DIALISP: A Lisp Machine

Gheorghe M. Ştefan

Dept. of Electronic Devices, Circuits and Architectures, Politehnica Univ. of Bucharest,
1-3 Maniu Bd., Bucharest, Romania

Abstract

At the beginning of the 1980s, the design and production of LISP machines experienced a flourishing without precedent or traces. In the Bucharest Polytechnic, at the Faculty of Electronics, a LISP machine was designed and built, which was then put into production at the Bucharest company FEPER. We further describe the architectural experiment occasioned by this achievement.

1 Introduction

In the 1960s and 1970s, artificial intelligence (AI) programs required what was then considered a huge amount of computer power. The power was measured in processor time and memory space. On the commercial hardware which was optimized for procedural programming languages, these requirements were exacerbated by the Lisp programming language – a functional programming language Thus, researchers considered a new approach: a computer designed specifically to develop and run large artificial intelligence programs, and tailored to the semantics of the Lisp language.

But, unfortunately Roger Schank and Marvin Minsky warned the business community that enthusiasm for AI had spiraled out of control in the 1980s and that disappointment would certainly follow. Then, in 1987, three years after Minsky and Schank's prediction, the market for LISP machines collapsed. This happened synchronously with the onset of the AI winter and the early beginnings of the microcomputer revolution. Indeed, cheaper PCs soon could run LISP programs even faster than LISP machines, without special purpose hardware.

Although designing and building LISP machines were fleeting activities, they occasioned memorable experiences. It is worth, from this perspective, to bring back to the attention of the scientific community the gains we have acquired on these occasions. Although designing and building LISP machines were fleeting activities, they occasioned memorable experiences. It is worth, from this perspective, to bring back to the attention of the scientific community the gains we have acquired on these occasions. We will further describe the DIALISP system, a LISP machine that was designed, made in the form of a prototype that was later introduced into production. The next section explain why a specialized hardware and architecture is needed for a LISP-based software.

The third section describes the solution on which DIALISP system is based. Final remarks conclude our paper.

2 Why a specialized engine?

The LISP (LISt Processor/Processing) programming language was proposed in 1958 by John McCarthy inspired by Alonzo Church's *lambda calculus*, a computing model theoretically equivalent with Turing Machine (TM). The two models, Turing's and Church's, had been published in the same year, 1936, but MT was adopted as the origin for the abstract model (the Harvard model and the von Neumann model) of a computing machine in the 1940s, and had imposed the procedural programming modality that best suited this abstract model: *processor – communication channel – memory*. John McCarthy's proposal came after the *Dartmouth Summer Research Project on Artificial Intelligence*, a 1956 summer workshop widely considered to be the founding event of artificial intelligence (AI) as a field.

If the procedural high level programming languages were proposed starting from the way computers already worked, the LISP language was designed with an application field in mind: AI. From the beginning, difficulties in running programs written in LISP resulted. The theoretical equivalence between Turing computation and Lambda computation does not remove the essential differences between the two models when the problem of interpretation or execution on the machine that had its origin in MT was raised.

The main differences imposed by the LISP language come from the following two facts:

1. in LISP there is no difference between data and programs; the symbolic structures he works with are *S-expressions*: atoms and lists
2. LISP was the first language to introduce recursive descriptions for which Turing-based systems did not have mechanisms to support efficient running.

Unlike the running of a procedural program which involved the execution or interpretation of a sequence of instructions that a compiler would deliver starting from a program written in a procedural language, an S-expression had to be reduced through an evaluation process which involved the use of a very large stack (LIFO).

The main consequence was the fact that programs written in LISP were run slowly and used a lot of memory. The recursive forms are very compact, but their running consumes time and memory resources for evaluation due to the processing of the chained lists used to represent the S-expressions. Multiple nested recursions consume time and space and Turing-based computers are not optimized to manage stacks extended on spaces that require external memory and evaluation mechanisms that assume reductions of S–expressions. At the same time, arithmetic type operations are in the minority compared to symbolic ones for which standard processors are not properly equipped.

Lisp machines designed, produced and sold in the 1980 years were general-purpose computers designed to efficiently run Lisp programs usually via specific hardware support. They are the first example of a high-level language computer architecture. Several firms designed and produced Lisp machines in the 1980s such as: Symbolics (3600, 3640, XL1200, MacIvory), Lisp Machines Incorporated (LMI Lambda), Texas Instruments (Explorer, MicroExplorer), and Xerox (Interlisp-D workstations).

Following the example of American academics and entrepreneurs, in the early years of the 9th decade in the Functional Electronics Laboratory of the Department of Electronic Devices, Circuits and Appliances (today Devices, Circuits and Electronic Architectures) of the Faculty of Electronics of the Bucharest Polytechnic, a team of teaching staff, researchers and students designed and put into production at the Peripheral Equipment Factory (FEPER) in Bucharest the LISP machine called DIALISP [13]. The block schematic of the DIALISP system is represented in Figure 1, where:

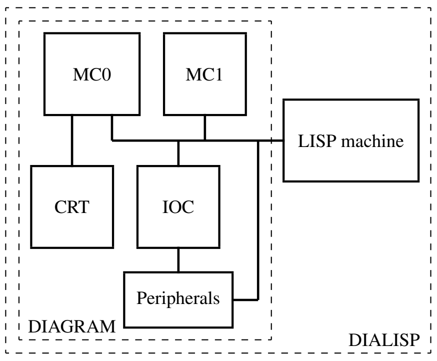

Figure 1: DIALISP, is the DIAGRAM system with a Lisp machine incorporated as an accelerator for running Lisp programs.

MC1 : a microcomputer operating as a central processing unit

MC0 : a microcomputer controlling the alphanumeric and graphic display on a black and white or color CRT monitor

IOC : a microcomputer controlling the system input-output devices

LISP machine : is the LISP hardware interpreter included in the alphanumeric and graphic display DIAGRAM [11, 12].

The impact of the production, in a small series, of this machine was practically zero due to the anti-AI policy of the 1989s in communist-totalitarian Romania.

3 A dual-thread system

The solution proposed for the LISP machine starts from the two main processes identified in the interpretation of a program written in the LISP language: (1) the evaluation by reduction and (2) the management of a large stack. Consequently, we designed a dual-threaded hardware structure with architectures oriented towards the two functions: eval and stack.

In the technological context of the 1980s, the hardware solution was a dynamically microprogrammed machine with the possibility of running two threads interleaved. Interleaved dual-thread technology [1] had been developed in previous years in the Functional Electronics Laboratory during the design of the DIAGRAM alphanumeric and graphic display.

The basic idea in the dual-thread interleaved processor version was to share as much as possible the physical resources available from a mono-thread structure. This allowed the optimization of the use of Intel 3002 chips to be used completely in the two phases of the execution cycle, Φ.

The dual-threaded interleaved processor was designed to avoid data and control dependencies in the pipelined structure of a microprogrammed structure.

3.1 LISP machine general structure

Takeing into account of the following facts:

- the memory access time is approximately two times larger than the execution cycle time
- the memory accesses are frequent
- LISP implementation may imply at least two parallel processes to the main evaluation one (e.g. GC, stack memory, tree structure functions etc)

we were able to define the dual-thread processing engine represented in Figure 2, where:

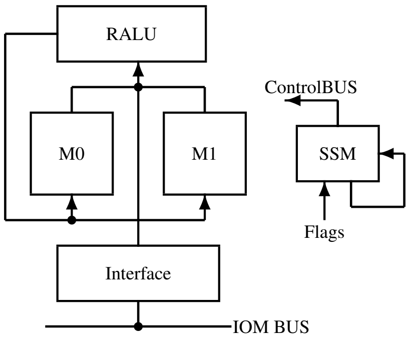

Figure 2: Block diagram of the LISP machine.

- RALU: Registers with Arithmetic-Logic Unit implemented with Intels 3002 whose registers are shared by both processes, P0 and P1
- SSM: Stack State-Machine
- M0 and M1 are memories having 256 KB each and use Intel 2164-25 chips one each associated with the two processes, P0 and P1

The dual-threaded mechanism works as follows:

- when $\Phi = 0$, the process P0 uses RALU and SSM, and the process P1 works with memory M1
- when $\Phi = 1$, the process P1 uses RALU and SSM, and the process P0 works with memory M1

thus ensuring full time use of physical processing resources.

3.2 Stack State-Machine (SSM)

The hardware structure of SSD is represented in Figure 3, where:

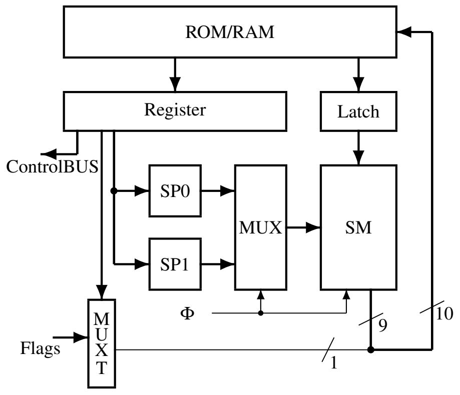

Figure 3: Block diagram of the Stack State-Machine (SSM).

- SM: is a stack memoory used to close the state-machine loop
- SP0 and SP1: are counters used as stack pointers in SM for P0 and P1
- MUX: selects the stack pointer using the signal Φ used to synchronize the two phases of the processing

- MUXT: selects the flag used to determine the transition of SSM.

The way in which the SSM loop is closed allows very efficient control of the recursive processes involved in the interpretation of a program written in LISP.

3.3 Data structure

The word structure used in DIALISP is of 32 bits:

- 1B used as tag;
- 3B used for addresses and for short integers.

The hardware data types are:

```
(S-expression
 (atom
  (constant
   (number
    (short-integer)
    string)
   identifier)
  dotted-pair)
)
```

3.4 Software hierarchy

The three software levels of DIALISP are:

1. microprogramming level
2. assembly level
3. LISP level.

3.4.1 Microprogramming level

Taking into account the position of the DIALISP LISP machine in the range of LISP machines, a memory of microprogram of $1K \times 72$ bit has been chosen; it contains approximately 80 microprogrammed functions. They are:

- initializing functiona
- I/O functions
- pure LISP funstions
- stack functions

The address space is 16 Mwords, the memory being extended on a cartridge disk. The virtual space is logically divided into the following subspaces:

- binary code space (BS), used for assembled functions;
- cell space (CS).

In the version implemented P0 process is driving BS and CS and P1 the stack space.

3.4.2 Assembly level

The assembly language instructions have 8 bit op-code and 0 up to 4 24-bit operands. The operands may be constants, registers (up to five, R0-R4) or jump addresses.

Because the assembler-code provides the control of evaluation choosing a good instruction set improves LISP speed. Thus we have considered two methods:

1. DIALISP is register-like oriented
2. DIALISP is stack-like.

The first option, 1), results in a small set of pure LISP functions which must be microcoded and hence an economical use of control memory. However this solution is slower than 2) regarding evaluation time. Option b) is dealing with an asymmetrical and redundand set of instructions. Most of them are stack-oriented hence there are less pure microprogrammed LISP functions.

3.4.3 LISP level

The table below shows some execution times (clock cycle: $250 ns$) for pure LISP functions:

Function	Time
CAR	4.5 μS
CDR	5 μS
RPLACA	12 μS
RPCLAD	14 μS
CAAR	7 μS
CADR	7.5 μS

Let's note that in the same period - the beginning of the 1980s - Symbolics produced the 3600 microprogrammed machine that operated at a frequency of around 5 MHz. If there were no restrictions due to the embargo, the DIALISP machine could have run at a frequency close to 10 MHz.

4 Concluding remarks

A small series of DIALISP was produced at FEPER. The impact on the Romanian scientific and economic environment of those times was minimal. But a significant sequel of the DIALISP project was the CONNEX project which led to the realization in Silicon Valley of a video processor with 1024 execution cells [10, 9].

References

[1] Gheorghe M. Ştefan (1979) *Circuite LSI pentru procesoare* (LSI Circuits for Processors), PhD Thesis coordinated by Mihai Drăgănescu.

[2] Gheorghe M. Ştefan, Aurel Paun, Andy Birnbaum, Virgil Bistriceanu (1984) DIALISP - A LISP Machine, *The 1984 ACM Symposium on LISP and Functional Programming*, pp. 123–128. At `https://www.academia.edu/64552217/DIALISP_a_LISP_machine`

[3] Gheorghe M. Ştefan, Andy Birnbaum, Virgil Bistriceanu, Aurel Păun (1984) Implementarea hardware a limbajelor pentru inteligenta artificiala, *CNEAC 1984*, IPB Press.

[4] Aurel Păun, Gheorghe M. Ştefan, Andy Birnbaum, Virgil Bistriceanu (1985) DIALISP - experiment de structurare neconventionala a unei masini LISP, *Calculatoarele electronice ale generatiei a cincea*, Ed. Academiei RSR, pp. 160–165.

[5] Gheorghe M. Ştefan, Virgil Bistriceanu, Aurel Păun (1985) Catre un mod natural de implementare a LISP-ului, *Sisteme cu inteligenta artificiala*, Ed. Academiei Romane, Bucuresti, 1991 (paper at *Al doilea simpozion national de inteligenta artificiala*, Sept. 1985). p. 218 - 224.

[6] Gheorghe M. Ştefan (1998) The Connex Memory: A Physical Support for Tree / List Processing, *in The Roumanian Journal of Information Science and Technology*, **1**(1):85–104.

[7] Gheorghe M. Ştefan (2002) Mașina DIALISP - o realizare cu efecte întârziate, *Conferința CALCULATOARE ȘI REȚELE DE CALCULATOARE ÎN ROMÂNIA 1953-1985*, Academia Română.

[8] Gheorghe M. Ştefan, Mihaaela Malița (1996) Chaitin's Toy-Lisp on Connex Memory Machine, *Journal of Universal Computer Science*, **2**(5):410–426.

[9] Gheorghe M. Ştefan (2018) Searching Beyond of the Turing Based Architectures to Surpass the Moore's Law Challenges *Advances in micro- and nanoelectronic technology*, as number 27 of the series Micro- and nanoengineering edited by the Romanian Academy, pp 23-48.

[10] Gheorghe M. Ştefan The Connex Projectat `http://users.dcae.pub.ro/~gstefan/2ndLevel/connex.html`

[11] *Diagram on Wiki*, at `https://ro.wikipedia.org/wiki/Diagram`

[12] *Diagram 2030 on Wiki*, at `https://ro.wikipedia.org/wiki/FEPER`

[13] *Dialisp on Wiki*, at `https://ro.wikipedia.org/wiki/DIALISP`