OVERVIEW OF THE LMI LISP MACHINE SOFTWARE

(The software described in this document executes on both Series III and Lambda machine hardware.)

LISP MACHINE INC.
3916 South Sepulveda Boulevard
Suite 204
Culver City, California 90230
(213) 390-3642

OVERVIEW OF LMI LISP MACHINE SOFTWARE

LMI LISP Machines are integrated hardware and software systems, providing a richer LISP interactive environment than that obtained from either MACLISP or INTERLISP. The LISP Machine provides greater productivity by reducing the time, trouble and cost of programming large systems. As the last eight years of continuing programming development at MIT have shown, large programs can be built, tested and debugged rapidly and with ease. This allows software developers to spend more time on program development rather than building system utilities.

Most computers are built first, with software then tailored to match the machine. The LISP Machine's architecture, however, was implemented around the LISP language. This results in a very fast and flexible system designed to fit the needs of efficient LISP operation. Basic LISP functionality is provided in either system microcode or directly implemented in hardware.

Built on top of this architecture is a very extensive LISP implementation that, while compatible with MACLISP, provides thousands of very rich and useful features. These features naturally extend the power of the basic LISP language in ways that could not be done on conventional architectures. Application programmers on the LISP Machine can, thus, concentrate on the task at hand, choosing from an extensive set of appropriate system functions and language constructs without having to either implement system utilities or restrict their software design.

Additionally, the LISP Machine also provides an extensive array of software development tools. Along with the basic LISP interpreter, the system provides a powerful screen-oriented text editor, compilers that generate either micro or macrocode, as well as a complete user environment with excellent debugging, networking and file system software.

Significant characteristics of the LISP Machine include:

- A powerful dedicated processor with a large virtual memory system (up to 4.3 billion words on the Lambda machine) and disk.
- An interactive, high resolution display console (800 x 1024) including an "AI" keyboard (100 keys providing a superset of ASCII) and "mouse" interactive graphic pointing device.
- Complete datatype error checking via the LISP Machine's tagged architecture.
- A built-in network capability permitting LISP Machines to communicate with other LISP Machines and with host computers.
- An industry standard bus permitting the addition of a broad range of peripheral devices.

The basic system software includes nearly 10,000 compiled functions and over 30,000 symbols representing an initial core image of about 4 million words. Additionally, the user is provided with over 200 micro-coded functions residing in approximately 10K of user-writable control memory.

The functionality represented by this extensive core image is available to every user, yet a majority of the virtual address space remains for user applications.

High Resolution Display

One of the most striking aspects of the LISP Machine concept is the incorporation of a high resolution (800 x 1024) raster display. Refreshing at 60 times a second from a dedicated video memory, the display offers resolution capable of reproducing a standard 8½" x 11" page of text.

Window System

To control this display, the LISP Machine uses a powerful utility called the Window System. This system allows the creation and placement of separate virtual screens of arbitrary size anywhere on the display.

Each window can have a different process associated with it. Windows can be resized or moved, or may partially or totally overlap one another. This allows the display to be as user-tailored as a desk-top; as cluttered and yet accessible as desired. To manage windows, the user also has access to a sophisticated set of menu driven routines allowing him to create or direct attention to any window within the system. A simple mouse command generates the menu and the user need only point to the desired item to produce the result.

A typical use for windows is to maintain both an editing and execution environment side by side on the screen. During the debugging cycle, one is able to bounce back and forth between a ZMACS window, containing program source, and a LISP window containing its execution. Another use is to maintain several editing windows simultaneously on the screen; insuring that specific editing changes are reflected in dependent code.

Users may also use the window system to display their own applications or menus. One may choose from any of the standard features available within the window system or add new ones. This capability is made possible because the window system is built using FLAVORS (a message passing capability integrated into LISP Machine LISP) so that defining the specific behavior required for a window is easily done. Existing functionality available for use by application programmers includes:

- Ability to generate borders and labels.
- Defining objects that blink.
- Generation of basic screen graphics such as boxes and rectangles.
- The ability to represent multiple fonts within a given window.
- Menus.
- Definition of screen objects that are sensitive to being pointed at by the mouse ("Mouse sensitive" items).
- Text and graphics manipulation primitives including character delete, scrolling, "MORE" processing on non-scrolled windows and automatic clipping of text and graphics.

ZMACS Editor

At the heart of the LISP Machine software environment is the ZMACS screen-oriented, a real-time editor. ZMACS' power derives not only from its inherent text and LISP editing capabilities, but also from its integration into the global LISP environment. Thus, it is both an important system utility and an example of the unified yet modular structure that makes the LISP Machine environment so powerful.

Rich Command Set/Sophisticated Help Capability

ZMACS accepts over four hundred commands in addition to allowing users to construct their own. Most commands are implemented in easy to remember mnemonic arrangements, allowing experienced users great speed. For the novice, or for the user exploring new features, the system offers a great deal of on-line help. Commands may be given as text strings (with automatic name completion) or the user can access self-documentation for an unfamiliar keystroke. By using the APROPOS command, one can search through the commands for those containing relevant keywords (if you want to experiment with fonts, then APROPOS FONT will retrieve documentation on all commands containing the word "font").

Integrated Graphics

Unlike its predecessor, EMACS, ZMACS has full access to the LISP Machine's graphics. The mouse is fully integrated into the editing process, allowing rapid positioning of the cursor and the selection of functions with its three buttons. For example, the mouse may be used to designate a section of text to be either deleted or moved to a new location. Within EMACS, this would require a list of cursor commands entered exclusively from the keyboard. ZMACS, however, allows the user to sweep across the screen going directly to the desired point.

User Friendly Environment

In addition to moving the cursor, the mouse allows relative ease in selecting which portion of the file is currently displayed. If the file is larger than the available ZMACS window, scrolling may be achieved by placing the mouse at the bottom (or top) of the window. By rolling the mouse against the window border, one physically moves down the page. If a broader jump is required, one places the mouse against the left border of the window. A heavy black line will appear over a portion of that border signifying the location and relative proportion of text displayed on the screen. Moving to a new location within the file only requires that you move the mouse to a new location along the boarder and press the proper mouse button.

It is important to note that these features, while utilized by ZMACS, are also available to the programmer for specific applications (for example, these features are available within the INSPECTOR). Because of the modularity of the LISP Machine environment, many features available in system software may also be incorporated in user programs; one need only call the necessary library routine to utilize the functionality.

Powerful Software Development Tool

In addition to text processing, ZMACS also excells as a software development tool. Because LISP Machine LISP is dynamically linked, pieces of program can be changed and independently compiled without the need to re-link the entire program. One can edit a function in ZMACS and, with one keystroke, compile and load it into the current LISP environment. Code which calls that function, or is called by it, need not be touched. One just reloads the edited function and initiates execution.

Complementing ZMACS's modular editing capability is its ability to retrieve LISP source code. Functions within the LISP environment

maintain source file information on their property lists. In this way, a ZMACS user can find and edit the source definition of any function within the system. The function may be specified either by typing in its name or by pointing at the function with the mouse. Similarly, one can list all functions in the environment which call a selected function. After the list appears, one simply "mouses" the desired calling function to edit its definition. By providing easy source code retrieval integrated into the editing process, one eliminates the need for cumbersome filing systems to keep track of source code text, facilitating multiple uses for a given piece of software.

ZMACS also knows a great deal about LISP syntax. As you edit a function, levels of parenthesis are aligned and indentation performed automatically by an "incremental pretty-printer". There exists keystrokes for commands such as "forward one s-expression" or "delete s-expression" in analogy to standard character and word manipulation commands. Positioning the cursor next to a right parenthesis causes the corresponding parenthesis to flash, though the system ignores parenthesis that are commented out. ZMACS transforms LISP programming from a language annoying to edit into one that is relatively easy. By providing LISP specific aides, ZMACS encourages readable, elegant and less error-prone code.

Error Handling

Along with a sophisticated editing environment, the LISP Machine architecture also provides exceptional run-time error facilities. Each LISP Machine word has an eight bit tag field, five of which indicate its data type. While data types have normally been limited to numbers and characters, the LISP Machine provides for a whole host of different data objects besides the more traditional atoms, functions, strings and arrays. Datatype information is kept at the architectural levels of the machine to allow data checking and error reporting to be done quickly and efficiently within the system microcode.

When an error is detected, the offending function is suspended and control passed to the error handler. At this point, the user has two options available. The first is the more traditional approach toward debugging, seen in past LISP systems. One has access to a LISP read-eval-print loop within the environment of the stalled routine. The programmer can poke around at the various local variables and data structures that are currently active at that point of the computation. While providing the user with a wealth of information about the current software state, one is forced into a rather myopic view having to query the debugger for each component of the LISP environment. To alleviate this problem, the LISP Machine provides users with the "Window Error Handler".

Graphics-Oriented Error Display

The Window Error Handler is a graphically oriented display of the offending functional environment. The screen is divided into areas representing currently active functional arguments and local variables, a history of the source level expressions that have been evaluated, as well as the current macrocode environment and stack frame. The programmer can either point to various objects with the mouse to expand their contents or evaluate atoms or expressions within the LISP error environment.

In addition, ZMACS is integrated within the Window Error Handler. Users have the option of having compile-time or run-time errors automatically generate a ZMACS window pointing to the offending source code. One need only edit and reload to continue execution.

INSPECTOR

Examining complex data structures within LISP has always been problematical. Earlier LISP systems required that the debugger CAR and CDR his way through complex pointer structures with little or no system help to keep track of where he had been or where he was going. The

INSPECTOR provides a graphically oriented tool, allowing users to examine and modify complex structures carefully and conveniently.

Similar to other software development tools resident within the LISP Machine, the INSPECTOR consists of a window further divided into a number of areas. At the top of the screen is a LISP read-eval-print loop providing an environment for evaluation. Next is a history list providing a full record of all objects visited during the course of the INSPECT session. The final three areas are the most recently examined objects, fully expanded.

To operate the INSPECTOR, one may either type an object into the LISP read-eval-print loop or, more often, point to an existing object on the screen. The object is then placed on the history list and expanded. Based on what is seen the user can then repeat the process.

Incorporated into the INSPECTOR are many of the text handling features resident within ZMACS. Utilization of the mouse to scroll through long structures, or to move proportionally through a structure, operates in a similar manner. Actually, the same code is used which makes the INSPECTOR an excellent example of how easy it is to incrementally build applications software within the LISP Machine environment.

Garbage Collection

The single largest impediment to the acceptance of LISP as a language for production programming has been its garbage collector. Earlier LISP implementations would periodically pause while garbage collection took place. While an annoyance, earlier LISP programmers accepted moderate delays in order to access the LISP programming environment. With the advent of more sophisticated LISP systems, accessing much larger address spaces, these delays have become a good deal longer and quite unacceptable.

The LISP Machine environment solves this problem by offering a parallel garbage collector. By executing concurrently with other LISP Machine

processes, the garbage collector does its job "incrementally", allowing LISP computation to proceed without delay. Users need not fear that a complete garbage collection might occur during a time-critical piece of code. With the addition of a parallel garbage collector, the LISP Machine architecture allows LISP to directly compete with other "real-time" programming languages, yet still retain its enriched programming environment.

File System

The LISP Machine File System allows users easy access to files residing locally on the system disk as well as remote hosts. By defining a uniform file syntax that incorporates a host declaration, users have a general mechanism to access files network-wide. Features such as automatic name completion and wildcard characters are available even though these facilities may not be native to the host.

For files residing within the LISP Machine environment, redundancy and protection are key. Files have version numbers and the system retains a specified number of generations per file. Additionally, the user has access to a two level delete scheme allowing both a delete and non-reversible delete. At the lower levels there is totally redundant information within the file blocks so that the file headers can be totally recreated if need be. Additional file routines include a scavenger to collect displaced file blocks and an incremental/total tape dumping system.

ZMAIL

ZMAIL implements a highly functional mail system, providing service both locally within the mainframe, as well as over the network. Relying heavily on both ZMACS and the Window System, ZMAIL preaches the same philosophy of incremental software development standard throughout the LISP Machine.

Making heavy use of menus and "mouse-sensitive" items, the ZMAIL user controls mail processing by simple mouse movements. One has access to

commands that delete or undelete a message, save messages on a file, reply to an existing message, or send new mail. To move to a new message, one need only point to its entry within the summary window which is permanently resident on the screen. When creating a new message, ZMAIL generates a ZMACS environment, giving the user the full power of the system editor in which to express his thoughts.

LISP Machine LISP Control Structures

STACK GROUPS

The LISP Machine programming environment is a single user/multi-process system. Separate processes are easily defined and controlled by use of the Stack Group feature.

A Stack Group holds all the variable bindings and control stack for each process, defining its state at any point in time. User programs may utilize the Stack Group feature to execute background processes or to generate a prioritized agenda of tasks to complete. Process scheduling may either be done via a simple source level predicate, or one may build sophisticated process control mechanisms with a comprehensive set of system utilities.

Macros

Utilizing all the power of the LISP environment, LISP Machine macros provide an efficient substitution mechanism for LISP forms. Acting similarly to LISP functions, macros produce another LISP form which is evaluated in place of the original macro call. Within the macro body one has access to the full LISP processing environment to produce the desired form. Routines can be as simple or complex as desired, and may even be compiled. Macros are fully recursive, and macro-defining macros are not uncommon. Macro expansion can take place at evaluation time in the case of interpreted functions or at meta-evaluation time during compilation.

Utilization of the "Backquote"

A heavily used feature in macro programming, and a useful facility in general, is the backquote. Selectively turning off evaluation, the backquote allows easy construction of LISP forms having a combination of both constants and functional expressions. Overly complex s-expressions containing multiple levels of LIST and CONS, long the bane of macro programmers, are no longer required. Instead, one uses an elegant notation of quotes and backquotes achieving the desired result with a maximum of clarity.

Packages

A problem in the implementation of large, multi-author, software projects within LISP is the coordination of function and variable names. Everyone wants to use names generic to the application. Past LISP implementations determined which function body to execute based on a unique function name within the entire virtual address space. If the same name exists in more than one place within the address space, one definition would overlay the other.

Utilizing PACKAGES as Separate Name Spaces

To alleviate this problem, LISP Machine LISP dimensions the programming environment into PACKAGES. While still maintaining a common data area, PACKAGES provide the user with separate OBLISTS allowing user control on how ATOMS are resolved. Within multi-author projects, each author implements his code within a separate name space which corresponds to a separate PACKAGE. Programmers requiring services from another package simply make an explicit reference to the desired package and function. In this fashion, large and complex systems can be eaily integrated, accessing common data, without conflicts between low level routines.

PACKAGE Hierarchies

PACKAGES can also be hierarchical. If a function cannot be resolved within the current PACKAGE, a search procedure is initiated which examines the next PACKAGE up the chain. In this way, inheritance schemes can be implemented, defining a PACKAGE's attributes in terms of its predecessors.

Message Passing - FLAVORS

FLAVORS are a way of associating code with data structures and to construct new FLAVOR objects out of currently existing ones. FLAVORS, called like functions with keywords or "MESSAGES" as arguments, search through their associated "METHODS" for one which handles that particular MESSAGE. If no METHOD is found, the MESSAGE is referred to the component FLAVORs that make up the object. Eventually, either a METHOD is found and its code executed, or an unclaimed MESSAGE error is signaled.

Extended SMALLTALK

FLAVORS has its roots in the CLASS system of SMALLTALK but differs in that an object may inherit from more than one SUPERCLASS. In SMALL-TALK, one has the difficulty that pizzas, for example, must be either in CLASS "food-objects" or "physical-objects", whereas in FLAVORS they can be both. Additionally, upon FLAVOR definition, the system control structure eliminates redundant paths to SUPERCLASS items, optimizing MESSAGE processing.

Used Extensively by Window System

Illustrative of the power of FLAVORS is the observation that much of the LISP Machine system software uses it, particularly the Window System. Each window is an instantiation of a FLAVOR, inheriting properties from the component FLAVOR "standard-tv-window". This object defines the generic behavior of any window. Typical METHODS supported

for windows include accepting characters from the keyboard, drawing lines or controlling the mouse. One can also define groups of METHODS as a unit. While not qualifying as a full fledged FLAVOR, these groupings are called "MIX-INs", defining them as additives rather than components.

STREAMS

Another powerful use of the FLAVOR system are STREAMS, which are how standard input and output get passed between different programs and processes. Although conceptually similar to UNIX "PIPES", STREAMS are much more versatile I/O ports, since each is a full FLAVOR instantiation. One can easily tailor an appropriate interface using existing FLAVORs and MIX'INs to exactly suit the particular application, with backup "METHODS" automatically installed to handle any "MESSAGES" for which the optimized operations don't apply. Once a STREAM between two programs or processes is created, information moves quickly and efficiently, easily redirected if required.

SUMMARY

From its inception, the LISP Machine has emphasized an efficient hardware environment providing a host of useful software tools. Software development is greatly enhanced by the ease in which the system aides the programmer; through rich extensions to the LISP language, powerful system utilities, as well as with convenient diagnostic capabilities. By maintaining both a clean user interface, yet providing extremely powerful programming tools, the LISP Machine eases the time and effort to construct and maintain highly sophisticated software applications. As LMI continues to evolve the LISP Machine environment, this basic philosophy will be maintained and extended.