Apple Lisa

Jon Freeland - Joslin Hendricks - Keith Powell - Sharad Tiwari

Table of Contents

Introduction:

Upon its introduction in 1983, the Apple Lisa was the first personal computer to make use of a Graphical User Interface (GUI). The Lisa (sometimes LISA) was initially rumoured to have been named after Steve Jobs' daughter or the daughter of one of the developers, but officially Apple said it meant Logical Integrated Software Architecture. Lisa was based off of the Smalltalk system, developed by Xerox at Xerox PARC (Palo Alto Research Center). When development began on the Lisa in 1979, the charter for this project specified that this computer should be easy to use and it allows the user to spend more time being productive than figuring out how to use the computer.[1][4]

Design Goals:

The Lisa had several design goals that would make it stand out from other personal computers at the time. First, the Lisa should be intuitive so that the user can easily figure out how to perform the tasks he is trying to accomplish. Second, it should be consistent so that once the user learns how to perform a task, he can perform that task the same way everywhere. Third, the system should work the way people do and not the other way around. Users should be able to work on multiple tasks in multiple applications without having to close one application to start another one. Fourth, the system should have enough performance to be able to accomplish tasks in a reasonable amount of time. The features that the Lisa had were very resource intensive, especially with the graphics processing that was required. Fifth, the system was supposed to provide an open software and hardware architecture to allow for expansion to meet user requirements not only with solutions developed by Apple Computer, but by third party vendors as well. The sixth goal was reliability. Users should be able to use the system and not have to be concerned about the system crashing unexpectedly and their files being lost. The seventh goal for the Lisa was to be aesthetically pleasing and be a good fit in a corporate environment. The Lisa, unlike some of the computers popular at the time, was contained in one chassis that included the monitor, floppy drives, and all of the circuit and processor boards. The only items to plug in were a keyboard and mouse and optionally a five or ten MB external hard disk.[1]

Intended Application:

The Lisa was intended for corporate desktop use as well as consumer use, but its high price (US $9995.00) made it impractical for most consumers. The Lisa was able to do word processing, spreadsheets, graphic design, databases, project management, connect to other text-based systems, and other applications - not new concepts at the time, but the way that the Lisa accomplished this and presented it to the user in an intuitive, easy to learn manner made it radically different from other systems available at that time. The concepts behind the Lisa were not necessarily new - many had been around for several years, but were not implemented in commercially available systems.[1][4]

Architectural Classification:

The Motorola 68000 is a CISC microprocessor, the first member of a successful family of microprocessors, which were all mostly software compatible. It is a Von Neumann SISD processor. The 68000 microprocessor is packaged in a 64-pin dual in-line processor. It is a 16-bit microprocessor that communicates with the outside world via a 16-bit bi-directional data bus. In the standard version of 68000, the clock frequency may be set to anything as long as it doesn't exceed 8 MHz or drop below 2 MHz. Originally, the Motorola 68000 was designed for use in household products When the Motorola 68000 was introduced, 16-bit busses were really the most practical size.

Architectural Specifications:

Codename:	Lisa
CPU:	MC68000
CPU speed:	5 Mhz
FPU:	None
motherboard RAM:	512 k
maximum RAM:	2MB (via 3rd party upgrade)
number of sockets:	2 -- lisa cards
minimum speed:	n/a
ROM:	16k of diagnostic and bootstrap code present
L1 cache:	n/a
L2 cache:	n/a
data path:	16 bit
bus speed:	5 Mhz
slots:	3 Proprietary
SCSI:	none
Serial Ports:	2 RS-232
Parallel Ports:	1 (dropped in Lisa 2/MacXL)
Floppy:	2 internal 871k 5.25" (400k Sony 3.5" in Lisa2/MacXL)
HD:	5 MB external (10MB in some configurations of Lisa 2/MacXL)
CD-ROM:	none
Monitor:	12" 720 x 360 built-in (B/W)
Sound Input/Output:	Continuously Variable Slope Demodulator (CVSD)
Ethernet:	none
Gestalt ID:	2
power:	150 Watts
Weight:	48 lbs. Dimensions: 15.2" H x 18.7" W x 13.8" D
Min System Software:	LisaOS
Max System Software:	LisaOS/MacWorks
introduced:	January 1983
terminated:	August 1986[4]

Implementations (Versions) of Lisa:

Between January 1983 and April 1985, there were three versions of Lisa. The first version, Lisa, came with two proprietary Twiggy(5.25", 860K) floppy drives, 512K or 1MB of RAM, a 5MB ProFile external hard drive. The 12" monochrome monitor was built in to the Lisa chassis. When Apple release the Macintosh in 1984, they also released a cheaper version of the

Lisa called the Lisa 2. The Lisa 2 had one Sony 400k 3.5" floppy, and configurations were available with 2MB of RAM and a 10MB external hard drive. In January 1985, the Lisa 2/10 was renamed as the Macintosh XL. Apple also released a software emulator called Macworks that would allow the Lisa to run the Mac OS. The Lisa/Macintosh XL was discontinued in April 1985.[4]

Hardware:

- CPU - The Motorola 68000 processor was first engineered in September 1979. Prototypes were released in 1982 and the processor was commercially available in 1983. While it was supposedly intended for use in household products, it actually ended up being used in several well known computers and gaming consoles. The 68000 is a 16 bit processor because its external data path is 16 bits, but its internal data paths and registers are 32 bits and its address bus is 24 bits. It was packaged in a 64 pin chip - 40 of those pins were used for data and addressing.[13]

  The 68000 uses sixteen 32 bit registers which are broken off into eight data and address registers. For the address registers, one is reserved for the stack pointer while the rest were limited to move, add/subtract, and load effective address.

  ### CPU Resources

  External data accesses by the 16 data lines (bus) and the 23 address lines (bus) give the 68000 the ability to address over 16 million bytes of memory. Since devices are in constant need of the processor, there will often be bus contention. To reduce the bus contention, the 68000 has special control signals discussed in "Bus Arbitration," that activate the external circuitry that allows smooth interfacing with it's outside devices.[12]

  The 68000 has eight general purpose registers and address registers, each are 32 bits in length. It is great to have an enormous space to work with but some operations do not require the entire use of the register. One problem with big registers is that you could be wasting space, but the 68000 is able to work with partitions on each registers. The registers can be broken down into 16, 8, or 1 bit at a time for instruction operations. Each partitioned size, except for the 1 bit, can be addressed by using a special extension along with the instruction mnemonic.[12] The last address register, A7, was used for the standard stack pointer.[13]

  Coprocessors and other functional units where implemented for the later version of the 68000 series, but the 68000 was capable of mapping to another processor and run in parallel given the proper implementation procedures.[20]

  ### Internal Bus

  The 68000 is said to have a single-bus architecture since the I/O and memory units share the same bus with 68000. The number of signal lines devoted to addresses, data, and control signals essentially determines the capability of the single bus system. The 68000 actually had 64 signal lines to accommodate addresses and data, addresses, and control signals in additions to several other functions.[16]

  ### Categories of Functionality:

  - Miscellaneous: Pins Vcc and Gnd fall under this category. These two pins connect power to the chip. Vcc is the power supply and Gnd stands for ground.

  - System Control: System control is comprised of CLK, RESET, and HALT pins. The CLK pin or clock does all the internal timing for the 68000. The clock input must never stop, fall below, or exceed the pulse variance length. The bi-directional RESET pin, once activated with an active-low level signal, loads the supervisor stack pointer A7 from location 0 then makes the processor execute a sequence of actions under the reset operation which is under exception handling. Also, RESET can reset all of the external devices without resetting the chip itself. The bi-directional pin HALT has three functions: it can make the 68000 stop processing at the end of the current bus cycle; in conjunction with the BERR pin, it can be used to repeat a failed bus cycle; and it can be used as an output to interpret what happened when it can not recover from an operation.

  - Address Bus: The 23-bit Address Bus is comprised of pins A01 to A23 makes the majority of the pins on the 68000. Every pin is unidirectional and can allow up to 223 16-bit words to be addressed uniquely.

  - Data Bus: The 16-bit bi-directional Data Bus, D00 to D15, can act as input during a read cycle and output during a write cycle.

  - Asynchronous Bus Control: Asynchronous data transfer deals with transfers that refer to the address strobe in processing the addresses and data. The AS (active-low address strobe) pin determines the validity of the content in the address bus. The R/W (read/write) pin chooses each cycle to be either read or write. The Upper and Lower data strobes, UDS and LDS, control the data bus to either the entire 16 bits be addressed or either the top or lower half will be addressed. The DTACK pin acts like a check in station for the processor (stands for Data Transfer ACKnowledge). When DTACK is asserted, the processor will complete the access and start on the next cycle. When DTACK is not present, then the processor will produce wait states until DTACK is present or an error occurs. Finally the BERR pin, bus error control, informs the 68000 that something is wrong with the bus cycle allowing for the processor to recover.[11]

- **Synchronous Bus Control:** With synchronous data transfer, each address is validated and processed in sync with the clock pin, making it easier for interfacing. The VPA, or valid peripheral address, is activated when a device sends a active-low valid address input to the pin requesting for a synchronous bus cycle by use of the VMA and E pins. The VMA, or valid memory address, sends an active low signal back to the device telling it that it has a valid address sent from the VPA. The E pin (enable) is the time that is used for communication between the device and the processor. The E clock cycle is equal to ten 68000 clock cycles.

- **Interrupt Control Interface:** These three pins (IPL0, IPL1, and IPL2) are used by external devices to request service. Each bit are used as an interrupt mask bit to determine the level of interrupt. An interrupt level goes from 0 to 7 in a three bit number where 0 is the lowest level and 7 is the highest level. High level interrupts will always override the low level interrupts.

- **Bus Arbitration**: This groups main focus is to allow smooth communication with other processors. Even though the 68000 is considered the "bus master," other processors may use the system bus with its permission. The bus request pin (BR) is asserted when another devices wants to use system bus. This signal have to be responded to by the 68000 for this signal does not share the properties of an interrupt. If the 68000 accepts the request, then the bus grant pin (BG) is asserted and sent back to the corresponding device to let it know that it can proceed. Once the bus grant acknowledge pin (BGACK) is asserted, then the system bus is under the control of the device. Nothing else, including the CPU itself, can access the system bus as long as the BGACK is asserted.[20]

- **Function Code:** Three pins (FC0-FC2) serve as a three bit mask to determine the type of cycle being executed. The chart display the following cycle type:

  Function Code Output

FC2	FC1	FC0	Processor Cycle Type
0	0	0	Undefined, Reserved
0	0	1	User Data
0	1	0	User Program
0	1	1	Undefined, Reserved
1	0	0	Undefined, Reserved
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	CPU Space (interrupt acknowledge)

  The CPU can run in two states: user or supervisor. The user state is more associated with programs executing under the OS while the supervisor state is more associated with the OS itself. The supervisor state has the highest privilege and there are a few instructions that can only be executed in this state.[20]

- **Addressing Modes and Formats** - The 68000 also supports fourteen different addressing modes, which ranks it high among the most powerful microprocessors. These addressing modes are derived from six basic types, which are absolute, immediate, register direct, register indirect, program counter relative, and implied.[11] Addressing modes are used to calculate the actual address, or effective address, of the operand.[16] Within immediate addressing, the operand simply follows the instruction. Similarly, in absolute mode, the address, which is in short 16-bit form or long 32-bit form, the address of the operand follows the instruction. When register direct addressing is used, it is no longer necessary to calculate the effective address because the address of the operand is specified directly.[16] Indirect addressing uses an address register to hold the address of the operand. Program counter (PC) relative addressing is slightly more complex. Within PC relative, the effective address is calculated by adding a displacement to whatever value is in the program counter. The displacement can either be a positive or negative number. There are a number of variations with PC relative addressing including PC relative with displacement and indexing.[16]

- **Data Types** - The 68000 supports operations on five main data types including bits, binary coded decimal (BCD) digits (4 bits), bytes, words, and long words.[19] The long words are required to store high words in memory first. Bytes, words, and long words may be formatted together to form multiple precision numbers. That is to say for example, a routine could be written for a 32-bit operation using two 16 bit words.[21] Data types may be considered as signed or unsigned. The range for unsigned data types is 2^{n-1}, where n is the length of the data in bits. For signed datatypes, the range is -2^{n}- 1 to $+2^{n-1}$ -1.[21]

- **Instruction Set** - The Motorola 68000 processor used in the Apple Lisa utilized several groups of variable-length instructions within the instruction set.[19] These instructions were broken down into eight groups consisting of arithmetic, logical, data transfer, branches, shift and rotate, bit manipulation, BCD operations, program control, and system control.[11] Previous architectures often contained several instructions that basically performed the same operations, however the 68000 was designed to eliminate the need for unnecessary mnemonics. Instead, the instruction was coded to perform operations on either 8-, 16-, or 32-bit registers.[11] Most of the instructions were dyadic meaning that the operation had a source and a destination, and the destination was changed.[19] Each instruction consisted of an opcode that determined what operation to perform, a designation of the length of the operand(s), and a specification of the location of the operand(s) involved. Instructions were classified by type or by the number of operands. The latter determined whether the instruction was single-address or double-address.[16]

- Instruction Execution and Timing - During normal execution of an operation, the 68000 has two basic tasks which are fetching instructions from memory and executing the fetched instructions.[15] The 68000 employs a 2-stage pipeline which simply means that the next instruction can be fetched during execution.[17]

  When performing an instruction fetch, the microprocessor uses the address bus to send the address of the instruction, then issues a memory read signal (R/W = 1). These instructions for this particular family of processors are "always stored in multiples of 2, 3, 6, 8, or 10 bytes in length." The PC (program counter) register, which is 24 bits wide, is used to monitor the program and the location of the next instruction. The instruction is decoded and goes into execution when it has been relocated from memory into an internal instruction register.[15]

  There are three main steps that occur during the execution phase: (1) data transfer, (2) arithmetic and logic, and (3) decisions, with data transfers covering a great deal of the operation. Data transfers occur between the microprocessor and the memory, between the microprocessor and I/O, and between internal registers of the microprocessor.[15]

  Cycle time, t_c, is what the 68000 uses to calculate execution time of instructions and other operations. Since the speed of the 68000 is 8 MHz, the cycle time has a range of 500 ns maximum to a minimum of 125 ns. These numbers are based on the time to fetch an instruction, compute the effective addresses, and fetch or store the operand.[16]

- Exception Processing - Exceptions for the 68000 are divided into two categories: those caused by an instruction and those caused by an external event. This particular processor supports 255 different exceptions. Exceptions caused by instructions are called traps. Traps occur when "exceptional conditions" are caused by the program and detected by the CPU. Some examples of the traps are trace, divide-by-zero, and privilege violation. Hardware error exceptions caused by external events are called interrupts. Motorola refers to this hardware error as a bus error. A few examples of these include bus error, address error, and RESET.[11][16]

  The 68000 had seven "strictly prioritized" interrupt levels, which means that higher-numbered interrupts took priority over lower-numbered interrupts. This was accomplished in the status register by using a privileged instruction to allow one interrupt to set the minimum interrupt level, which in turn blocked the lower priority interrupts. The minimum interrupt level was then stored back into the status register. Level 7 was the only level considered non-maskable.[13]

  The 68000 also provided an exception table, or interrupt vector addresses. Since the addresses were fixed at 0 through 1023, this allowed for 256 32-bit vectors. The first and second vectors were used as the starting stack address and the starting code address, respectively. A range of errors were reported in vectors 3-15 and vector 24 began the "real interrupts."[13]

- Procedure Calling - Subroutine instructions are used in programs to improve the overall performance. Since they only handle one job at a time, it allows for easier development of modular programs. Subroutines in the 68000 can either be called using the BSR (branch to subroutine) instruction or the JSR (jump to subroutine) instruction.[15] The RTS (return from subroutine) instruction is used to return from a subroutine. While the JSR instruction operates on any address, the BSR instruction only operates on a 16-bit signed displacement.[18]

- Control Unit - The Motorola MC68000 had a two-level, microprogrammed control unit design. The control unit instructions were 17 bits. Each microinstruction could have a 10 bit microinstruction jump address or a 9 bit nanoinstruction address. The nanoinstructions were stored as 68-bit words and were used to identify microinstructions active in any given clock cycle. These nanoinstructions, along with other decoding logic, were used to drive the 196 control signals in the 68000.[6] There are 544 17- bit words in the microengine and 336 68-bit words in the nanocode engine which makes up 32,096 bits of ROM. There are fewer nanocode instructions than microcode instructions in the 68000; this is to allow common microcode instructions to map to the same nanocode instruction.[7]

- Memory Management System - Before we discuss the LISA's memory management unit, we have to give a quick overview on how the processor addresses memory. The 68000 supports both synchronous and asynchronous bus transfer. The address and the data bus are driven by tristate outputs, which allow the buses to be controlled by other devices during DMA operations in multiprocessors. The pins that control the flow of data between the buses are the R/W, AS, UDS, LDS, and DTACK pins. Running at 8 MHz, the 68000 can use low-cost dynamic memory (DRAM) without any wait states. Because cache memory is both expensive and complex, the cache memory generally was not found in most low-cost microprocessors. Cache memory was not implemented until the 68020 processor and other later versions in the 68000 family; this kept the 68000 at a low price.[20]

  In the Lisa, memory can be broken into three parts: main system memory, I/O memory, and special I/O memory. Main system memory, or RAM, in this system can be upgraded to 1 MB of RAM (2 MB in the Lisa 2/Mac XL). Initially, the base main memory starts off at 512K. The I/O memory is reserved for the I/O devices. Finally, the special memory is used for booting up the computer and diagnostics.[5]

The Memory Management Unit (MMU) helps the operating system relocate objects in memory. The MMU method of addressing is called segmentation. As mentioned before, the 68000 has 23 address lines. Lines A17-A23 are used to select a specific segment out of 128 total segments. The rest of the bits serve as offset for segments. Each segment has two registers assigned to it: SORG which describes the origin and SLIM which describes the size. The board then has a segment limit register that correlates to the MMU the size of the segment, is the segment valid, and control which address space is accessed. In order for the MMU to be setup, the SORG and the SLIM registers of all the segments must be initialized by the OS.[16]

During an access, a memory error can be detected as either a soft error or a hard error. A soft error is an error that happens during a board access that can be corrected. A hard error is when either a parity error or an unrecoverable error happens during memory access. Once either of these errors is signaled, the board latches the address of the error, interrupts the 68000, and puts the segment and page number in the status register. Therefore, a program can log soft errors in the board and keep track of hard failures in main memory on a page by page basis for the MMU to map out the bad pages.[5]

- I/O System - The Motorola 68000 CPU is able to perform memory mapped I/O in either byte or word lengths. It requires a memory address decoder, connected with the appropriate circuitry. The Motorola 68000 uses memory mapped I/O, and device registers are assigned unique addresses within the memory address space. I/O data and control registers are used like memory locations.[14]

  I/O peripherals are connected with the Motorola 68000 through address decoders. These I/O address decoders are used to map hardware peripheral registers to specific memory locations. When the processor reads or write in one of these I/O locations, the peripheral device will take an action. "For a memory-mapped output location, the memory address decoder provides a clock pulse to a latch capable of storing the output data. A memory-mapped input location will use the memory address decoder to enable an octal buffer, placing data on the CPU's data bus when active. A 16-bit I/O design requires two octal buffers, one latch/buffer for each half of the data bus. The same address decoder may be used with UDS (Upper Data Strobe) and LDS (Lower Data Strobe) controlling the latch/buffer that gets activated. The LDS and UDS signals are used in the 68000 to indicate that an 8-bit bank of memory is used for the memory or I/O transfer." LDS is active when least significant 8 bits are accessed, and UDS is active when the most significant 8 bits are accessed.[13]

  Data Transfer Operations:

  Data transfer between devices requires the following signals:

  - Address bus A1 through highest numbered address line
  - Data bus D0 through D15
  - Control signals

  The read, write, read-modify-write, and CPU space cycles are described following. The indivisible read-modify-write cycle implements interlocked multiprocessor communications.[15]

  Read Cycle:

  The processor receives either one or two bytes of data from the memory or from a peripheral device during a read cycle. When the instruction indicates a byte operation, the processor uses the internal A0 bit to determine which byte to read and issues the appropriate data strobe.[11] If the instruction indicates a word or long-word operation, the MC68000 processor reads both upper and lower bytes concurrently by maintaining both upper and lower data strobes. When A0 is equal to zero, the upper data strobe is issued; when A0 equals one, the lower data strobe is issued. When the data is received,the processor internally positions the byte accordingly.[11]

  Write Cycle:

  The processor sends bytes of data to the memory or peripheral device during a write cycle. When the instruction indicates a byte operation, the processor uses the internal A0 bit to determine which byte to write and issues the appropriate data strobe. If the instruction specifies a word operation, the processor issues both UDS and LDS and writes both bytes. When the A0 bit is equal to zero, UDS is used; when the A0 bit is equal to one, LDS is used.[11]

  External Bus Structure:

  The Motorola 68000 microprocessor has a 16-bit external data bus and a 24-bit address bus which can address 16MB of external memory, while implementing 32 bit registers internally. Only 23 of these address lines, which are A1 to A23, are available for use. Address line A0 is used inside the processor to control other signals: UDS (Upper Data Strobe) and LDS (Lower Data Strobe).[15]

  Even though, the Motorola 68000 microprocessor has a 16-bit external data bus, it is able to transfer 8 bits through the lower or the upper half of its data bus. The lower data bus is used to transfer all bytes that have even addresses and upper data bus is used to transfer memory addresses. Bus arbitration is another method of handling I/O. The BR (Bus Request) input get activated when a DMA is requested. By activating this input, the 68000 microprocessor generates a logic zero on the BG pin. BG shows that Motorola 68000 has stopped executing software and has open-circuit-ed its address, data and control bus connections. This allows an external DMA controller, or another microprocessor, to enter I/O and memory space of the microprocessor, therefore allowing the external controller to access memory and I/O directly. The bus request, bus grant, and bus grant acknowledge signals form an arbitration circuit to classify which device becomes the bus master device.[11]

Bus Request (BR):

This input is wired with bus request signals from all other devices that could be bus masters. The signal indicates to the processor that some other device needs to become the bus master. Bus requests can be issued any time during a cycle or between cycles.

Bus Grant (BG):

This output signal indicates to all other potential bus master devices that the processor will relinquish bus control at the end of the current bus cycle.

Bus Grant Acknowledge (BGACK):

This input indicates that some other device has become the bus master. This signal is not started before the following conditions are satisfied:

- A bus grant has been received.
- Address strobe is inactive, which indicates that the microprocessor is not using the bus.
- Data transfer acknowledges is inactive, which indicates that neither memory nor peripherals are using the bus.
- Bus grant acknowledge is inactive, which indicates that no other device is still claiming bus mastership.

Read-Modify-Write Cycle:

The read-modify-write cycle does a read operation, changes the data in the arithmetic logic unit (ALU), and writes the data back to the same address. The address strobe (AS) remains maintained throughout the entire cycle, causing the cycle indivisible. The test and set (TAS) instruction uses this cycle to provide a signaling feature without deadlock between processors in a multiprocessing situation. The TAS instruction is the only instruction which uses the read-modify-write cycle, operates on bytes only. Therefore, all read-modify-write cycles are byte operations. "Bus arbitration is a technique used by bus master devices to request, to be granted, and to acknowledge bus mastership."[11]

Requesting The Bus:

External devices capable of becoming bus masters maintain BR to request the bus. This signal can be wired from any of the devices in the system that can become bus master. The processor, which is at a lower priority level than the external devices, gives up the bus after it finishes the current bus cycle.[11]

Receiving The Bus Grant:

The processor maintains BG as soon as it can. In general, this process follows internal synchronization immediately, except when the processor has made an internal decision to execute the next bus cycle but has not yet maintained AS for that cycle. In this situation, BG is delayed until AS is affirmed to indicate to external devices that a bus cycle is in progress. BG can be routed through a network or through a specific priority-encoded network. Any method of external arbitration which observes the protocol can be used.[11]

Asynchronous data transfers are done using the following control signals: address strobe (AS), read/write (R/W), upper and lower data strobes (UDS, LDS), and data transfer acknowledge (DTACK). The address strobe signal shows there is a valid address on the address bus. Read/write defines the data bus transfer as a read or write cycle. The data strobes maintain the flow of data on the data bus and the data transfer acknowledge shows that the data transfer is finished.[15]

Asynchronous bus control:

Asynchronous data transfers are controlled by the following signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are described below:

Address Strobe (AS):

This three-state signal indicates that the information on the address bus is a valid address.

Read/Write (R/W):

This three-state signal defines the data bus transfer as a read or writes cycle. The R/W signal relates to the data strobe signals described in the following paragraphs.

Upper Data Strobe And Lower Data Strobes (UDS & LDS):

These three-state signals and R/W control the flow of data on the data bus. Table 3-1 lists the combinations of these signals and the corresponding data on the bus. When the R/W line is high, the processor reads from the data bus. When the R/W line is low, the processor drives the data bus. In 8-bit mode, UDS is always forced high and the LDS signal is used.

Data Transfer Acknowledge (DTACK):

This input signal indicates the completion of the data transfer. When the processor recognizes DTACK during a read cycle, data is latched, and the bus cycle is terminated. When DTACK is recognized during a write cycle, the bus cycle is terminated.

Lisa OS:

To meet the design goals and accommodate the applications planned for the Lisa, an OS had to be chosen that could handle the intensive graphics, have a reliable filesystem, do multiple tasks at once (multitasking), have good memory management, and have a mechanism for inter-process communication. This left most of the popular operating systems out (e.g. DOS, CP/M) because they could not perform multitasking. UNIX would have been a good choice, but at the time was too large to be of practical use and no graphics or IPC support had been developed as yet. It had a very fragile filesystem that the end-user would likely not be able to recover data from in the event of a system crash. Re-writing UNIX to conform to the Lisa's requirements would have been very costly and more time consuming than was practical, so Apple wrote an OS for the Lisa from the ground up. There are four main functional areas to the Lisa Operating System. It manages files, processes, memory, and handles events and exceptions.[1]

- **Filesystem** - The Lisa OS handled files in a similar fashion to UNIX and Multics. Devices and disks had to be mounted to be used by the OS. All I/O was device independent and was handled as an uninterpreted stream of bytes, so all objects were treated the same way with special functions used to handle particular needs of a device when required. Since one of the goals of the Lisa was reliability, a departure was made from the UNIX filesystem (as it existed at the time) and some redundancy was built in to keep users from losing data in the event of a system crash. Each file maintained its descriptive attributes in the master disk and in a block at the beginning of the file. Each block also maintained information to tell which file it belonged to. Apple provided a "scavenger" program that could rebuild the disk catalog if it became corrupted.[1]

- **Process Management** - The Lisa OS was able to run multiple processes at once, similar to a UNIX environment without the multi-user features. Processes are organized in a tree structure and can be spawned by a master process that is initiated at boot time. A similar multiplexed, prioritized scheduler is used to allow processes to get CPU time, but unlike UNIX, a nonpreemptive algorithm is employed. By implementing this type of algorithm, the overhead of locking and unlocking resources prior to use is avoided. The Lisa OS allows all processes to perform certain functions. Processes can "suspend, activate, kill, or otherwise control any other process."[1] Termination of a parent process causes all of its child process to be terminated as well. Inter-process communication (IPC) is done by sharing files, data segments in memory, and/or events. To make the operating system more reliable, each process has its own stack space and logical address space.[1]

- **Memory Management** - Since the Motorola MC 68000 did not provide virtual memory capability, this functionality was relegated to the OS and a separate memory management unit (MMU). Apple designers used a segmented memory model to implement virtual memory. Processes have data and code segments. The data segment is created as a stack and the code segments, specified by the programmer at compile time, help break the program up into smaller parts. All data segments for a program have to be loaded into memory before that program can run. This is because the instructions that reference the data segments are not restartable. Segments of code, however, are loaded into memory from disk as needed. The process can request additional stack space if needed - up to sixteen additional segments.[1]

  Code segments can be either intrinsic (system libraries) or regular (shared by processes that are running different instances of the same program). The reason that code segments can be swapped in as needed is because the instructions that access the code segments are restartable. If one of those instructions (there are four - JMP, JSR, RTS, and RTE) tries to access a code segment that is not currently loaded into memory, it will cause a bus error. This causes a trap to be sent to the Lisa OS and it will then load the segment it needs and restart the instruction without the running program being interrupted.[1]

  When the physical memory becomes full, the Lisa OS uses a clock page replacement algorithm to determine which memory segments it will swap out. Information about all of the memory segments is kept in a Segment Descriptor Block (SDB) that is internal to the MMU. The clock page replacement algorithm is based upon a circular list that can be thought of as a clock face. In this case, the circular SDB list is represented as the clock face. There is a pointer that can be thought of as a clock hand and when a page fault occurs, the segment that the clock hand is pointing to is examined. Upon examination, this can result in either swapping out the current segment and advancing the hand, or ignoring the current segment and advancing the hand, then examining the next segment until if finds enough free segments or swaps out enough segments to load the segments that are being requested. The bus error method is used here as well to determine which segments of memory haven't been used and need to be swapped out.[1][10]

- **Exception and Event Processing** - The Lisa OS also handles certain types of errors that occur during process execution. These errors are called exceptions. Usually, they are caused when a process tries to execute an illegal instruction, tries to access a memory address that is not allocated to that process, tries to divide by zero, or other errors that would cause problems elsewhere in the system. When one of these errors happens, a process can use one of three types of exception handlers: OS-supplied default handlers that will simply terminate the process and keep it from interfering with other processes, handlers that are supplied by the process, or user defined handlers.[1]

  Events occur when processes need to pass messages or control signals to each other. In the Lisa OS, the system was written the event headers and it is up to the sending process to write the event. The receiving process interprets the event message. Events are passed on channels and the names of these channels are cataloged by the file system. The Lisa OS maintained an event channel with no name that allows processes to get system events that relate to its child processes [1].

OS Support:

The Apple Lisa supports several operating systems in addition to the Lisa OS including Unix, CP/M, XENIX, and the Mac OS (with the MacWorks emulator). [4]

Lisa Software:

To make the Lisa a useful tool, Apple released a productivity suite known as the Lisa Office System (later called "Lisa 7/7" because "seven sevenths make a whole" [9]). These applications were precursors to modern office software. Apple claimed that anyone could learn these applications and put them to productive use in about 30 minutes. All of them claimed WYSIWYG (What-You-See-Is-What-You-Get) fidelity for printing, a "revert to previous version" feature that would allow a user to return to a previous document version if mistakes were made, and data from one application could be inserted into another. [8]

The Lisa's word processor was called LisaWrite. The document size was only limited to the amount of disk space available. Spreadsheet functions were performed by LisaCalc and each spreadsheet could have a maximum of 255 rows and 255 columns. LisaGraph was used to generate graphs that could be directly created from LisaCalc or by manual input. It would display pie, bar, line, and scatter graphs with maximum of around 2000 data points and the graph would update instantly. LisaList was a personal database program. It could handle databases of approximately 600KBytes. It had a maximum row size of 990 bytes and allowed for a maximum of 100 columns. Data was indexed using B* indexing, and data could be represented as text, number, date, money, social security number, time, phone number, and zip code. LisaTerminal allowed connections (via serial ports/modems) to other computers. You could connect using either of the two serial ports, and LisaTerminal provided VT52, VT100, and TTY emulation at speeds ranging from 50 to 19,200 baud. With Apple's Cluster Controller Emulator, the Lisa could be used as an IBM 3270 display station [8]. Project planning, management, and scheduling were done by LisaProject. It could do parallel or resource scheduling and could create project charts for visually displaying project plans, schedules, and tasks. [8]

To further its claim of the Lisa being easy to use, Apple developed an online help system called LisaGuide. People could teach themselves how to use all of the Lisa's features with the LisaGuide online training course. If the printed manuals were not available, LisaGuide had all of the necessary documentation online to answer nearly any question a user might have. There was a hardware diagnostic program called LisaTest that was sold for a brief period, but Apple discontinued it and referred users to Apple dealers for hardware diagnostics. [3]

Most of the Lisa applications were written in Pascal. To encourage third party developers, Apple provided the Lisa Workshop development environment. It was a scaled down version of the Lisa Monitor development environment used by Apple's internal developers and focused mainly on Pascal. There was also a Lisa ToolKit that allowed outside developers to access the Lisa Desktop Libraries. [3]

Other Lisa Features:

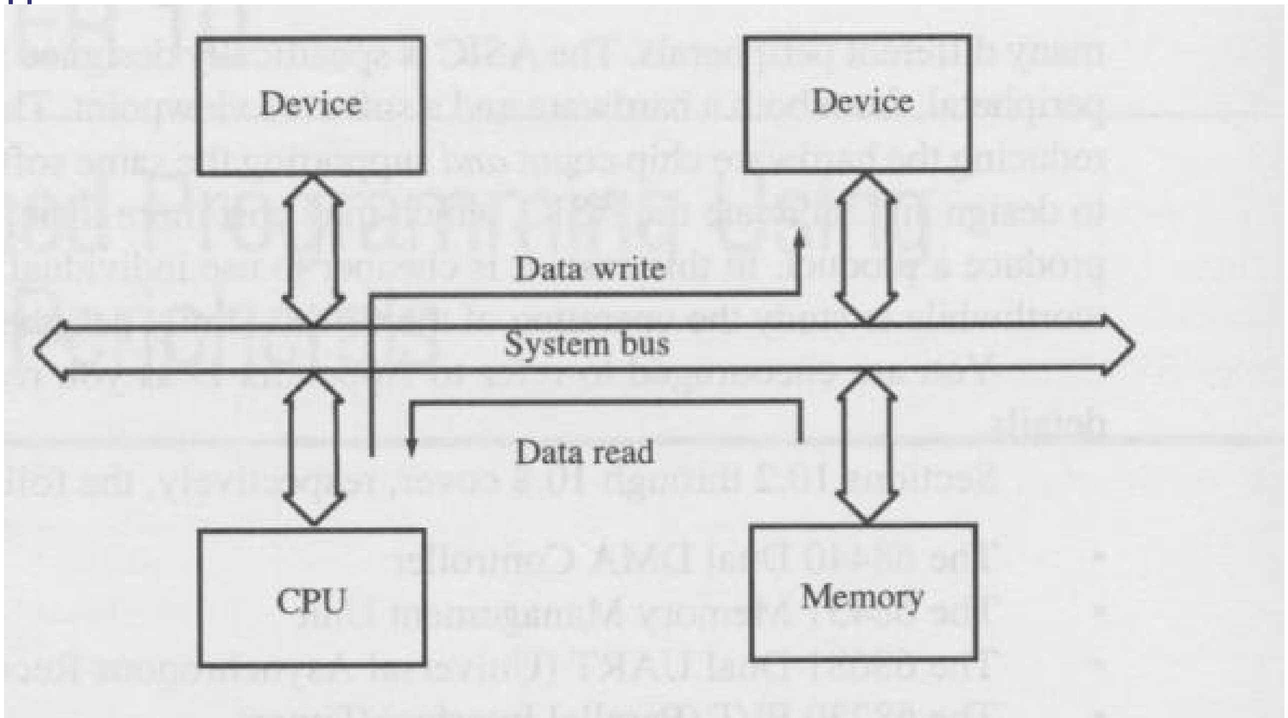
The Lisa had several other features that are worth mentioning. When the Lisa was powered off, it would save its desktop state and all application states and go into a "standby" mode. When it was powered back on, all of the users applications and documents were restored to their previous state. There was a software adjustable contrast setting for the monitor and a dimming feature that would dim the screen to protect it after a period of inactivity.

References:

1. Bruce Daniels, The Architecture of the Lisa Personal Computer, November, 1983
2. The Lisa/Lisa 2/Mac XL, http://www.apple-history.com/quickgallery.html?where=lisa.html
3. David T. Craig, The Legacy of the Apple Lisa Personal Computer: An Outsider's View, http://lisa.sunder.net/mirrors/Simon/Lisa/LisaLegacy/LegacyIndex.html
4. Glen Sanford, www.apple-history.com, http://www.apple-history.com/lisa.html, 1986-2002
5. Apple Computer, Inc. Lisa Hardware Reference Manual, http://www.applefritter.com/lisa/texts/LisaHardwareManual1981.pdf, 1981
6. Mark Smotherman, A Brief History of Microprogramming, http://mprc.pku.edu.cn/users/chengxu/Org_web_ext/BriefHist_up/uprog.html, March 1999
7. Chip Weems, University of Massachusetts Amherst Computer Science 535, Chapter 15 Control Unit, http://cs.ddart.net/computer_architect/CmpSci535/Discussion15.html, 1995, 1996, 2001
8. Apple Computer, Inc. Product Specification Brochure, http://www.archaic-apples.com/files/lisa/brochure/, November 1983
9. Semaphore Corporation, Semaphore Signal, Issue 14, Exploring Lisa's New Office System, http://www.applefritter.com/lisa/texts/LisaSemaphoreSignal.pdf, July 1984
10. William Sawyer, CPSC 342 - Operating Systems - Memory Management, http://www.cs.bilkent.edu.tr/~will/courses/CS342/HTML%20slides/Chapter-04/sld024.htm, 31-Jan-2002
11. J. L. Antonakos, The 68000 Microprocessor: Hardware and Software Principles and Applications, Second and Fourth Editions, Prentice Hall, Columbus, Ohio, 1998
12. Zargham, M.R. Computer Architecture: Single and Parallel Systems, Prentice-Hall, N.J., 1996
13. Wikipedia, The Free Online Encyclopedia, http://www.wikipedia.org/wiki/Motorola_68000_family
14. David D. Redhed, The Lisa 2: Apple's Ablest Computer, Byte, vol. 9, Dec. 1984, pp. A106-A115

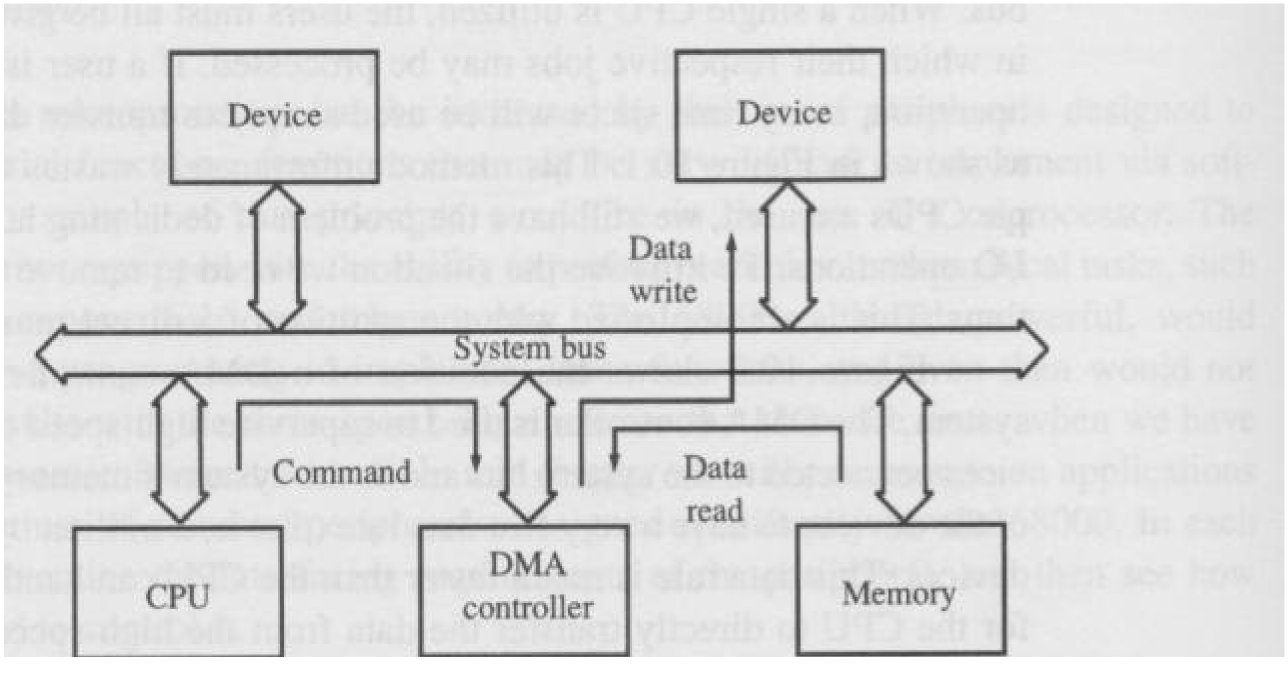
15. Barry B. Brey, The Motorola Microprocessor Family: 68000, 68008, 68010, 68020, 68030, and 68040, Saunders/HJB, 1992
16. Thomas L. Harmon and Barbara Lawson, The Motorola MC68000 Microprocessor Family, Prentice-Hall, 1985
17. The Free Online Dictionary of Computing, Motorola 68000, http://burks.brighton.ac.uk/burks/foldoc/81/75.htm, 1999
18. Chris Chabris, Introducing 520ST Assembly Language - MC 68000 Tutorial, Antic Vol 4, No. 8, December 1985
19. Gabriel Acosta-Lopez, Richard Clark, and Anne Wysocki, Two Representative CISC Designs, http://physinfo.ulb.ac.be/divers_html/PowerPC_Programming_Info/intro_to_risc/irt4_cisc3.html, 1995
20. Alan Clements, Microprocessor System Design - 68000 Hardware, Software, and Interfacing, PWS, 1987

Appendix:



A system using the CPU to perform all I/O

Figure 1



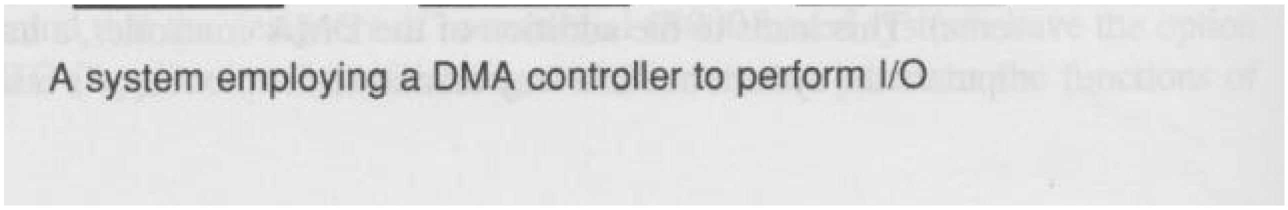
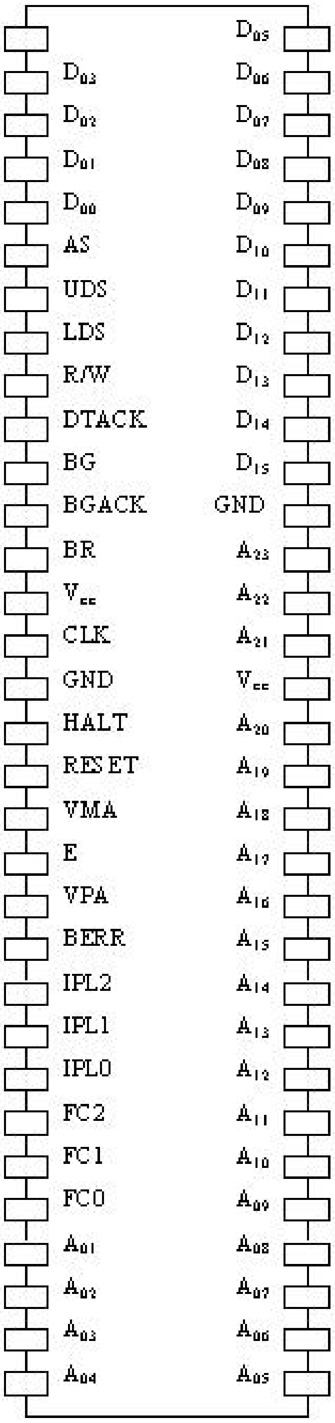


Figure 2

Motorola 68000 Pin Configuration



Miscellaneous

V_{cc}(2), GND(2) – 4 pins

Function Code

FC0-FC2 – 3 pins

System Control

CLK, RESET, HALT – 3 pins

Interrupt Control

ILP0-ILP2 – 3 pins

Bus Arbitration Control

BR, BG, BGACK – 3 pins

Synchronous Bus Control

E, VPA, VMA – 3 pins

Asynchronous Bus Control

AS, R/W, UDS, LDS,DTACK – 6 pins

Data Bus

D_{00}-D_{15} – 16 pins

Address Bus

A_{01}-A_{23} – 23 pins

Figure 3

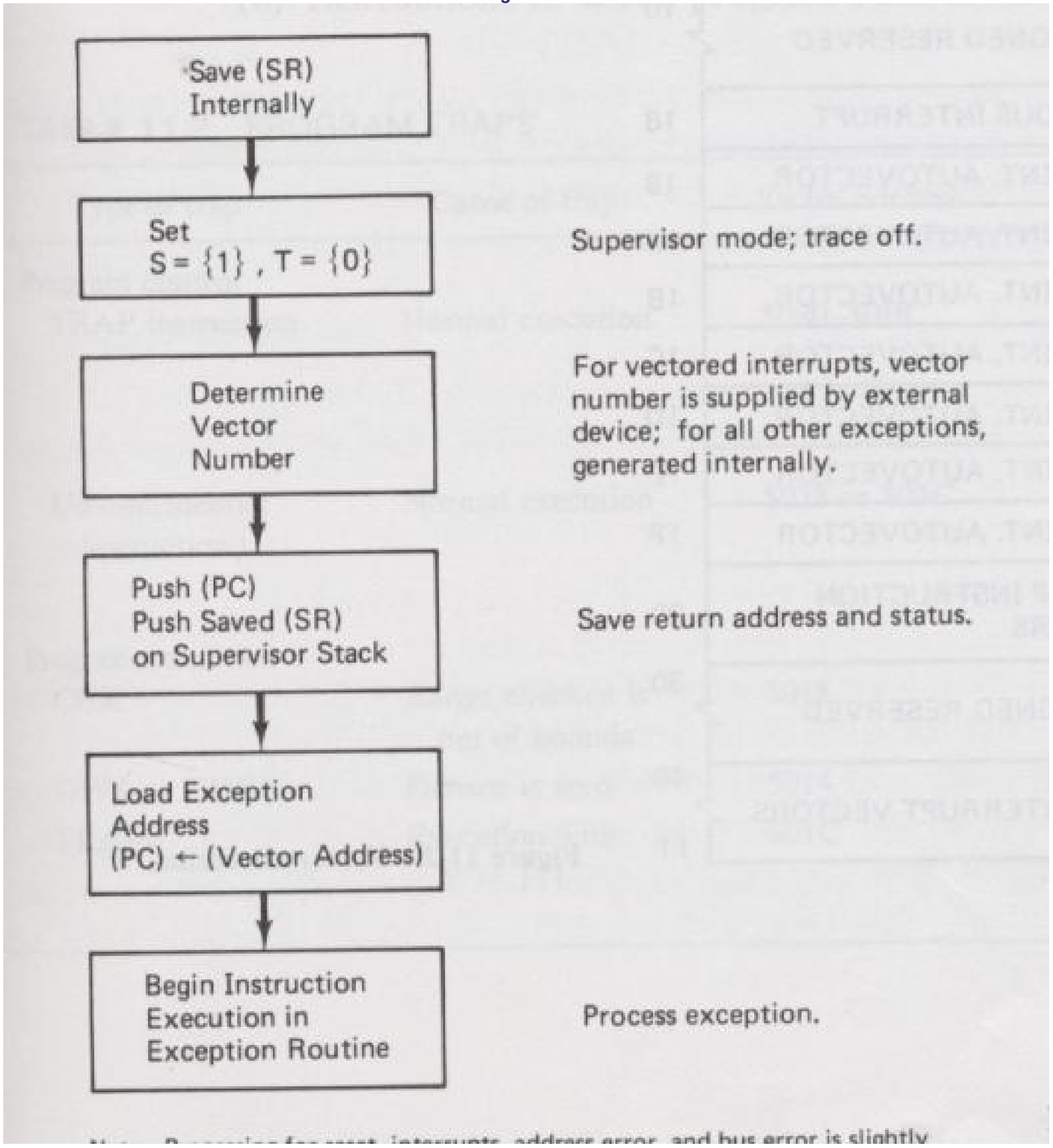
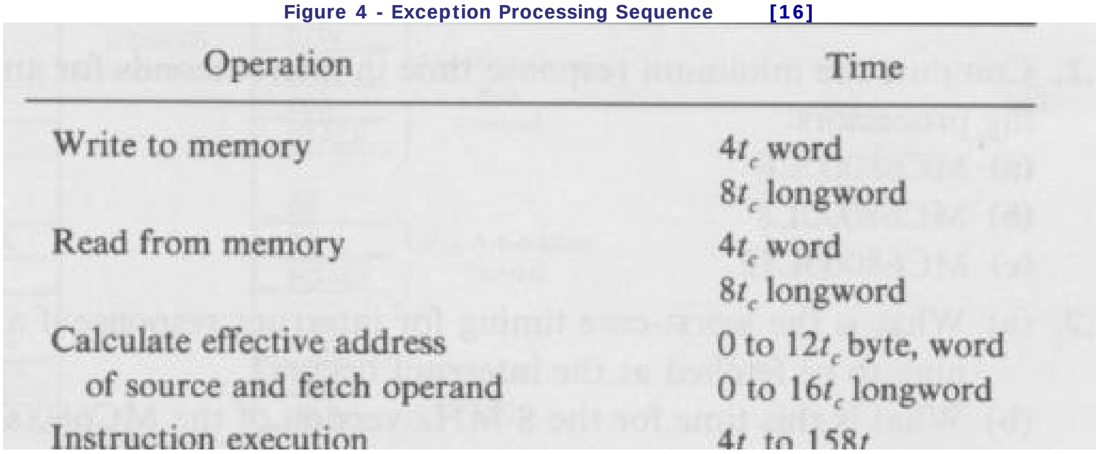
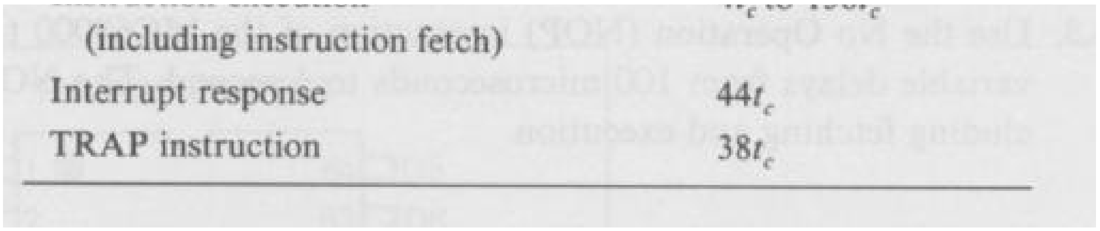

Save (SR) Internally
Set S = {1} , T = {0}
Supervisor mode; trace off.
Determine Vector Number
For vectored interrupts, vector number is supplied by external device; for all other exceptions, generated internally.
Push (PC) Push Saved (SR) on Supervisor Stack
Save return address and status.
Load Exception Address (PC) ← (Vector Address)
Begin Instruction Execution in Exception Routine
Process exception.

Figure 4 - Exception Processing Sequence [16]



Operation	Time
Write to memory	$4t_c$ word $8t_c$ longword
Read from memory	$4t_c$ word $8t_c$ longword
Calculate effective address of source and fetch operand	0 to $12t_c$ byte, word 0 to $16t_c$ longword
Instruction execution	$4t_c$ to $158t_c$



(including instruction fetch)	
Interrupt response	$44t_c$
TRAP instruction	$38t_c$

Figure 5 - Instruction Execution Time [16]