Apple Lisa Computer
Technical Information

Apple Lisa Computer: Lisa Architecture (Apple 1984)

Lisa Computer: 1983 - 1985

The Architecture of the Lisa Personal Computer

Bruce Daniels
Apple Computer
1984

"LisaArchitectureMar84 00.PICT" 268 KB 1999-02-01 dpi: 399h x 418v pix: 2741h x 3825v

The Architecture of the Lisa™ Personal Computer

BRUCE DANIELS

Invited Paper

The Lisa personal computer provides a new and better way of relating to a computer. This paper presents an outline of how such a complex, modern personal computer system is developed. The architecture of both the hardware and the software of the Lisa is examined in detail. Design goals and considerations are also discussed.

BACKGROUND

In 1979 there was a desire within Apple Computer Inc. to develop a new kind of personal computer product. Personal computers like the Apple II made computing affordable enough to meet the needs of a single person. For just a few thousand dollars, one could purchase a real computer to do word processing, accounting, spreadsheet calculations, and other applications. However, there is a critical limitation with such personal computers, as well as with the older minicomputers and mainframe computers. All these computers are difficult to learn to use. They require the understanding of a whole world of new computer concepts and jargon such as programs, data files, file directories, command languages, etc. Because these computers operate in ways that are not even self-consistent, they present a formidable barrier to their use [19].

It has been observed by the Training Department of Apple Computer Inc. that it takes about 20 to 30 h of instruction and practice before a person can learn enough to begin using a traditional computer. This represents a real obstacle to the widespread use of computers to help solve people's problems. Most people are not willing or able to spend the time required to learn to use a traditional computer. Such computers are unfortunately limited to those people who are computer proficient or are willing to become proficient.

The Lisa Charter

The Lisa charter was to build a revolutionary computer that was truly easy to use and thereby to mitigate the limitation of existing computers. A computer which is *revolutionary* may not be compatible with existing products or even with various industry standards and practice. Naturally the Lisa would not be incompatible just for the sake of being different but to be better. Developing a computer which is an order of magnitude easier to use than traditional computers requires major departures.

Manuscript received November 10, 1983; revised December 8, 1983.

The author is with Systems Software, Apple Computer Inc., Cupertino, CA 95014, USA.

™Lisa is a Registered Trademark of Apple Computer Inc.

Design Goals

The first design goal for the Lisa was to be intuitive. This implied departing from traditional computer usage which employs textual communication through a formal command language and with an alien vocabulary. Only by building on what the user already knows and working the way the user expects could the Lisa fulfil its charter.

The second goal was that the Lisa be consistent. If a capability works a certain way in one part of the system, then it must work the same way throughout the system. This means that once the user learns to use a standard feature in one place, then he automatically knows how to use it everywhere. More complex capabilities are built on the principles that the user has already learned. Therefore, complex tasks are possible with only a little more effort.

The third goal was an integrated system conforming to the ways in which people actually work. Day-to-day work consists of a variety of diverse activities that are in progress at the same time. People should not be required to terminate one activity before starting another. Instead people should be allowed to easily switch back and forth from one to another. These activities may be related so that information from one activity should be transferable to another with minimum effort.

The fourth goal was to get enough performance to do the job and do it in a way that minimizes its cost and complexity. High system performance is necessary to satisfy the heavy demands of the unique Lisa software, particularly its graphics capabilities. However, high performance is not inexpensive. It increases the complexity, the speed, and therefore the cost of the processor, the hardware bus, memory, etc.

The fifth goal was to provide an open architecture to facilitate the addition of new software, hardware, and peripherals by not only Apple Computer Inc. but also other developers. The Lisa system was announced with a rather extensive selection of hardware and software. However, the Lisa must be expandable to be able to continue to meet all the needs of its diverse community of users.

The sixth goal was reliability. The Lisa must operate day after day in a correct and accurate fashion. When a rare

"LisaArchitectureMar84 01.PICT" 442 KB 1999-02-01 dpi: 399h x 418v pix: 2700h x 4115v

failure occurs, the problem should be quickly detected, isolated, and fixed. After such a failure when the system is restarted, the user's data must be in a state just as they were before the failure.

The Lisa's final goal was to be pleasing and fit naturally into the everyday work environment. It should not consist of units interconnected with a maze of cables or be a massive and noisy cabinet sitting beside the desk.

The Lisa Hardware

The Lisa hardware [3] consists of a compact, desktop unit that contains the screen, removable power supply, hardware boards, and the floppy disk unit. In addition, a detachable keyboard, mouse, hard disk drives, printers, and other peripherals plug into the main unit as illustrated in Fig. 1. The hardware modules inside the unit are accessed

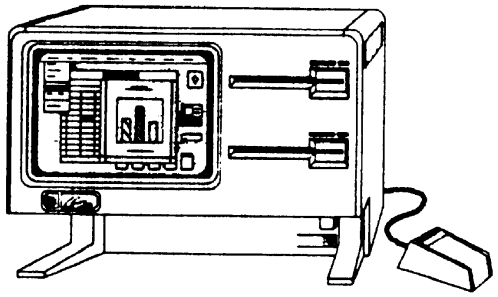


Fig. 1. The Lisa machine with its mouse.

by removing the front and back panels which causes the Lisa to be turned off by the panel safety interlocks. The entire unit can be easily disassembled for service in less than a minute without any tools.

The Lisa hardware consists of four main logic boards: a processor board, an I/O board, and two memory boards. The mother board provides the buses that interconnect these main logic boards. A small video board generates analog signals that actually drive the monitor. Three expansion slots on the mother board accommodate additional logic boards.

The Lisa Processor

Initially, the possibility of using a special Apple Computer Inc. designed processor was investigated. This processor would have provided a special instruction set tailored for the efficient execution of Pascal code. It was to be constructed out of standard 2901 bit-slice microcode circuits. However, designing a new instruction set and its processor for the Lisa could not be justified on economic or engineering grounds. The cost of a standard, off-the-shelf processor drops significantly with mass manufacturing.

Existing 8-bit processors did not offer the levels of high performance that was necessary. The 16-bit processors offered better performance but suffered from a limited architecture. The Motorola MC68000, which had just become available, was chosen as it had a rich architecture with a 32-bit internal data path, multiple addressing modes, and an addressing range of 16 Mbytes [18]. The powerful set of instructions and their fast execution offered high performance. The broad repertoire of instructions could compile high-level languages efficiently. In particular, since the majority of the Lisa software was to be written in Pascal, it was important to minimize the code size.

The Memory Management Unit

The MC68000 processor generates 24-bit logical addresses to access data and instructions. Therefore, it provides a logical address space of 16 Mbytes. In the Lisa this 16-Mbyte logical address space is divided into 128 segments. Each segment consists of up to 128 kbytes in blocks of 512 bytes. The upper 7 bits of a 24-bit logical address is the segment number and the remaining 17 bits is the offset within that segment. The offset consists of 8 upper bits which is the logical block number and 9 bits of the displacement within the block. This can be seen in Fig. 2. To access actual locations in the Lisa hardware, logical addresses are translated into physical addresses by a section of logic on the processor board known as the Memory Management Unit (MMU) [24]. The MMU hardware permits the operating system to control the entire relocation process. The MMU prevents a particular process from accessing areas of memory outside of the portion assigned to it.

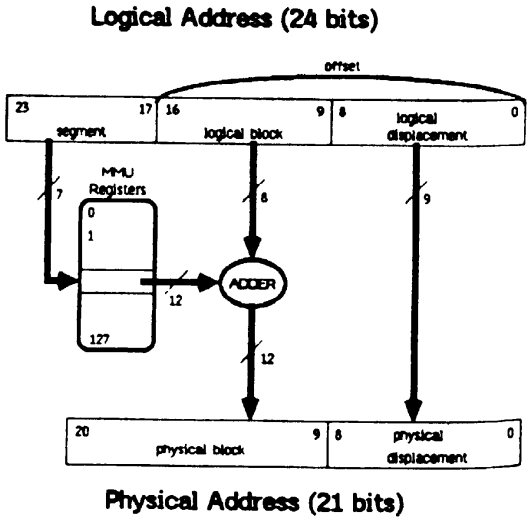


Fig. 2. MMU address transformation.

The Lisa's RAM memory occupies 2 Mbytes of physical address space. This would imply that only 16 segments, each of size 128 kbytes, could be meaningfully used. However, each segment does not necessarily occupy the full 128 kbytes allotted to it in logical address space. Each segment can be mapped into as little as one 512-byte block. Therefore, more than 16 logical segments can map into 2-Mbyte physical memory. Areas larger than 128 kbytes can be accommodated by treating multiple logically contiguous segments as one segment. The translation of a logical

address to a physical address by the MMU is performed on a segment-by-segment basis. Associated with each segment in the MMU is its origin which is the 12-bit block number in the physical address space where the corresponding segment begins. The logical block number from bits 9–16 of the logical address is added to the segment's origin value to produce the physical block number to be accessed. The nine displacement bits from the logical address translate directly into the physical displacement. This translation process is illustrated in Fig. 2.

The MMU also performs access checks to verify that the requested access is allowed. The MMU checks each access to ensure that it does not exceed the bounds of the specified segment. In addition, an attempt to access a segment which is not mapped or an attempt to write into a read-only segment will generate an access violation. These access violations trap to the operating system for handling.

Within the MMU there are four distinct sets of translation registers, each representing a different mapping from logical addresses into physical addresses. Each set is called a *context*. Only one context is current at any given time. Context 0 is reserved for use by the operating system. Contexts 1, 2, and 3 are used for client processes supported by the operating system. However, more than three processes are possible by using the three MMU contexts as a "cache" of the three most recently used processes. By simply switching contexts, rapid switching among processes and the operating system is accomplished. The Lisa automatically selects context 0 whenever an access is made in Supervisor mode. Thus a TRAP instruction can be used to generate a call from a user process to the operating system. Normally the context is changed while executing in the operating system in Supervisor mode. Execution in the new context begins when user mode is entered.

The Lisa Display

Traditional personal computers employ a text-only display that restricts the output to a limited set of characters at fixed positions on the screen. The Lisa uses a high-resolution bit-mapped display which allows virtually unlimited graphics and the use of multiple sizes and styles of text fonts, including proportionally spaced fonts. Considerable time and effort went into the selection of the dimensions, resolution, composition, and refresh rate of the Lisa's display.

When the refresh rate of the screen is too slow, it produces a maddening flicker of the screen. A high refresh rate demands a higher memory access bandwidth which requires either fast and expensive RAM's or it significantly degrades system performance. On the other hand, a slower refresh rate can be employed if one uses a slow phosphor in the CRT. However, a slow phosphor causes objects removed from the display to slowly fade away on the screen and moving objects to smear. It was found after much experimentation that a responsive yet solid display required a moderately fast phosphor and a noninterlaced refresh rate of 60 Hz [22], [23].

Another important property of a display is its dimensions. When the development of the Lisa began, some research computers, for example see [28], provided a *full page* bit-mapped display that allowed a complete 8 1/2- by 11-in sheet of paper, or up to 66 lines by 102 characters to be shown at once. However, the CRT tube for such a large display is rather expensive. In addition, a full page display places great demands on the memory access bandwidth. For example, a full page display requires a bit map approximately 768 pixels wide by 1024 high, which corresponds to 96 kbytes of memory. With a screen refresh rate of 60 Hz, a memory bandwidth of 5.76 Mbytes per second would be required for just the video alone!

Although a full page display is convenient, the Lisa's half page display provides most of the benefits but at a considerably reduced cost and complexity. A *half page* display still shows the full 8 1/2-in width of a page but only about 5 or 6 in of the height. Such a smaller display reduces the video bandwidth requirements by half. Another way that the Lisa reduces the video bandwidth is to employ different resolutions for the horizontal and vertical dimensions. The most demanding use of the video resolution is for the display of text. However, the display of sharp and accurate text requires a higher resolution in the horizontal dimension than it does in the vertical dimension. Therefore, on the Lisa display there are 3 pixels in the horizontal dimension for every 2 pixels in the vertical. These different resolutions do complicate, slightly, the design of the Lisa graphics software. For example, the software must remember that a graphical object 150 by 100 appears as a square, however 100 by 100 is a rectangle. Because of its half page size and different resolutions, the bit map for the display (720 pixels wide by 364 pixels high) requires only 32 kbytes corresponding to a memory bandwidth of 1.92 Mbytes per second, which is only one third that of a full page display.

There is one other subtle aspect of the Lisa display that is worth mentioning. Ordinary computers display white characters on a black background. The bit map display hardware for such a computer is implemented with 1 bit per pixel, a black pixel as a Zero and white as a One. To display a character, the computer software sets the appropriate bits to binary One or white. In contrast, the Lisa display shows black characters on a white background to mimic the way text actually appears on a real printed page. Ergonomic studies [23] have shown that when looking back and forth from the screen to a real piece of paper, it is actually easier on the eyes if the screen is the same black on white as the paper. Display of black characters on a white background can be accomplished in software by resetting the appropriate screen bits to binary Zero (black). However, it is much faster and easier with the MC68000 and most other processors to set selected bits to One, using the OR instruction, than it is to reset selected bits to Zero, which requires a NOT-OR-NOT instruction sequence. As a result, the Lisa bit map display can operate faster by representing a black pixel as a One and white as a Zero.

The Lisa Operating System

To support the kind of advanced, integrated software that was planned for the Lisa, a powerful multitasking operating system is required. This requirement eliminated all the popular personal computers operating systems such as CP/M, MS-DOS, the UCSD System, and Apple DOS. Their primary design constraint is that they work in just a small portion of the restricted memory space of existing PC's.

The UNIX operating system [21], [29], seems to be more suitable. It does provide multitasking, good memory

Architecture of the Lisa Personal Computer [3 of 12]

management, and a powerful device and file system. However UNIX is a relatively large operating system with several features such as a multiuser timesharing capability, user accounting, and protection which would be wasted in a *personal* computer such as the Lisa. Since the Lisa would be used by people who were not computer experts, the system must be very robust. However, the UNIX file system is fragile and unreliable [10], [14]. If the power is interrupted or a system crash occurs the UNIX file system can easily be damaged. Unless a systems programmer is present to repair the damage, a user can easily lose all his data. In addition, UNIX does not provide the general inter-task communication facility that the Lisa requires. UNIX memory management does not offer sophisticated sharing of code and data between tasks. Finally, UNIX and all other operating systems do not provide the support for graphics, multiple windows, the mouse, integration, etc., which are essential to the Lisa. Such capabilities cannot be built on top of an operating system but must be built in to work correctly and efficiently. An attempt to modify UNIX to overcome all of these deficiencies would have taken longer than designing a new operating system with all of the needed capabilities.

The Lisa Operating System (the Lisa OS) performs four main functions: file management, process management, memory management, and event and exception handling. The file system provides for a uniform naming mechanism for objects (peripheral devices, disk volumes, files, etc.) as in Multics [8] and UNIX [21]. Before a device or disk volume can be accessed, it must be mounted by using the MOUNT system call. Mounting an object logically connects it into the name space of the system. In addition, mounting a disk volume makes the files on the volume accessible. The file system provides device-independent I/O to objects which means that I/O is performed the same way, whether the ultimate destination or source is a disk, a printer, or something else. The file system treats I/O as an uninterpreted stream of bytes. Special device-control functions are available to perform any device specific functions needed, such as setting the baud rate of a serial device.

Some operations apply only to disk objects. New disk files are created, removed, and renamed by changing entries in the disk's catalog. In addition to the data in a disk file, the file itself has certain system attributes, such as its size and creation date. Programs can define their own attributes in a special label associated with each file. System calls are available to access these file attributes. When writing to a disk file, space is allocated as needed. Since this space need not be contiguous, such automatic allocation could result in a severely fragmented file. The resulting performance degradation can be avoided by using system calls to pre-allocate contiguous space for a file. This also ensures that space on the disk will not be exhausted while writing.

To reduce the impact of a system crash, the file system maintains distributed, redundant information about the files on disk storage [17], [20]. Duplicate copies of critical information are stored in different forms and in different places on the media. For example, the information in the central disk catalog about a file is also stored in a special disk block at the head of that file. Also each block on the disk specifies the part of the file to which it belongs. Since all the files and blocks are able to identify and describe themselves, there are several ways to recover lost information. A utility called the scavenger is able to reconstruct damaged catalogs from the redundant information stored about each file.

The Lisa does not have a color display. The hardware necessary for really good color is not available. For a low resolution color display, a CRT tube from a standard color television is suitable. However, a color CRT tube suitable for the Lisa must have sufficiently high resolution to display sharp text so that one can do word processing on the machine all day long without getting fatigued. Such a high-resolution color CRT tube and the associated video electronics would add thousands of dollars to the price of the Lisa. In addition, color on the video screen is rather futile unless one can also produce color on the printed page. High-resolution color printers and office copier machines are even more expensive and difficult to obtain. Many of the Lisa applications, such as word processing and spreadsheet calculation, do not have any important use for color. There are certainly some applications, such as graphs and drawings, where color would be convenient. But even with these applications, the Lisa's use of multiple shades, stripes, cross-hatchs, and patterns eliminates the necessity to have color in order to distinguish and emphasize various graphical objects.

The Lisa Hardware Bus

The hardware bus of the Lisa provides access to the physical memory. This physical memory consists of three separate address spaces. The main memory space contains up to 2 Mbytes of RAM memory for storage of programs and data. Parity checking of this RAM memory is supported to ensure reliable operation. The I/O space provides access to status and control registers of various peripheral devices, both built-in and external. A special I/O space provides access to the bootstrap ROM and special system registers.

Both the MC68000 processor and the video display contend with each other for access to memory. To simplify this contention problem, the Lisa has adopted the same technique used in the Apple-II bus, [2], [31]. Video access to memory alternates regularly with the processor access to memory. This interleaved memory access guarantees the video display the regular, dependable access to memory that it needs for a flicker-free display. To give the video sufficient memory bandwidth, the principal bus timing is an 800-ns cycle consisting of one 400-ns video access to memory followed by one 400-ns processor access. The timing considerations of the Lisa bus lead to a 5-MHz 68000 clock with a clock period of 200 ns. Since either a read or write cycle of the MC68000 processor requires four clock periods, then such a read or write will require 800 ns. This is the same as the Lisa 800-ns bus cycle. While the video access is being performed, the MC68000 is preparing its 24-bit logical address and presenting it to the Lisa's MMU for mapping into a physical memory address. If one of the Lisa's three expansion slots requires Direct Memory Access (DMA), then its request will take priority over and delay the processor's memory access.

The architecture of the Lisa hardware bus allows for a simple and low-cost implementation while still providing some powerful capabilities such as DMA and memory management. This is the primary reason why the Lisa did not adopt some other bus standard such as MultiBus. In

addition, the Lisa's hardware boards must be specially designed and shaped anyway to fit into the Lisa's compact cabinet.

THE LISA SOFTWARE

Never before has software been such a large part of the development of a personal computer or been so crucial to its total system architecture. The Lisa Operating System [4] provides virtual memory, multiple processes, and a reliable, device independent file system. The Lisa user interface defines how the software appears and interacts with the user. The software library provides a rich set of primitives for graphics, windows, printing, etc. The Lisa Desktop Manager functions as a system executive in performing filing operations and running application programs. There are seven specific application programs developed by Apple Computer Inc.: LisaCalc—spreadsheet, LisaGraph—business graphs, LisaWrite—word processing, LisaList—personal database, LisaDraw—graphics editing, LisaProject—project management, and LisaTerminal—data communications, see [30] and [11]. The Workshop software development system includes compilers, editors, linkers, etc., and is available for the programming languages: Pascal, BASIC, COBOL, and C. QuickPort and the Toolkit are software packages that aid the software developer in producing software applications for the Lisa. In this exposition of the Lisa Software architecture we concentrate on the Operating System, the Lisa user interface, the software library, and Lisa Desktop Manager. The additional software components are not mentioned here, not because they are uninteresting or unimportant, but because they are not central to the exposition of the fundamental Lisa software architecture.

A Lisa OS process is an instance of an executing program, its stack, and associated data. When the OS is booted, it creates a "shell" process which can then create other processes for the user. Since every process is created by another process, the resultant structure is a tree of processes. Each newly created process has the same standard system capabilities which can then be changed by system calls. A process can suspend, activate, kill, or otherwise control any other process. When a process terminates, all of its descendant processes are also terminated. The CPU is multiplexed among the runnable processes by using a priority based, nonpreemptive scheduling algorithm. This nonpreemptive scheduling policy guarantees correct access to shared resources, such as the bit-mapped display, by interactive processes without the performance penalty of having to explicitly lock and unlock these resources for each access. The memory accesses of an executing process are restricted to its own logical address space. Processes can share their code and data, but each has its own stack. Processes can communicate with other processes by using shared files, shared data segments, and events.

The Lisa OS memory manager provides a segmented virtual memory capability. It is concerned with memory segments and their location in physical memory or on the disk. Memory segments are of two basic types: code segments and data segments. Each process has a data segment that the OS automatically creates for it to use as a stack. This stack segment is automatically enlarged by the OS as more space is needed by the process. Up to sixteen additional data segments can be acquired by the process for uses such as heaps and interprocess communication. These data segments can be either private, accessed only by the creating process, or shared, accessible by any process that opens those segments. The maximum size of a shared data segment is 128 kbytes. The OS allows a private data segment to be as large as the physical size of the system (2 Mbytes) by employing multiple sequential MMU registers.

Code segments allow a program to be constructed as multiple, independently swappable parts. The division of a program into these named code segments is dictated by the programmer through commands to the Compiler and Linker. The MMU allows up to 106 code segments. A given program consists of both intrinsic and regular code segments. Intrinsic code segments, such as the units in the Lisa software Library, are shared by all processes. Regular code segments are shared by just those processes executing the same program. The maximum size of a code segment is 128 kbytes.

Code segments are automatically swapped into physical memory as they are needed. Since they are write protected, code segments do not have to be swapped out. Although instructions in the MC68000 processor are not generally restartable, we have empirically determined that the four instructions that access code segments, JMP, JSR, RTS, and RTE, are restartable. When one of these four instructions attempts to reference a code segment that is not currently present, it causes a bus error which traps to the OS. The OS memory manager can then load the missing segment and restart the instruction without the client process having to be aware of what has happened. This mechanism allows the Lisa to support full swapping of code without the expense of a second processor to handle swapping. Because the instructions that reference data are not all restartable, the system does not do automatic swapping of data segments. The memory manager must swap in all data segments needed by a process before that process is allowed to execute. However, the OS gives programs the ability to *unbind* data segments that are not needed in the memory while a particular part of the program is executing. Since an executing process requires only its current code segment and its bound data segments to be in physical memory, the total amount of logical memory used by a single process may actually exceed the physical RAM of the Lisa.

When the memory becomes full, the system uses a clock algorithm [7] to determine which segments to swap out or to replace. There is a Segment Descriptor Block (SDB) in the memory manager for each code or data segment currently in use. These SDB's are chained together in a circular list which constitutes the "clock face" of the clock algorithm. The memory manager has a pointer to a current SDB which constitutes the "hands of the clock." A segment will be in one of three states: on disk (not in memory), an *overlay candidate* (in memory but not mapped), or not an overlay candidate (in memory and mapped). If the memory manager needs to swap in a segment and there is insufficient free physical memory available, then the clock hand is advanced to the next SDB and this segment is examined. If the segment is not an overlay candidate, then it is made an overlay candidate and the clock hand is advanced again. If the segment is an overlay candidate then it is swapped out (written to the disk if it is a data segment) and its space is added to the free pool. The clock hand continues to ad-

vance around the circular list until enough free space is accumulated to satisfy the current request. Since a code segment which is an overlay candidate is not mapped, any attempt to reference it will generate a bus error just as if it were not in memory. The memory manager handles such a bus error by changing the code segment to not-an-overlay candidate, remapping the segment, and restarting the code reference. A segment is changed to an overlay candidate whenever the clock hand passes by; but, if referenced frequently, is changed back before the hand has gone around again. However, a segment which is used infrequently will remain an overlay candidate and will be swapped out. Therefore, the memory manager uses its bus error mechanism both to handle a reference to a missing code segment and also to determine which segments in memory have not been used recently.

An OS exception is an unexpected condition in the execution of a process (an interrupt). System exceptions are generated by various sorts of errors such as divide by zero, range check out of bounds, illegal instruction, and illegal address. Default exception handlers are supplied that terminate the process. However, a process can supply its own exception handlers if it wants to recover from the error. The exception handler is passed information about the interrupted environment: register contents, condition flags, and program state which it can examine and modify. User exceptions can be declared and exception handlers supplied to process them. A program can then use these new exception handling mechanisms.

An event is a message from one process to another sent through an event channel. The event is a fixed-size data block consisting of a header and some text. The header contains control information, the identity of the sending process, and the type of event. The header is written by the system, not the sender, and is readable by the receiving process. The event text is written by the sender; its meaning is defined by the sending and receiving processes. The name of an event channel is cataloged by the file system and can be accessed by any process. An event channel with no name is used by a process to receive system-generated events pertaining to its descendant processes. A process that expects a message can wait for an event on a channel. If the receiving process is not ready to receive the event, then the event channel queues the event. In addition, an event channel can be made to generate a user exception whenever a message arrives.

The Lisa User Interface

Traditional user interfaces are textual with input coming from characters typed at a keyboard and output being printed text. Many of these user interfaces do not even make use of the random access and editing properties of CRT terminals. Such interfaces work equally well with hardcopy terminals. The command language form of user interface has been in existence since the very start of computing itself. It is based on the same sort of formal syntactic structure as the various programming languages. In fact, some of the programming techniques that are used to parse programming languages can be used to parse command languages. A command language user interface is a very precise, rigid form of interaction and the language itself seems very artificial to a new user. If the correct command is ERA then it does no good to type REMOVE, DELETE, KILL, or even ERASE. The precise order of arguments to a command is critical. Even the details of punctuation may be important, such as a comma, semicolon, slash, or whatever. A menu-based user interface frees the user from remembering the exact names and spelling of commands, [1], [25]. However, because a menu displays a set of choices and then forces the user to pick one, it imposes a rigid structure of its own. Multiple menus are usually required, since the number of choices that can be displayed in a single menu is limited. This produces a hierarchical menu structure with menu choices from the root menu serving to bring up additional submenus. Choices from such submenus may bring up further subsubmenus and so on. To return from the lower level menus back up to the root menu requires some sort of QUIT menu item. To invoke a particular command, the user is required to navigate around this maze of menus to find the proper menu in which the command appears.

The user interface ideas of the Smalltalk system, as developed at Xerox PARC, [13], [32], [27], provide alternatives to the traditional user interfaces, and form the conceptual basis for the Lisa User Interface. Smalltalk is a heavily graphics oriented user interface presenting an image of multiple, overlapping pieces of paper on a grey electronic desktop. Each piece of paper, or window, can be a separate activity which can proceed independently of the others. Smalltalk makes use of the "mouse," [6], [12]. For example, a window is indicated not by typing its name but by simply pointing at it with the mouse. The most important use of the mouse is as a single, uniform method for selecting data objects. Using the mouse to operate a scrolling mechanism brings the desired data into view on the window. One then selects the data by pointing at them, irrespective of whether the data are textual, numeric, graphical, spreadsheet, or of any other kind. If you can see the data then you can select them with the mouse. Another use of the mouse is to invoke commands from menus. Two of the three buttons on the mouse cause menus to "pop-up" on the screen. The mouse is used to select the desired command. These Smalltalk concepts were refined, augmented, and made more efficient and practical to form the user interface of the Lisa (see Figs. 3 and 4).

Early user tests demonstrated that a single button mouse was much easier for new users to learn. With a multibutton mouse the user would stop and look at the mouse and try to remember which button did what. With a single button mouse, the Smalltalk concept of using the extra buttons to pop-up menus was not possible. However, these pop-up menus were too limited in number for the sophisticated applications that were desired for the Lisa. The Lisa solution was to place a special menu bar along the top of the screen. This menu bar contains the titles of up to twelve menus that are simultaneously available. Clicking on one of these menu titles causes the corresponding menu to "pull-down" from the menu bar for selection of the desired command. Since each menu can contain twenty or more entries, there are literally hundreds of commands that are available. For the sophisticated user, frequent commands can be invoked directly from the keyboard.

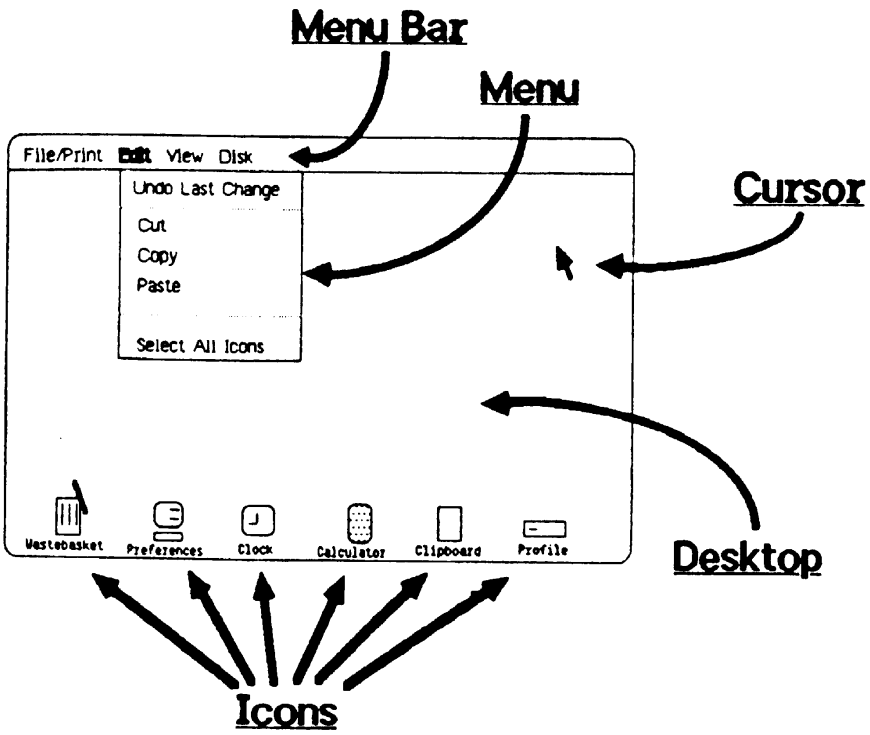


Fig. 3. The Lisa user interface.

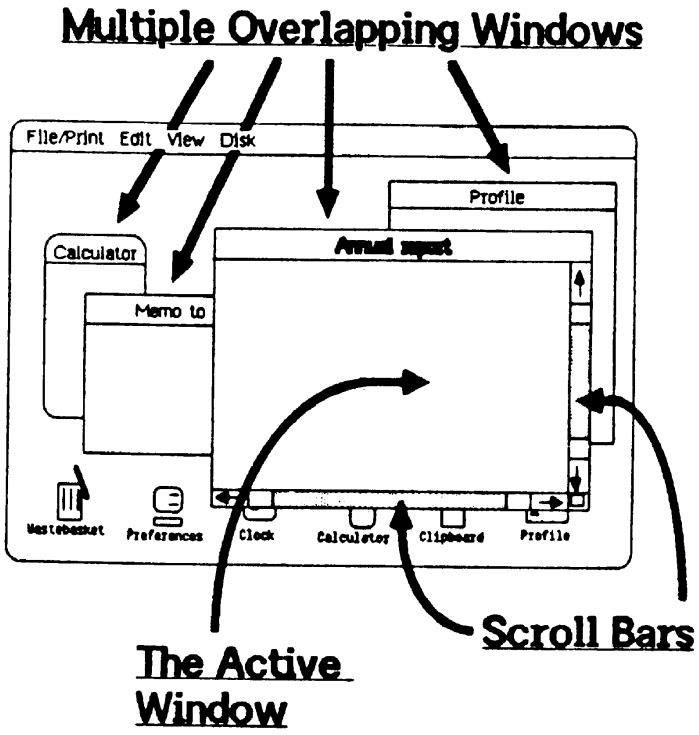


Fig. 4. More of the Lisa interface.

Architecture of the Lisa Personal Computer [7 of 12]

"LisaArchitectureMar84 07.PICT" 173 KB 1999-02-01 dpi: 399h x 418v pix: 2516h x 3986v

The Lisa user interface employs its mouse and graphics to provide a more intuitive and consistent way for people to interact with a computer. The Lisa display shows graphic images of familiar objects on an electronic desktop. The user controls the machine by simply manipulating these images, called *icons*, [26] rather than by typing command sequences. Using the mouse, one selects the desired object by pointing at it, and then chooses the desired command to operate on the selected object. All the conventional "filing" operations are performed by pointing with the mouse. For example, to delete a document one points at the document icon with the mouse and then drags it over on top of the Wastebasket icon. Just as something thrown into a real wastebasket can be retrieved, the last object placed in the Lisa Wastebasket can be retrieved. To create a new document one points at a stationery pad icon and then clicks the mouse button twice to tear off the new document. To copy an existing document one duplicates the document icon, then points at the location where the copy should be placed. A document can be renamed by simply pointing at the icon and typing its new name. Deletion of a document, or file, is accomplished on the Lisa in the same fundamental way as on UNIX and other systems (remove its entry from the catalog and return its disk blocks to the free pool). The difference is that the Lisa provides a more intuitive and visual means to express this and other operations.

An icon can be selected and then opened into a window on the desktop in order to get access to its contents. The icon for a ProFile™ Winchester hard disk drive or floppy diskette can be opened to show, as a window containing icons, what is on the disk. A folder icon, which can be used to group related objects on a disk, can be opened to show its contents. By placing folders inside of folders, which in turn are inside other folders, and so on., the user can arrange information exactly as is possible on a conventional hierarchical file system. A Preferences icon can be opened to allow the user to adjust system parameters such as screen brightness or to configure peripherals or disks. Document icons indicate visually not only that the object is a document, but also the type of document: spreadsheet, business chart, drawing, list, text, etc. Opening a document icon shows the information so the user can work on it. The user does not have to run programs, called *tools* in the Lisa. Opening a document automatically causes the appropriate application program to be run which then will interpret, display, and allow the user to manipulate its data. Opening a text document "ALPHA" on the Lisa accomplishes the same fundamental operation as a command like "Edit ALPHA" on a conventional system (i.e., run the Edit program on the ALPHA file), but, again, in a more intuitive and visual way that shields the user from unimportant details.

The Lisa Software Library

An integral part of the Lisa system is a vast library of software units (see Fig. 5). These units establish protocols to

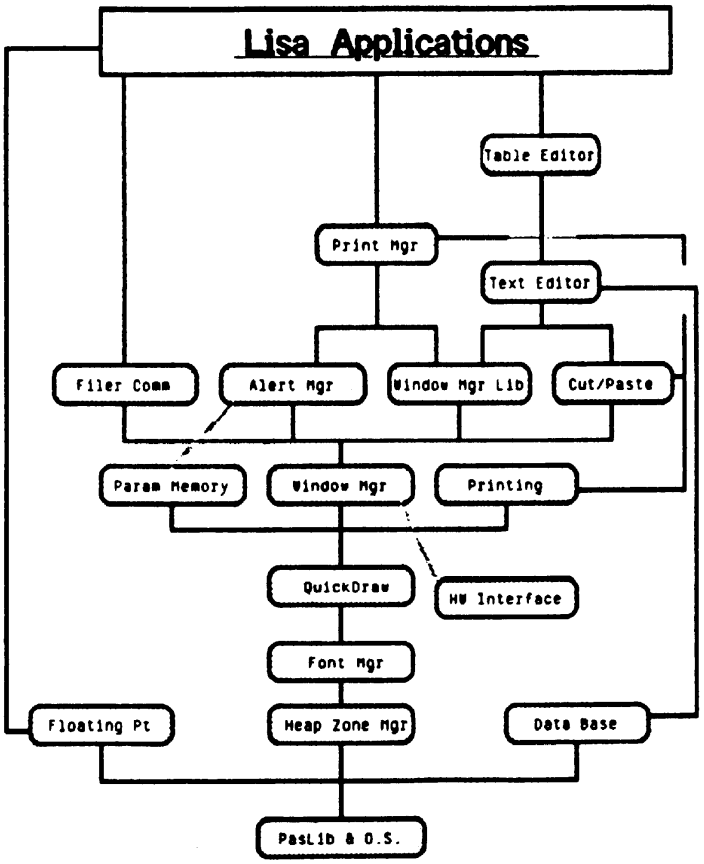


Fig. 5. Structure of the Lisa software library.

Architecture of the Lisa Personal Computer [8 of 12]

"LisaArchitectureMar84 08.PICT" 364 KB 1999-02-01 dpi: 399h x 418v pix: 2708h x 4083v

be followed across all the applications to implement cooperatively a consistent Lisa user interface. The library consists of over half a megabyte of code with 4000 callable routines.

The capabilities for the management of the graphics screen are provided through the close cooperation of various library units, the Lisa OS, and the actual application programs themselves. The **QuickDraw** unit is the Lisa's high-speed bitmap graphics unit. It is significantly more powerful than the BITBLT capability of previous bitmap graphics routines [15], [16]. QuickDraw automatically clips all of its output to nonrectangular *regions* to support a multiple, overlapping window environment. QuickDraw, in conjunction with the **Font Manager** unit, draws text to any starting pixel from a variety of fonts which are automatically swapped into memory as needed. It supports proportional widths, multiple drawing modes (OR, XOR, and BIC), and display styles such as bold, italic, underlined, and shadowed. Both fonts and QuickDraw bitmaps can be automatically stretched or shrunk to fit into a destination, giving multiple sizes of these objects. QuickDraw supports the primitive graphical shapes: lines, rectangles, ovals, arcs, and rounded corner rectangles. These shapes can all be drawn with specified pen width, height, and texture pattern and with a variety of drawing modes (OR, XOR, BIC, etc.). The same QuickDraw *region* mechanism that is used for clipping can also be used to define, manipulate, and display new shapes. A QuickDraw *picture* object represents an arbitrary piece of graphics through a compact transcript of the drawing calls. These pictures are used as the universal medium of exchange of graphical information between applications. While providing all these unique, powerful capabilities, QuickDraw is still able to offer high performance such as displaying 4000 characters per second, 800 lines per second, and 160 large solid rectangles per second.

The **Window Manager** unit is responsible for keeping track of the number of open windows, the location of each window on the screen, the size of each window, and which windows are in front of or behind the other windows. The Window Manager knows the process which "owns" the window and is responsible for its actions. For each window that is covered by other windows and therefore partially obscured, the Window Manager calculates the region of the window that is currently visible (see Fig. 6). QuickDraw automatically restricts or clips any output to that window to

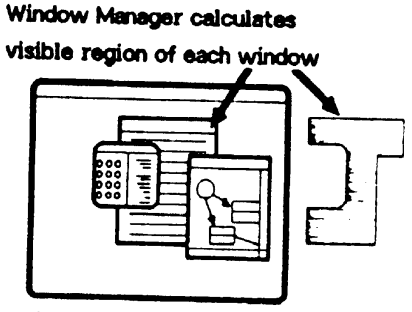


Fig. 6. Visible region of a window.

the portion of the display that is visible. An application process can safely draw into its window at any time without having to know about the windows in front of it. When windows are moved, resized, or otherwise changed, the Window Manager makes sure that the portions of windows that have been uncovered are redisplayed. This is accomplished by the Window Manager keeping a QuickDraw picture for each window which is drawn, but can also be accomplished by asking the application to redisplay the missing content. As a result, the application designer does not need to be aware of where on the screen the window is located or what portion of it is currently visible.

Support for the mouse and the keyboard are provided by two Library units in conjunction with the Lisa OS. The **Hardware Interface** unit responds to interrupts from input devices such as the mouse or keyboard. It queues information about these input events so that they are not lost even if the system is busy. Since the keyboard and mouse are shared by all the windows for their input, the Window Manager defines one window as the *active* window with which the user is currently interacting. The Window Manager classifies input events and routes them to the process of the active window for handling. The processes of other windows which request input are blocked until they become active. Additional shared resources such as the menu bar and the alert box for messages are owned by the active window. To switch context and make a different window active, the user simply points at it with the mouse. The Window Manager sends a deactivate event to the process of the currently active window and then an activate event to the process of the new window. The Window Manager keeps the priority of the active window process higher than those of other windows or any background processes so that this interactive process is guaranteed the best performance from the OS.

The guiding philosophy behind the Lisa's advanced printing technology is: "What you see on the screen is just what you will get on the printed page." In the past this had been accomplished by restricting output to very high resolution, and very expensive, laser printers. The Lisa **Print Manager** represents the first time that this philosophy has been accomplished with a much cheaper device, such as the less than $700 Apple Dot Matrix Printer, and with even a non-raster device, such as the Apple Daisy Wheel Printer. The Print Manager matches fonts to the specified print device and uses QuickDraw's ability to automatically stretch and shrink objects in order to print to the resolution of its output device. It is capable of printing good quality graphics as well as the usual text on the Daisy Wheel Printer. The Print Manager supports background printing so the user can continue working while printing is in progress. This is accomplished by recording on the disk a QuickDraw picture of each page to be printed.

The **Menu Manager** unit is used to display and select commands from pull-down menus. Another unit allows different portions of a document to be viewed using scrolling. The **Alert Manager** unit displays messages informing the user of errors. There are other software units to enter and edit simple lines of text, to perform floating-point computations, to access database information, and for other specialized applications. Just a few of the one hundred available units have been described. The Lisa software library provides an unusually rich and complete set of capabilities and, therefore, establishes a firm foundation for the applications.

The Lisa Desktop Manager and Applications

The Lisa Desktop Manager serves the same basic functions as the Shell or command interpreter in conventional

systems. It provides a mechanism for the user to create and manage documents or files (copy, move, rename, delete, etc.), to run tools or programs, and, in general, to control the system. The desktop image, implemented through the Window Manager, is treated as a special window that is always open to the full width and height of the screen, is always behind any other open windows, and has a grey background pattern rather than the usual white. The Desktop Manager displays the icons that are out on top of the desktop and the icons in any open windows associated with disk, diskette, or folder icons. The Desktop Manager recognizes the user's manipulations of any of these icons and responds interactively with the appropriate visual feedback. The Desktop Manager also performs any filing operations such as file deletion or copying implied by such manipulations, invoking the necessary OS file system calls, and then displaying the resultant visual image. When a user "opens" any document icons, the Desktop Manager first determines the exact type of document which the user desires to open. Associated with each document type is a Lisa tool, or application program. The Desktop Manager then creates an OS process running the desired tool. Next the Desktop Manager calls the Window Manager to establish a window with the same size and position on the screen as the document had when it was last opened. The Desktop Manager sends a DocOpen event to the new process passing both the window to be used and the identity of the document to be opened. When the application process receives the event, it opens the document files and displays the document in its window. Finally, the Desktop Manager makes the new window be the active window so that the user can proceed to manipulate and edit its contents.

The Clipboard is a desktop icon that serves as the medium of information exchange, or integration, in the Lisa. When the user selects some information and performs a cut or copy operation, data are placed on the Clipboard that allows this information to be put into another location with the paste operation. The architecture of the Clipboard supports the transfer of information within a single document, between documents of the same type, and also between documents of different types. In fact, the architecture supports transfer between documents of not only the existing applications but also of future applications. The Clipboard is implemented as a common shared data segment that is accessed by all application processes. Data structures that define the information to be moved are placed into this data segment. No single data structure will suffice for all information transfers. Such a single data structure would tend to lose information even for transfer between documents of the same type. For example, spreadsheet data must include not only the visible cell value but also the formula to compute the value, the column width, the numeric format, etc. The Clipboard data structures are self-describing so that it is possible to distinguish spreadsheet data from any other kind of data. This description allows new kinds of data, for example voice data, to be defined and added in the future. Applications can accept only those kinds of data that they can recognize and handle and can reject unknown kinds of data. However, the Clipboard architecture also allows two application programs, which do not recognize and accept each others data, to transfer information between themselves. To accomplish this, information is placed into the Clipboard in more than one data format. These multiple data formats are arranged in a sequence of increasing generality. The least general format is "application specific data," such as the spreadsheet data that have already been described. If an application does not understand this application specific data, then it can attempt to use a so-called "universal text" form of the same information. This consists of just text characters and formatting commands such as tab and carriage return. For example, a word processor would not be expected to recognize and accept the application specific form of spreadsheet data, but would accept the universal text form which would be cell values represented as text and separated by tabs and carriage returns. The most general form of information is called "universal graphics" and consists of a QuickDraw picture which can be used to generate an image of the information. For example, a word processor could accept this form of information in order to paste a picture of a bar chart, a project schedule, a drawing, or anything else into the middle of a written report.

Summary

The hardware and software of the Lisa establishes a new standard of innovative architecture among personal computers. The contrast with the primitive architectures of the personal computers available even a few years ago is immense. In fact, the hardware and software architecture of the Lisa can be reasonably compared with those of the newer minicomputers. For the first time, a personal computer like the Lisa with the performance, capacity, and architecture offered by a super minicomputer can now be placed on an individual's desk for less than $3500. Software applications which were just not possible or affordable with previous personal computers or time-shared minicomputers have now become possible. By building upon the architecture of the Lisa, these programs are able to more than fulfill the original charter of the Lisa project to build a computer that is ten times easier to learn than traditional computers.

References

[1] R. B. Allen, "Cognitive factors in human interaction with computers," in *Directions in Human/Computer Interaction*, A. Badre and B. Shneiderman, Eds. Norwood, NJ: Ablex Publishing Corp., 1982, ch. 1, pp. 1–26.

[2] Apple Computer Inc., *Apple II Reference Manual*, Cupertino, CA., 1979.

[3] Apple Computer Inc., *Lisa Hardware Reference Guide*, Cupertino, CA, 1983.

[4] Apple Computer Inc., "Operating system reference manual for the Lisa," in *Lisa Pascal Manual Set*, Cupertino, CA, 1983.

[5] Apple Computer Inc., "QuickDraw reference manual," in *Lisa Pascal Manual Set*, Cupertino, CA, 1983.

[6] S. Card, W. English, and B. Burr, "Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT," *Ergonomics*, vol. 21, no. 8, pp. 601–613, 1978.

[7] R. Carr and J. Hennessy, "WSCLOCK—A simple and effective algorithm for virtual memory management," in *Proc. 8th Symp. on Operating Systems Principles* (ACM), vol. 15, no. 5, pp. 87–95, Dec. 1981.

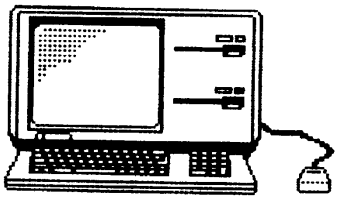
[8] F. J. Corbato and V. A. Vyssotsky, "An introduction and overview of the Multics system," in *Proc. AFIPS Fall Joint Computer Conf.*, pp. 667–668, Oct. 1971.

[9] B. Daniels, "Lisa's alternative operating system," *Comput. Des.*, vol. 22, no. 9, pp. 159–166, Aug. 1983.

Architecture of the Lisa Personal Computer [10 of 12]

"LisaArchitectureMar84 10.PICT" 586 KB 1999-02-01 dpi: 399h x 418v pix: 2692h x 3971v

[10] H. M. Deitel, "Case study: UNIX," in *An Introduction to Operating Systems*. Reading MA: Addison-Wesley, 1983, ch. 18, pp. 479–504.
[11] J. L. Ehardt, "Apple's Lisa: A personal office system," *The Seybold Rep. Office Syst.*, vol. 6, no. 2, pp. 1–26, Jan. 24, 1983.
[12] W. English, D. Engelhart, and M. L. Berman, "Display-selection techniques for text manipulation," *IEEE Trans. Human Factors Electron.*, vol. HFE-8, no. 1, pp. 21–31, 1967.
[13] A. Goldberg and D. Robson, *Smalltalk-80 The Language and its Implementation*. Reading MA: Addison-Wesley, 1983.
[14] R. B. Greenberg, "The UNIX operating system and the XENIX standard operating environment," *BYTE*, vol. 6, no. 6, pp. 248–264, June 1981.
[15] D. H. Ingalls, "The Smalltalk-76 programming system: Design and implementation," in *Proc. Principles of Programming Languages Symp.*, pp. 9–16, Jan. 1978.
[16] ——, "The Smalltalk graphics kernal," *BYTE*, vol. 6, no. 8, pp. 168–194, Aug. 1981.
[17] B. W. Lampson and R. F. Sproull, "An open operating system for a single user machine," in *Proc. 7th Symp. on Operating Systems Principles*, pp. 98–105, 1979.
[18] Motorola Inc., *MC68000 16-Bit Microprocessor User's Manual*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[19] D. A. Norman, "The trouble with UNIX," *Datamation*, vol. 27, no. 11, pp. 139–153, Nov. 1981.
[20] D. D. Redell *et al.*, "Pilot: An operating system for a personal computer," *Commun. ACM*, vol. 23, no. 2, pp. 81–91, Feb. 1980.
[21] D. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *Bell Syst. Tech. J.*, vol. 57, no. 6, pt. 2, pp. 1905–1930, July–Aug. 1978.
[22] B. E. Rogowitz, "The human visual system: A guide for the display technologist," *Proc. Soc. Informat. Display*, vol. 24, no. 3, pp. 235–252, July 1983.
[23] B. A. Rupp, "Visual display standards: A review of issues," *Proc. Soc. Informat. Display*, vol. 22, no. 1, pp. 63–72, Jan. 1981.
[24] S. Schmitt, "Virtual memory for microcomputers," *BYTE*, vol. 8, no. 4, pp. 210–238, Apr. 1983.
[25] H. Simpson, "A human-factors style guide for program design," *BYTE*, vol. 7, no. 4, pp. 108–132, Apr. 1982.
[26] D. C. Smith, C. Irby, R. Kimball, and E. Harslem, "The Star user interface," in *AFIPS Proc. Nat. Comput. Conf.*, vol. 51, pp. 515–528, 1982.
[27] L. Tesler, "The Smalltalk environment," *BYTE*, vol. 6, no. 8, pp. 90–147, Aug. 1981.
[28] C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, "Alto: A personal computer," in *Computer Structures: Principles and Examples*, D. Siewiorek, C. G. Bell, and A. Newell, Eds. New York: McGraw-Hill, 1982.
[29] K. Thompson, "UNIX implementation," *Bell Syst. Tech. J.*, vol 57, no. 6, pt. 2, pp. 1931–1946, July–Aug. 1978.
[30] G. Williams, "The Lisa computer system," *BYTE*, vol. 8, no. 2, pp. 33–50, Feb. 1983.
[31] S. Wozniak, "System description: The Apple II," *BYTE*, vol. 2, no. 5, May 1977.
[32] Xerox Learning Research Group, "The Smalltalk-80 system," *BYTE*, vol. 6, no. 8, pp. 36–48, Aug. 1981.



Apple Lisa Personal Computer
1983 - 1985

Architecture of the Lisa Personal Computer [11 of 12]

"LisaArchitectureMar84 11.PICT" 257 KB 1999-02-01 dpi: 399h x 418v pix: 2685h x 4018v

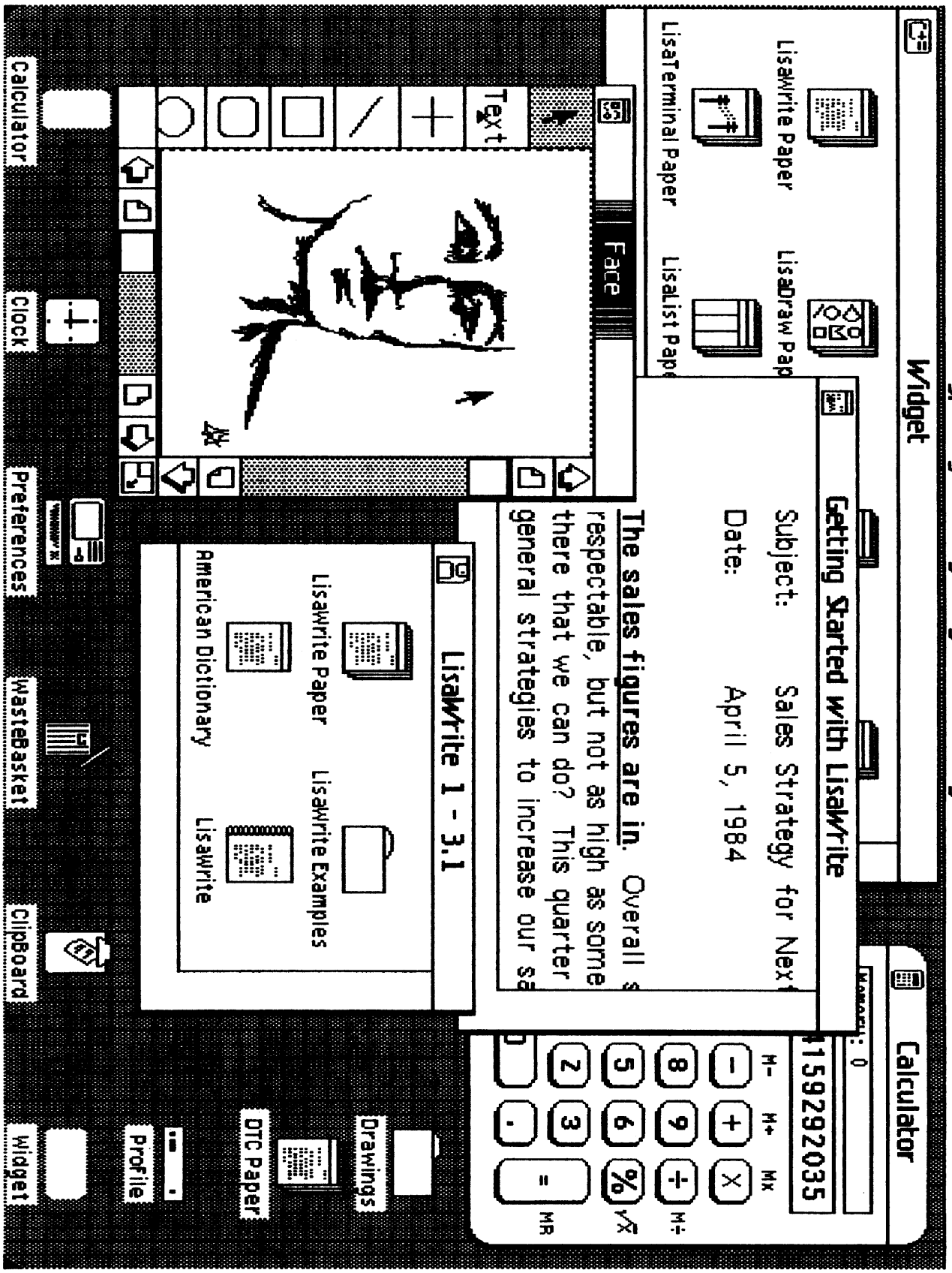
Bruce Daniels

Mr. Daniels received the B.S. and M.S. degrees in computer science from the Massachusetts Institute of Technology, Cambridge. He worked in the Ph.D. program in the areas of programming languages and compilers.

He has been a key member of the management team of the Lisa project at Apple Computer Inc., Cupertino, CA from its inception four years ago. He served as the Lisa software manager with overall responsibility for all the Lisa software, including the user interface, application programs, application intrinsics, operating systems, and development systems. He has also served as a leader of the Software Advanced Development Group for Lisa where he was responsible for software architecture, integration, performance, and reliability. Previously, he had served as the first software project leader and system architect for the Apple III. He was the co-designer of the Apple III Sophisticated Operating System (SOS). Before joining Apple Computer, he was at Hewlett-Packard in the General Systems Division where he was responsible for development systems.

Architecture of the Lisa Personal Computer [12 of 12]

LISA OFFICE SYSTEM 3 (7/7)



"LisaArchitectureMar84 13.PICT" 606 KB 1999-02-01 dpi: 399h x 418v pix: 561h x 741v

"LisaArchitectureMar84 14.PICT" 403 KB 1999-02-01 dpi: 399h x 418v pix: 569h x 731v