were directly implemented in an actual Lisp machine, they provide solutions to the environment maintenance and efficient list representation problems proposed by Pleszkun and Thazhuthaveetil, providing a great degree of detail in describing their implementation.

Steele and Sussman start their paper by introducing the basics of the Lisp dialect they want their processor to run. The representation of data structures in Lisp and how the program is traversed are of particular concern, as Lisp's code is itself a data structure. Therefore, their processor is built specifically for traversing the Lisp data structure in an efficient manner. To achieve this, they introduce a Lisp interpreter which functions recursively, operating on a small set of registers to hold pointers to a list memory system which holds the stack for the program's state information between procedure invocations. The evaluation component of the interpreter uses five global variables that simulate registers in a normal machine to hold different components of the program's state. They describe this system as a state machine implementation where the program moves between states as procedures are invoked [5].

To reduce evaluation times for atoms and lookup times for variables, the representation of list data is altered slightly so that each pointer contains a field that specifies how the data should be handled. This happens at compilation time, such that the code does not be modified to include these types, and the compiler will appropriately tag the data [5]. Refer to figure 1 for a graphical representation of this format generated from a small code sample.

The authors then propose that by combining these two ideas, an efficient interpreter can be designed directly into the hardware. The hardware itself is only concerned with the traversing of the Lisp structure, and so all data is accessed through pointers to memory, which are stored in the processor's registers. Commercially-available memory was only available in a form that stored linearly-indexed vectors rather than linked-lists at the time, so a storage manager was needed

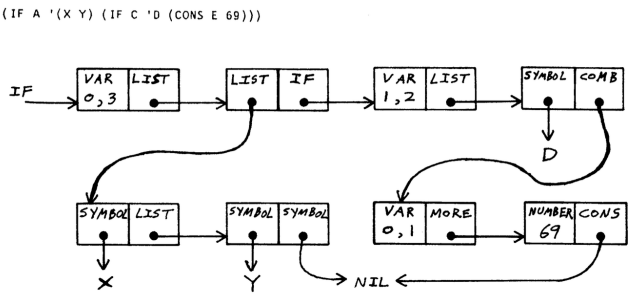

Fig. 1. Example of the tagged data representation used in the Steele an Sussman processor, intended to improve evaluation and lookup efficiency [5].

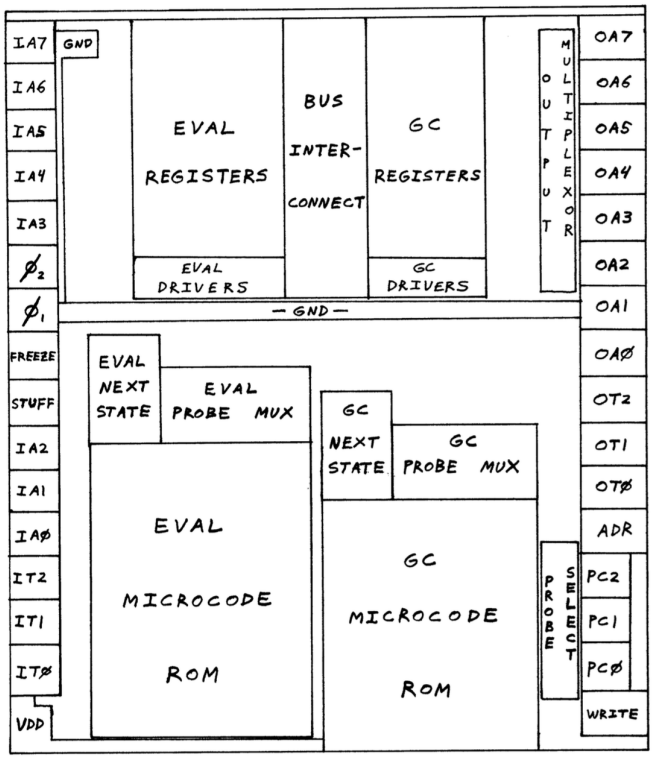

Fig. 2. An overview diagram of the processor design proposed by Steele and Sussman. This design was fabricated to test the author's proposed system, but was likely never implemented commercially [5].

to interface between the memory hardware and the processor to ensure the processor could properly utilize the memory. The registers in the proposed processor would hold an 8-bit address and 3-bit type value for each pointer [5].

A physical layout was given for this processor, wherein the evaluator and storage manager exist in the form of microcode implemented on read-only memory with a register above each functioning as a form of program counter. Above these are the registers for each unit connected by a bus to transfer information between them. The input connections are supplied on the left and output connections supplied on the right [5]. A diagram representing the layout of their processor is displayed

in figure 2.

As noted by Steele and Sussman, their processor does not include a meaningful arithmetic logic unit (ALU), and so can not be independent in a system that needs to process real data. However, it contains basic structures for logic and can perform incrementation, and so contains the necessary components to run any Lisp program in the described style, due to the simple nature of their proposed Lisp dialect. It is suggested that any actual calculations be done with modules closer to the memory, which most likely contain processors with more sophisticated ALUs [5]. As such, Lisp machines which implemented designs similar to this would likely take the form of a multi-processor system.

APPENDIX A
PROOF OF THE FIRST ZONKLAR EQUATION

Some text for the appendix.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.