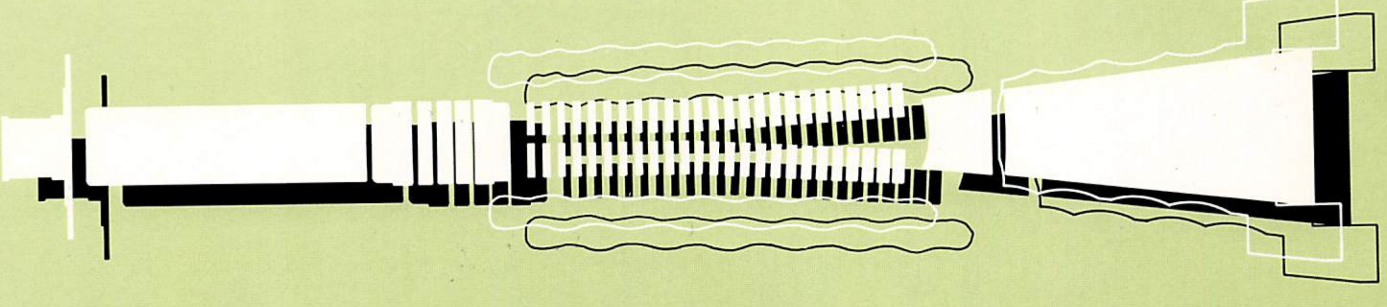
TEK LABS

M A G N O L I A
A
Single User System
Design and Implementation Plan

Roger D. Bates

APPLIED RESEARCH

TECHNICAL REPORT



TEK PRIVATE

TEKTRONIX®

M A G N O L I A
A
Single User System

Design and Implementation Plan

Roger D. Bates

Computer Research
Applied Research Group
Tektronix Laboratories

September 16, 1980

M A G N O L I A
A
Single User System

Design and Implementation Plan

ABSTRACT

This report describes a Single User System which I am calling Magnolia. It addresses many of the requirements for a high performance work station capable of supporting various hardware/software development activities.

Table of Contents

M A G N O L I A
A
Single User System

Design and Implementation Plan

Roger D. Bates

Computer Research
Applied Research Group
Tektronix Laboratories

1. Introduction

The current trend of supplying "smart" terminals for use as access to central computers is an interim solution to the economic balance of computer system design. As more and more compute power is placed in the "terminal", it will be able to take over virtually all of the computational needs, and will become the computer system that the user interacts with. At this point, the display functions will become an integral part of the computing environment.

Low performance systems of this type are currently called "personal computers." In order to maintain a distinction between this type of system and higher performance systems being developed, we are calling the latter "Single User Systems."

For Tektronix to be able to move into single user systems, we need to learn how to design, build, and program this type of computer. A product that is suitable for selling to a customer will have to have not just the hardware and operating system, but a great deal of application software in order to deliver the full "capability" of the system. Tektronix will not only have to be competitive in the hardware, but in the software to be able to compete with systems of the future.

The best way to learn what to put in such a system is to provide a Single User System tool. With this we can program and use experimental tools and thereby learn what the strengths and weaknesses of this technology are. Magnolia is a tool that we can use and learn from in order to better prepare to compete in this market.

2. Design Philosophy

The motivation for this study is to present a high performance system that, if implemented, would provide a single user computing tool which would extend the resources available over that which is available on our current 11/70, or even a VAX class multi-user system. I believe that it is important that we address this class of system in order to provide for the needs of computer research in CAD, VLSI, interactive graphics etc.

It is proposed here that we can not only design a system in this performance range, but that if the architecture is right, the system implementation cost can compete with the selling price of the 4025. I believe that it is important to make the right performance/cost tradeoffs in order to make it cost effective to provide more than just a single demonstration unit within CRG.

The major approaches that were considered in this design are summarized below:

- Use an available 16 bit microprocessor in order to avoid designing a new processor and assembly language.
- Allow for performance improvements by developing a system architecture which will allow for multiple/mixed processors.
- Provide a software environment which minimizes the start-up effort required.
- Provide for a maximum performance user interface in order to encourage uses and experiments in this area.
- Provide for a system with maximum flexibility and potential in the program areas that CRG is currently involved or likely to investigate.

The primary result that comes out of the above list is a design to provide a valuable **tool** within Tektronix. This does not rule out the secondary goal of demonstrating an architecture that could prove valuable to future IDD or MDP products.

3. Overall Hardware Description

The hardware architecture of Magnolia can best be seen by looking at Fig. 1. This drawing shows a system consisting of three microprocessors. Each one has a local "Bus" on which program and local data transfers take place. In addition, there is a common "Object" bus which is used for communication between processors. The protocol of the object bus is tailored to "high level language" designs, and is described in full detail later on.

The top processor shown in Fig. 1 can be thought of as the "system processor." This would initially be a very conventional processor with controllers for all the Input/Output devices, 1 or more boards of local memory, and some form of memory management. The "operating system" program would run in this processor, along with most of the application programs.

The middle processor is shown to represent the possible inclusion of 1 or more additional processors. These would be added when the application required enhanced performance. A "kernel" program would be required in this processor to interact with the Object Bus, handle

MAGNOLIA

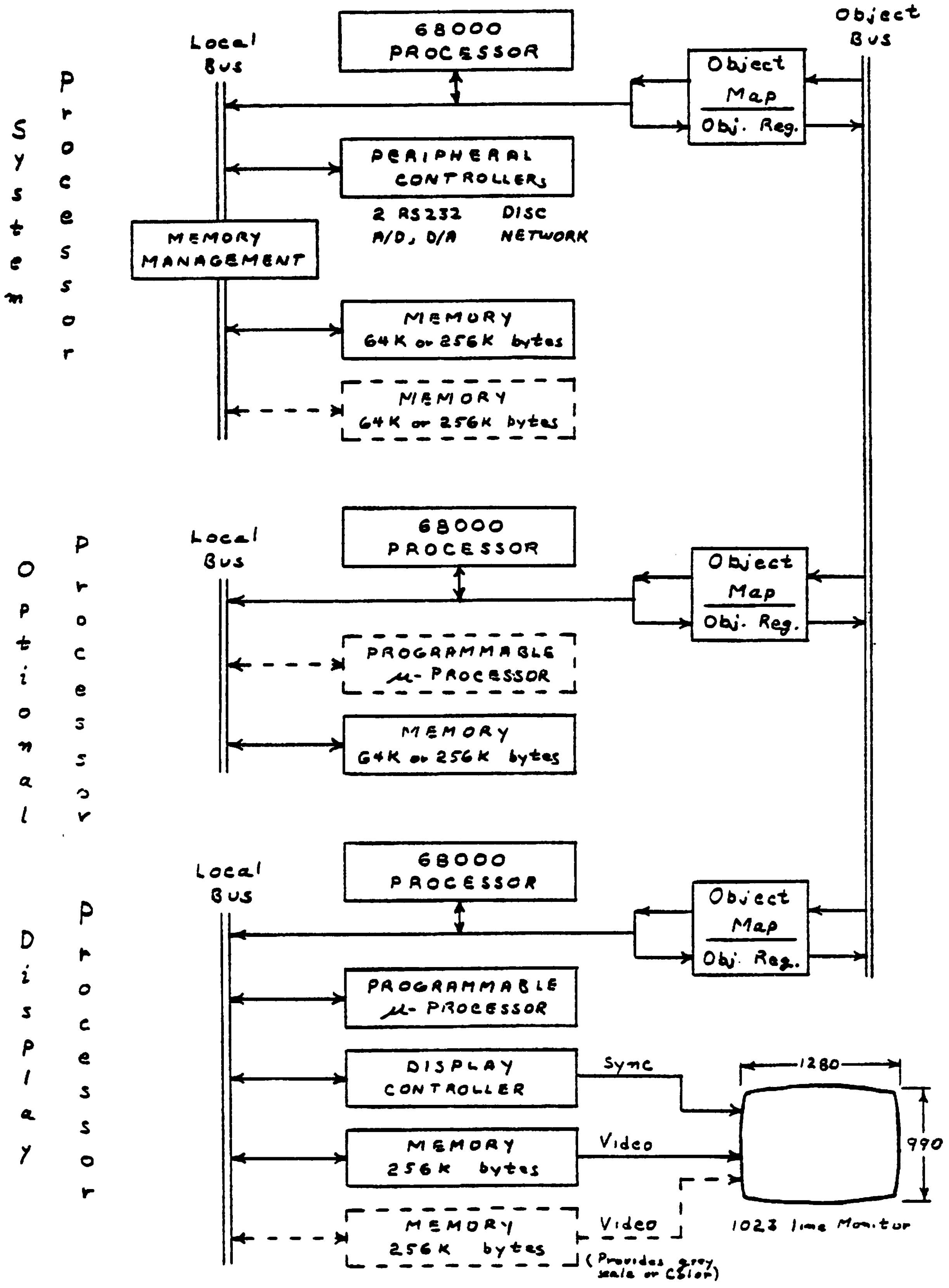


Fig. 1

5/29/80
R. Bate

procedure calls, and manage memory allocation, etc. This code would be quite minimal, leaving the bulk of the processor for application programs. In this particular example, an additional custom processor has been included which could provide enhanced performance for data manipulation or floating-point calculations etc.

The processor at the bottom of the drawing is the "display processor." The display system will consist of a high resolution image and a raster scan display processor that creates its image using a 990 pixel high by 1280 pixel wide "bit-map" image. The processor will implement display capabilities such as insert or delete objects such as lines or characters on the display. An additional microprogrammable processor is shown in this processor in order to enhance the performance of doing bit manipulation of the display memory. If this co-processor is not initially implemented, then the microprocessor could perform this function with degraded performance. The Display Controller is included as a minimal circuit to provide addresses to cycle through memory for refresh of the display. If the microcoded processor is present, this function could be done there.

A possible "packaging" implementation of the hardware system just described is given in Fig. 2. This shows a system where the monitor and movable keyboard sit on the user's table. The electronics (and optional disk) are placed in a separate enclosure that can be placed out of the way such as under the desk. This implementation minimizes the table surface that is occupied by the system.

4. Overall Software Description

The initial Magnolia system will be brought up with a version of the UNIX operating system. The two initial programming languages will be "C", with a minimal reliance on the microprocessor's assembly language. This has a number of significant advantages summarized below.

- Less risk is accepted when avoiding the simultaneous development of both new hardware and new software. The currently predominant software environment within CRG is UNIX and the "C" language. In this case, the current software environment should be brought up on a new hardware environment, and if desired in the future, a new software environment can be experimented with on what will then be an established hardware base.

- There is a large user community which exists around the UNIX operating system, and it is in ARG's best interest to remain a part of that community.

- There are a number of programs already written which would be useful in this environment. Screen text editors are available which we could use. It is also likely that VLSI and CAD programs for VLSI and other areas would be available for running under a UNIX environment.

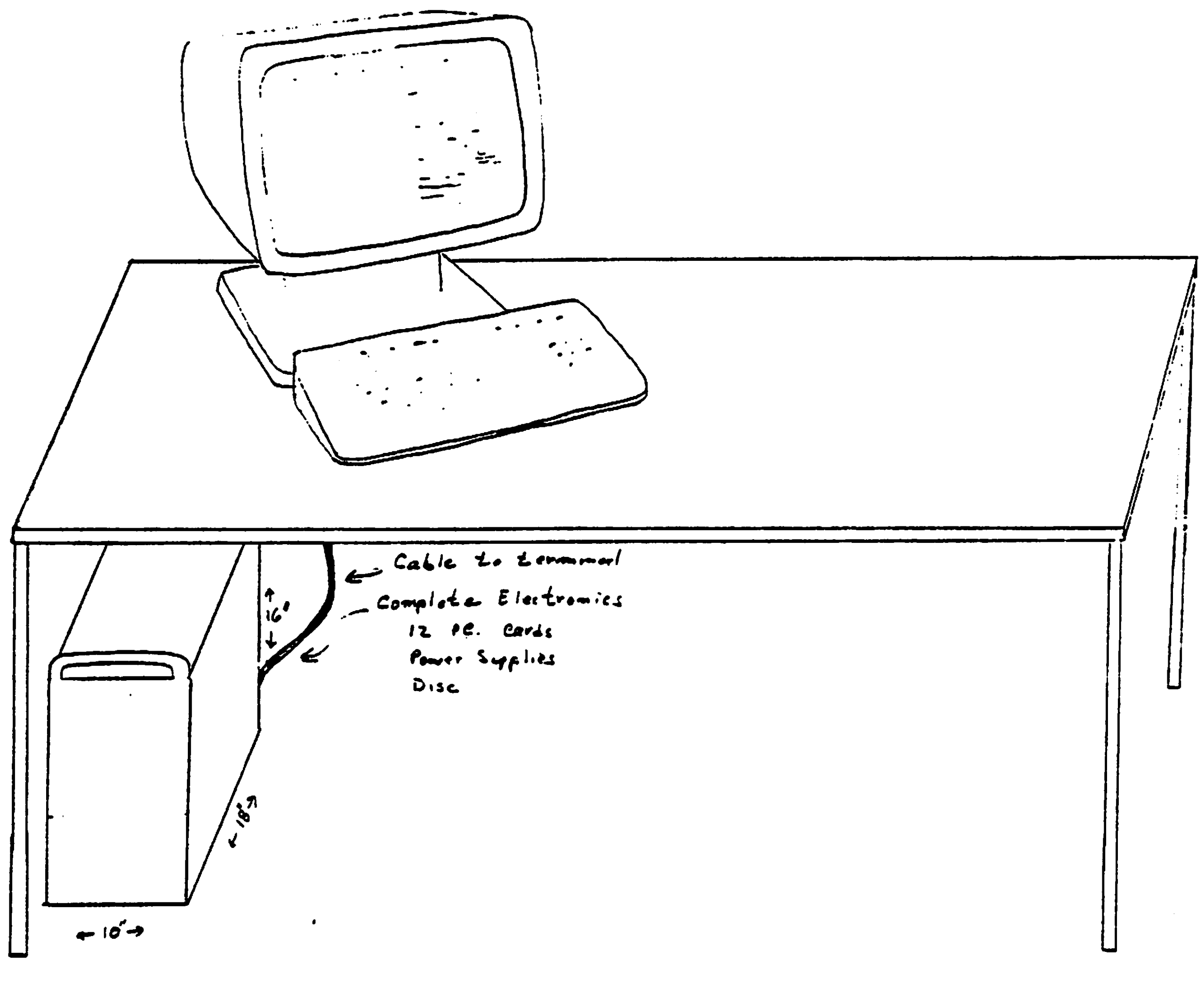

Cable to terminal
Complete Electronics
12 PC. Cards
Power Supplies
Disc
16"
18"
10"

FIG. 2

- The "C" language is well suited for implementing operating systems. This language provides most of the High Level Language (HLL) constructs that are needed, while still providing the low level operations that allow the system to minimize the amount of code written in assembly language.

The C compiler and loader can enable a program to be written as a group of well organized modules. The loader will need to have knowledge of the multiple processors, and implement commands to load code segments into various processors.

The UNIX operating system will need to be tailored to the Magnolia environment. The portions of code associated with multi-users can be eliminated, while multi-tasking will remain. The operating system will need to manipulate "windows" on the bit-map display where different tasks will get assigned different "window" areas. UNIX commands will have to be added to control the size and location of these areas.

5. Alternative Systems

It is important that we look at alternatives to building this computer, and recognize the appropriate advantages and disadvantages. The following list represents several of the alternatives that have been considered:

5.1. 4061 - TINA

This is a an obvious candidate in the class of a Single User System. As a product of Tek, it would be desirable to use this system. The draw backs become obvious as soon as one looks at performing experiments with it. The bit map display capability is built in to a dedicated display processor, and contemplating adding to or modifying display capabilities is not practical. While it does have a microcoded processor underneath, the control store is not writable, and it is not feasible to make it writable. Finally, it does not have its own local high speed hard disk, and is not software compatible with the work already done in 'C' on UNIX.

5.2. PERQ

This is a personal computer that has been "about to be released" from Three Rivers for some time now. It looks like they still have a lot of work to do. It is a microcoded processor designed to run Pascal code in a UNIX like environment. It does have a flexible bit map display and writable control store. The primary draw backs are availability and cost (around $20K each). It is also inherently a uniprocessor and doesn't lend itself to experiments in multiprocessor systems.

5.3. HP System 35, 45B, 300

Hewlett Packard is probably the closest product that is currently available. The systems emphasize the user interaction through the display, however the processor and display are not integrated as tightly as they should be, and the display resolution is quite low (less that 512 X 512 pixels). The low end systems provide BASIC as the programming language, while the high end systems are too much for a single user system - HP intends for them to be used in a multi-terminal installation.

5.4. ALTO

This is essentially the model with which all Single User Systems are currently compared. It is an internal product of XEROX, built for its own use. A number of ALTOs are now being given to selected universities, but they are not for sale.

6. Possible Applications

The uses for a single user system that come immediately to mind are listed below. The range of tasks appropriate are quite diverse because of the flexibility inherent in the hardware/software configuration.

6.1. Minimal Cost Personal Computer

This is the configuration that will first be implemented in bringing up Magnolia. It consists of a single processor board, local memory, and an interface board. The interface board contains two serial interfaces, one to connect with a local terminal and the other to connect with a host computer.

This configuration can be optionally enhanced with a floppy disk or Winchester hard disk. The next upgrade would be a second processor to implement the bit-map display.

In all of these hardware configurations, the UNIX operating system and user programs would run in a single microprocessor.

6.2. Multi-processor Applications

Investigations in the areas of data base and data driven systems could easily gain from the presence of multiple processors. The basic architecture and physical implementation of Magnolia will make it practical to build up a system consisting of say 8 or more processors if some project would warrant such an approach.

6.3. Processor Architecture

A project involved in implementing an experimental processor architecture needs a way to implement the basic hardware without having to provide the full operating system and I/O. Placing a new capability on the Object bus, loading in code and running it from the existing system provides a practical solution.

6.4. User Interface

The mode of communication with a user from a high performance Single User System needs to be qualitatively different from that of a time-shared system, or low performance personal computer. There is a strong need to investigate this area carefully, both in the area of display management as well as input pointing devices.

6.5. Interactive Editing

One of the most powerful advantages of a high performance Single User System comes from the possibility of providing high quality, fast response, easy to use interactions with the user in areas such as text editing and graphics editing. An example of a system used for graphics editing is the Tek 4081. Its mode of editing however is directed towards input of a "stable" drawing which will require only small updates. Systems like Magnolia have the potential for allowing a user to draw his conceptual design on a system, updating it as quickly and easily as he does with paper and pencil. This approach is not appreciated by many people, and its practicality needs to be demonstrated.

6.6. Advanced CAD programs

Many of the CAD programs which are currently available are sufficiently compute intensive that only a relatively small number of users can be supported on a time-shared system. Magnolia is aimed at equaling or exceeding the capacity of a system such as our 11/70 in order to efficiently run powerful and large design programs.

7. Detailed Description

This section will fill in as much detail of the Magnolia system as I can at this point. Some areas are filled in more than others, but the details presented here will give a clear idea of the direction the design would follow.

7.1. Processor

The introduction of 16 bit microprocessors makes possible building a high performance Single User System without basing the design on MSI components. The three processors considered were Intel 8086, Zilog Z8001, and the Motorola 68000. The three processors were compared based on criterias such as quality of instruction set, ease of electrical interface, and speed.

- The 8086 has a poorly ordered instruction set, and too small an address space. On the plus side is the chip support that Intel provides around it in terms of interfaces and addition of extended instructions set(s).

- The Z8001, which is not currently available, has an "almost" good instruction set - but many of the addressing modes are missing on most of the instructions. Even though the address space is large, you have to contend with a segmented address scheme which is annoying.

- The 68000 has the best instruction set, it is complete, symmetric, and coincidentally most similar to the DEC PDP 11 series computers. The address space is large and contiguous which should lead to cleaner programs. In addition, the internal architecture is entirely 32 bits wide, and is the most likely of the three to expand that capability to its I/O interface in the future.

Based on the above comparison, the Motorola 68000 will be the microprocessor used for each of the processors in Magnolia. Note that with the use of the Object Bus interface different processor chips could be used. I an very concerned with ease of software development, and therefore want identical processors so that program modules can easily be moved around within the system.

Besides the basic processor, there are a number of other capabilities to be implemented on the processor board. The most significant is the Object Bus interface described below. A programmable time chip will make timed interrupts available to all processors. A program PROM will be provided which contains enough code to "boot" the processor after power up as well as provide some basic debugging aids. This PROM will be initially mapped into address 0 of the 68000, but will later be mapped to a very high address location where it will be available for execution at any other time. Finally, there will be a simple mapped memory register supplied so that unforeseen slow and medium speed I/O devices can be easily added to any one of the processors.

7.2. Object Bus

The Object Bus represents the mechanism through which the various processors communicate with each other. This is probably the most unique feature of Magnolia, and has evolved through numerous conversations with Tom Cook. The primary objective is to provide access to data structures (Objects) which reside at unknown locations of unknown processors - knowing only the unique ID (Object number) for that data structure.

There are essentially two processor communication approaches currently in use.

- The first is to give each processor a two port memory, one port for local references, and another for remote references. All memories would be addressable via a sufficiently large address space. This is a very unstructured approach and requires a processor to know the exact state of every other processor's memory. This gives no memory protection from errant writes from any of the other processors. This is simplest to implement in hardware, simplest to start in software, hardest to maintain reliably, and is probably the most common approach used.

- The second approach is to provide a "mini network" between the processors through which messages are passed. This is a very structured approach, but can result in a lot of data passing and protocol requirements. While this approach provides the necessary memory protection, I feel it will be too slow and inflexible for the fast low level interactions appropriate within a multiprocessor computer.

The approach described here will allow for random read/write operations into data structures referenced by an ID (Object Number). The location (which processor and where in its memory) will not be known by the requesting processor. Finally read/write protection will be provided to maximize memory integrity.

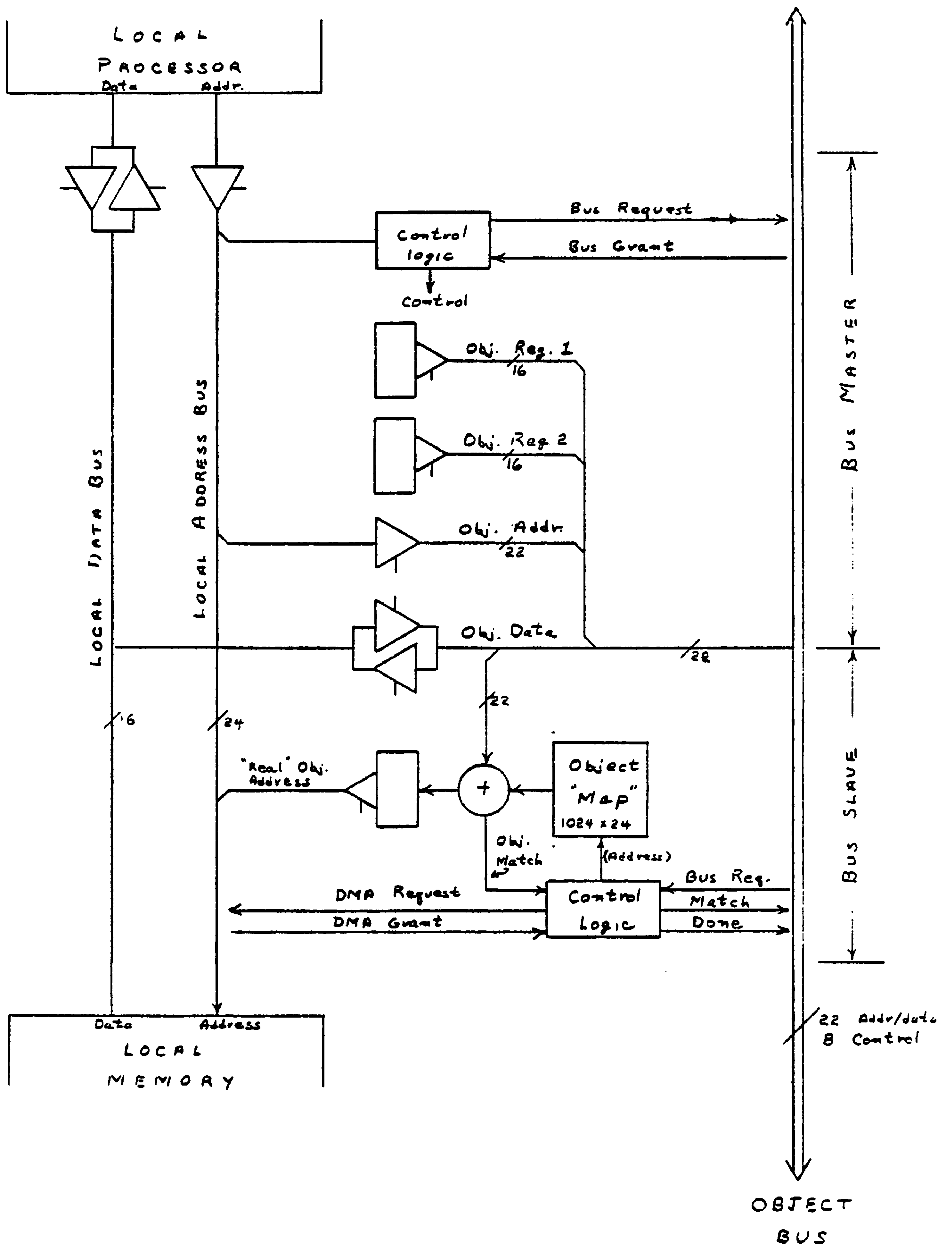
A block diagram of a processor tied to an Object Bus is shown in Fig. 3. The hardware is broken down into 2 sections. The first is used to initiate a transfer onto the Object Bus. The processor which is initiating a transfer is referred to as the "Bus Master", The second section responds to transfer requests on the Object Bus, and is referred to as the "Bus Slave." The two sections will be implemented to run simultaneously (full duplex) so that an Object reference by one processor may get a response from that same processor if the data structure resides in that processors memory space.

7.2.1. Bus Master

The Bus Master hardware consists of two Object registers, bus request logic, address drivers, and data transceivers. The Object Registers are used to hold the Object Numbers that will be used should an Object Reference be executed by the 68000 program. An Object Reference is sensed by monitoring the address range specified by a 68000 memory reference. The 16 megabytes of 68000 address space are divided up as follows:

Mbyte 15:	Object reference (use Object Reg. 1)
Mbyte 14:	Object reference (use Object Reg. 2)
Mbyte 13:	Hardware registers and memories etc.
Mbyte 12,,	
Mbyte 00:	Local memory address space

Each Object reference has one megabyte of address range reserved. This portion of the address space is used to specify an address within the object being accessed. This means that for some purposes (ie debugging), an entire



OBJECT BUS INTERFACE

FIG. 3

5/29/80

R. Bate

local memory could be addressed as an object.

To use the Object Bus, a program must first load the appropriate Object Register with the Object Number that it wants to access. The program then makes a memory reference within the megabyte range of the Object Register. When this happens, local memory reference is inhibited, the Bus Master logic places a request for Object Bus control, and waits until it has control of the Bus. It then places the Object Number (from the register) on the Object Bus and waits for a response from some (unspecified) Bus Slave that has the indicated Object Number. The Bus Master then sends the 22 bits of address information onto the address bus. This represents the address WITHIN the object, which may be up to 1 megabyte. Finally, the data is sent or received as appropriate and the Object Bus is released.

7.2.2. Bus Slave

The Bus Slave hardware consists of a "map" memory, DMA (Direct Memory Access) logic, and timing & control logic. The Map memory is 1K words long by 24 bits wide, and contains 4 words of information for each of 256 possible object entries. The format of a 4 word entry is given below.

```
Wd 0: 23,,16 not defined;  15,,00: Object Number
Wd 1: 23,,00: starting address of object in local memory
Wd 2: 23: Interrupt;   22,,00:  number of readable bytes
Wd 3: 23: Interrupt;   22,,00:  number of writable bytes
```

Words 2 and 3 are used during a read or write operation respectively to check that the object address received is valid (less than the maximum specified), and to see whether the local processor should be interrupted as a result of the object reference. Notice that by defining the data structure of an Object correctly, the object could provide the first few words as "read/write", while making the rest of the object read only.

The Bus Slave logic is always looking for requests on the Object Bus. When it sees a request, it first compares the Object Number on the bus with the last two object numbers it processed. If neither of these compare then the control logic begins a scan of the Map, stopping if it finds a match or reaches the end of the map. If a match is found by any of the three tests described above then a response is sent back over the Object Bus requesting the address portion of the Object Reference. The slave logic next reads the object base address out of the map, adds the object address then present on the Object Bus and saves the result in an address register. Next, a DMA cycle is requested from the local processor. While waiting for a DMA acknowledgement, the logic looks up the size of the object in the Map and compares that value with the address sent on the Bus. If the address is outside the range allowed for the reference then the access is aborted and an error flag is returned to the Bus Master.

7.2.3. Object Bus Performance

Since the Object Bus is the one resource that is shared, the performance of the bus is critical (especially if additional processors are added for performance). In order to make the bus as powerful as possible, the following features should be implemented:

1) Minimize the Map search time. First off, the Bus Slave will keep track of the last two object numbers accessed, and match immediately if they are used again. After that, the logic will begin a search of a 256 entry map. Running on a 125 ns clock, this would require 32 us max to scan the map. In order to cut this down, the map will be segmented into 4 64 object entries. The segment is chosen based on the two least significant bits of the Object Number. This will cut the maximum search time down to 8 us max or 4 us typical.

2) Two priorities of Bus access. The bus will normally be accessed in low priority mode, but programs may set a high priority flag for faster response.

3) Continuous Mode. If this flag is set, then the Bus Master will not release the bus after an access. A low priority access of this type will lose the bus to a HIGH priority access. The low priority continuous mode is provided to minimize the time spent searching maps for long bus transfers. In high priority mode, the bus will be entirely locked up. This will provide for autonomous operations of reading (multiple) word(s) within an Object, and setting some word(s). This is necessary for reliable hand shake operations in a multiprocessor environment.

4) Broadcast Mode. There may be some applications where separate copies of an object are maintained in more than one processor. In this case, one would like to wait for all Bus Slaves to find a match, and thus allow data to be written into the same object address of ALL copies of the object within the system. This mode does not make sense for reading.

5) Interrupt if accessed. The Map will have two bits which, if set, will cause an interrupt to be generated to the local processor if the object is written or read as appropriate.

7.2.4. How to use It

The Object Bus as described above allows for random access of remote (or local) data structures with appropriate memory protection. What one does with this capability essentially falls into 2 categories: data structures and message passing. A data structure could reside in one processor, but be accessible to all processors. Procedure calls can be implemented across the Object Bus. This is done by defining an Object in which one can indicate a request for a code segment to be executed, the parameters to be used, and the Object Number where the results may be returned.

In the case of the bit map display, a request to display a new line of text may be done by calling an object procedure with the appropriate parameters including a pointer back to the source text Object within the calling processor. At the same time, the bit-map display, which will reside in the display processors local memory will also have an Object Number assigned to it so that **any** processor can directly read and write the display bit-map.

7.3. Memory

The basic memory board for Magnolia will contain one row of 32 memory chips which will be accessible in byte, word, or double word modes. Configuring the memory this way will allow for full 32 bit microprocessors in the future. The 68000 microprocessor currently only uses byte and word accesses to the memory (double-word instructions cause 2 word accesses to be executed).

7.3.1. Choice of Memory Chip

There are currently a variety of chips available, and it is indeed tempting to want to provide for more than one type.

The standard memory part will be the 64K dynamic RAM chip. This will provide for 256K bytes of memory per board. These chips are currently available, although at a rather high price - $75 each. I would expect the price to come down substancially within 6 months to a year, and to $20 within 2 years.

The other option is to provide for current 16K memory chips. This would provide for 64K bytes of memory per board. This will probably be too small for most applications, but would make a good cost saving option until the price of 64K chips comes down.

7.3.2. Error Detection/Correction

There are two possible approaches in this area; byte parity with no correction, or error correction on the 32 bit words. The choice of implementation will depend on speed and complexity.

- Byte parity will require 4 extra bits per word, or 36 memory chips per board. This will simply provide error detection at a minimum cost, and will not require additional memory cycle time.

- Error correction requires more bits than simple parity. ECC on bytes requires 5 extra bits, words require 6, and double words require 7. Because of the byte operations of the 68000, the straight forward solution would require error correction on bytes. This is too costly, so the solution is to provide ECC on double words and turn all writes into Read/modify/write operations.

  The required steps for writing a byte would be as follows:

  - Read all 32 bits from the memory
  - Correct the data read if the ECC code indicates an error
  - Compute the new ECC with the new byte inserted
  - Write the entire 32 bits (or just the byte plus the ECC bits)

We will have to look carefully at memory timings, but I believe that the 65K RAM parts will have enough speed to allow us to do this.

7.3.3. Display Refresh Option

The basic memory board will also contain some optional additional logic intended to minimize the overhead of transferring bits from memory to the display. This will be implemented in the form of a second port to the memory whose address is on the standard local bus, but whose output is a FIFO and video shift register.

A memory access from a "display controller" will indicate a special video refresh mode access. In this case, the full 32 bits of data accessed will be loaded into a dedicated FIFO. The output of the FIFO will load a shift register whose output is a continuous bit stream of video.

If multiple memory boards are present, then the board select logic will always enable the board. Thus a single memory cycle will cause 32 bits from multiple boards to be read. This is done so that multiple memory boards can easily implement multiple bit-map planes (one plane per board) for color or grey scale displays.

The requirements for screen refresh (see section 7.6) will be 20 ns per bit, or a memory cycle every 640ns. The display controller driving this memory will have to be carefully designed so as to take only 3 (or 4 at the most) cycles per access so that the 68000 has sufficient access to the memory to make responsive changes to the memory.

7.4. Memory Management

Memory management is used here to indicate hardware assistance placed between the addresses generated by the 68000 and the addresses that are presented to the local memory. The amount of assistance needed is a function of the "type" of software being run in the 68000. The three levels that will be supported on Magnolia are discussed below.

7.4.1. None

In this case, there is no additional hardware. The addresses generated by the 68000 program are the addresses presented to local memory. This is appropriate for single task programs such as the display controller where all code is resident and available on demand at all times.

7.4.2. Base and Limit Registers.

This is required for a multi-task environment such as UNIX. It allows multiple "programs" to be dynamically relocated in real memory. This mechanism requires that all code for a given process be loaded in memory before running, so that no "page faults" will occur. The hardware required is very nominal, say 20 IC's.

7.4.3. Virtual Memory

Virtual memory is a capability that must, at some time, be addressed for Magnolia. The addressing capability of the 68000 makes very large programs possible. This large address space will never be implemented in real memory, hence some form of memory mapping must be provided.

Chosing the most appropriate implementation of virtual memory is fairly difficult. The basic concept is that of compromise - you can't have all the memory desired but with degraded performance you can still run the program. Further decisions of compromise must be made reguarding the complexity of hardware support. All out support for maximum performance during paging can involve extensive hardware. Keeping the implementation costs in balance with the rest of the system requires careful consideration of the overall design. Two possible approaches are discussed below.

- Instruction Faulting and Restart
  This is the "standard" approach. If the processor makes a reference to an address not in memory, then the

Virtual Memory hardware aborts the access and flags an error. Note that this is not necessarily between instructions, but at some intermediate point. The 68000 would then have to determine what new page is needed, where to put it, and generate the disk commands. When the needed page is brought into memory, the 68000 would have to re-start the aborted instruction and continue normally. This approach is almost impossible for the 68000 (as well as the Z8000 and 8086) since the processor is not designed to save, or report, internal state of an instruction.

- Hang during Fault
  This is an approach proposed by Forest Baskett (Stanford University) in which an 8 bit processor is included as part of the Virtual Memory hardware. This processor is used to service the page faults and generate the needed disk commands. The 68000 would be totally "hung" until the needed page is loaded after which the initial memory request is completed.

The first solution is a requirement in the multi-user large scale processor environment - the processor can service other users while one process page faults.

The second approach will work very effectively in the single user environment. The VM processor can be programmed to take care of low level page processing and maintenance of page tables. In most cases, the VM processor will have determined the next victim before a page fault occurs so that the disk can be activated immediately. The disk selected for use on Magnolia will allow typical page faults to be completed in 25-30 milliseconds. The relatively small number of tasks available for running means that it would be hard to make good use of the 68000 during page faults even if it were free to run.

The primary implication of a Virtual Memory that hangs the main processor is that all real time operations (such as timers and I/O) must be capable of running independently of the 68000. At the least, this means that all I/O must go through a DMA chip. The worst would be that another 8 bit processor would be required on the I/O board to act as a "smart" DMA controller for all the I/O.

7.5. I/O Capabilities

One board will be designed to implement as many of the I/O capabilities as we can justify for the system. A single board should be sufficient, although cable connections may be a limiting factor!.

Most requirements can be met through the use of 6800 peripheral controller chips. A DMA chip will be provided (wired for word transfers) to handle high speed I/O. A second DMA chip (wired for byte transfers), or an 8 bit processor, will be provided for low speed I/O.

The following list of capabilities represents some of the things that come to mind. They are ordered by decreasing importance.

7.5.1. Serial Interface

There will be two serial interfaces, one suitable for driving a local terminal, and the other suitable for connection to a host computer. These could be used in lieu of a local display processor, local disk, or network connections.

7.5.2. Hard Disk

A local high performance Winchester style disk is the most appropriate for Magnolia. This class of disk come in a small package (5" X 10" X 14"), is quiet, and holds around 20 - 30 megabytes of data. The chosen disk (from Micropolis) contains a formating processor internally and only requires a high speed parallel interface (8 bits wide). This interface should convert the 8 bit bus to the disk to a 16 bit 68000 interface.

7.5.3. Floppy Disk

With the addition of a floppy disk controller chip, we can have this option. There are a wide range of applications where a hard disk is not necessary (basic text editing for example) and there is a significant cost differential.

7.5.4. Network

Magnolia and the CRG network are made for each other! It is the high performance communication between processors that makes a community of personal machines work. It is the placing of high performance processors onto a network that make it really useful.

Initially, a parallel interface into the existing NIBB is the most appropriate connection. In the future it might be more convenient and cost effective to bring the NIBB software into Magnolia and simply provide a connection to the basic Network tap.

7.5.5. A/D converter

Four or more analog channels would be used for pointing device input, audio, etc.

7.5.6. D/A Output

Audio feedback for user response, games, music etc. would clearly be used.

7.6. Display

The display is the key ingredient which will permit the Magnolia system to provide qualitatively and quantitatively better performance over traditional time-shared systems. Magnolia will not be able to equal a VAX or 2020 for computational problems.* It will, however, be able to provide a user interface capability and responsiveness exceeding any of the time-shared systems currently in use.

The display that I propose to implement is based on a high resolution 1023 line monitor. Using a 17 inch display, and mounting the CRT horizontally, provides a useful screen resolution of 990 lines high by 1280 bits wide.

- The use of a bit-map display will allow the Magnolia processor to have a large measure of freedom over the images that a user might want to have displayed.

- The high resolution image will allow two text pages to be displayed side by side, plus some additional area above or below for menu and status information.

- The high resolution will decrease the objections jaggie lines raise in lower resolution systems.

- This screen resolution is well matched to the memory capacity that will be available through the use of 64K RAMs. The bit-map will consume 150K bytes of memory, leaving 100K bytes of additional memory for program code and data tables.

- The horizontal format will support graphic applications in the format which is most commonly accepted.

The Magnolia architecture of multiple processors will allow us to develop a system design where the functions necessary for screen manipulation are well organized and separated in the separate "display" processor. The system will, at the same time, provide flexibility for application programs to load some of their own "display routines" into the display processor, or simply access the "raw" bit-map as an Object.

7.7. Pointing Device

Interactive editing with a computer is limited by the speed with which the user can "point" to some place on the screen. By

* Unless augmented by special-purpose math processors.

"interactive editing", I mean the process of making small changes to a document that has already been entered. The ease with which the user can point will greatly influence the way one uses a system. For example, there are a great number of changes that I would like to make to this document in order to clarify, organize, or just plain improve the quality of writing. The point at which I quit making changes is determined by how hard it is to specify the changes, i.e. how good (or bad) the pointing mechanism is.

Magnolia should be an ideal opportunity to experiment with a variety of pointing devices. I would like people to be able to use and compare 4-function keys, track-balls, tablets, mice, joy-sticks, etc. In order to provide for these capabilities, both the hardware and software must have the appropriate provisions built in. The I/O devices mentioned above should make it easy for all the above devices. The operating system will need to separate out the operation of tracking the pointing device such that routines for any device can be easily substituted.

7.8. Programmable micro-processor

One of the most significant advancements in current computer architecture is the micro-coded processor. The philosophy of "programming" a machine's operation has lead to cleaner designs as well as more powerful instruction sets. A significant programming tool for achieving performance enhancement can be provided at a relatively small cost by making this micro-programming available to a running program.

While the 68000 is a microcoded processor, its internal operation is fixed by the manufacturer, and the microcode program is not alterable.

In order to extend the applications of Magnolia, some form of micro-programmable processor should be implemented in addition to the primary 68000 processor. The additional processor will be implemented as a "co-processor" which will run in parallel to the 68000 with interrupt and/or address flag communication.

This micro-programmable processor can be a reasonably simple design using 2901 bit slice parts or the new AMD29116 micro-processor. A minimal writable control store of 1K words would provide a useful system. The control store would be a "hardware memory" within the 68000 address space, and writable as an Object.

This option has immediate application for bit-map manipulation of the display, but would also be very useful for providing programmable microcode algorithms for LISP or High Level Language execution - for example.

7.9. "C" Compiler

The C compiler will be written by utilizing a UNIX program called PCC (for Portable C Compiler). This program provides the language syntactic evaluation, and requires only that the code generation for the specific end processor be implemented. Because of the similarity of instruction sets between the PDP 11 and Magnolia's 68000, it should be possible to "copy" much of the code generation implemented for the PDP 11/70 processor.

Since the C language is somewhat "typed", It would be advantageous to add the concept of an Object (accessed over the Object Bus) to the language so that the code generated is correct without explicit code being written by the programmer.

All code generated by the C compiler will be relocatable and re-entrant. Relocatable means that all addresses needed will be **relative** to either the program counter or some structure pointer. The code may be placed at any location within memory and be executed. Re-entrant means that all variables are stored in some data structure outside of the code being run so that no memory location within the code is ever modified during execution.

7.10. Linker/Loader

This is a standard program which will be needed to cause multiple "modules" (or files) from the C compiler to be combined and made available for loading and running. A mechanism will have to be provided to allow the linker to assign code modules to different processors, and to have them loaded appropriately.

7.11. UNIX Operating System

The purpose of implementing the UNIX operating system is to provide familiarity and compatibility with the time-shared system currently in use. The principal advantage will be in the ability to compile and run existing programs such as XED (line editor), FRED (screen editor), NROFF (document formater), etc. The continued use of Magnolia would result in new programs being written that would demonstrate the true capabilities of single user systems.

Most of the UNIX Shell commands, and a number of UNIX programs will be implemented. Simplifications and reduction of system tables will be made consistent with the elimination of multi-user capability. Despite its high performance, we must resist the urge to make Magnolia support more than one user at a time.

The initial implementation must be kept as straight forward as possible. One aspect of this is to keep the I/O, operating system, and user code in one processor. Only the display capabilities will be implemented in a separate processor. Future implementations should provide for code running in multiple processors, and might even make code placement dynamic at run time.

Magnolia's UNIX should make some use of the large display capability as soon as possible. The display driver should implement commands that allow the screen area to be partitioned into rectangular "windows." UNIX shell commands will be provided to specify the size and location of the windows, such that different "processes" can communicate with the user through different windows. An example of this capability would be initially used for document creation. One window would present the text editor, another window would present the formated output of MS (alias NROFF), while a third could present interactions with SPELL.

Modifications of the standard UNIX can also prove advantageous in the implementation of file name look up and access by the operating system. The UNIX file system implements a hierarchy of directories for locating files. On Magnolia, this hierarchy should extend beyond the local disk and automatically use the Network lo locate remote file systems where the specified file may reside. A high speed local Network with a high performance "file server" could make a Magnolia system fully functional with no disk at all.

8. Cost Estimate

8.1. Component Cost

This cost estimate is based on the construction of one Magnolia system, using current parts cost, and implementing the logic on high quality commercial wire-wrap boards, chassis, and power supplies.

A second set of numbers, in parentheses, are supplied as a cost estimate for systems in the future. This set of numbers assumes multiple systems, say 10 or more, to be build 1 1/2 to 2 years from now on ECBs. This is obviously a rough estimate since it encorporates a number of ICs that are very new in the market.

8.1.1. Processor Board

68000 processor chip	$150	($100)
Object Bus interface	$100	($75)
Misc ICs	$50	($50)
	Total= $300	($225)

8.1.2. Input/Output Board

RS232 (2)	$20	($20)
Hard Disk ICs	$40	($40)
Floppy Disk	$40	($40)
Network	$40	($40)
A/D, D/A	$40	($20)
Misc ICs	$20	($20)
	Total= $200	($180)

8.1.3. Memory - 256K Board

RAMs 40 @$75 ea.	$3000 ($800)	
Support ICs	$50 ($50)	
		Total=$3050 ($850)

8.1.4. Memory - 64K Board

RAMs 40 @$8 ea.	$320 ($160)	
Support ICs	$50 ($50)	
		Total= $370 ($210)

8.1.5. Memory Management - Base/Limit registers

Registers	$12 ($12)	
Misc ICs	$20 ($20)	
		Total= $32 ($32)

8.1.6. Memory Management - Virtual Memory

RAMs 8 @$30 ea.	$240 ($120)	
Processor (6502 +PROM)	$30 ($20)	
Misc ICs	$40 ($30)	
		Total= $310 ($170)

8.1.7. Display Interface

Misc ICs	$40 ($40)	
		Total= $40 ($40)

8.1.8. Programmable Micro-Processor

2901A 4 @ $7 ea.	$28 ($20)	
Memory 8 @ $7 ea	$56 ($40)	
Misc ICs	$50 ($50)	
		Total= $134 ($110)

8.2. System Cost

Processor #1 (128K memory)		
Processor	$300 ($225)	
I/O board	$180 ($160)	
Memory Management	$32 ($170)	
Memory	$370 ($210)	
Memory	$370 ($210)	
		Total=$1252 ($975)
Processor #2 (256K memory)		
Processor	$300 ($225)	
Memory	$3050 ($850)	
Micro-Processor	- ($110)	
Display Controller	$40 ($40)	
		Total=$3390 ($1225)
Wire-Wrap boards 8 @ $300 ea.		$2400 ($400)
Card cage		$300 ($100)
Power Supplies		$500 ($300)
Winchest Disk		$3700 ($3000)
Display Monitor		$680 ($500)
Keyboard		$100 ($100)

TOTAL= $12,322 ($6600)

8.3. Labor for implementation

8.3.1. Hardware

Tasks necessary to implement the hardware are as follows:

Processor	2 months
Memory	3 months (4 with error correction)
I/O	5 months
Display	1 month
Construction	3 months (technician)
Checkout	4 months
Total	18 months

The most appropriate number of people to implement the above pieces are 3. One for the processor and display, one for the memory, and one for I/O controllers. The hardware could be implemented in 6-8 months with 3 engineers working full time. The time to implementation would extend proportionally if people work on a partial schedule. The system would take say 9-12 months working 1/2 time.

8.3.2. Software

Tasks necessary to implement the software are as follows:

C compiler	4 months
Linker/Loader	2 months
UNIX	8 months
Display routines	3 months
Total	17 months

The most appropriate number of people to implement the above pieces are 2 or 3. One could implement the compiler and linker, while 1 or 2 would work on UNIX. I do not anticipate that the job would be much quicker with two working on UNIX, but the system implemented might be noticeable improved. The task of implementing UNIX is large enough that a full time commitment is pretty much necessary, so the software would take 8-10 months to implement.

Allowing 2 months for combined debugging of hardware/software problems, a running system could be achieved in the 12-18 months time frame. The resulting system would have a high resolution bit-map display available to multitask UNIX system running already existing UNIX programs.

9. References

C.P. Thacker, E.M. McCreight, B.W. Lampson, R.F. Sproull, and D.R. Boggs: "Alto: A personal computer" CSL-79-11, XEROX Palo Alto Research Center, 1979.

Forest Baskett: "Pascal and Virtual Memory in a Z8000 or MC68000 based Design Station," COMPCON 80, Digest of Papers, pp 456-459, IEEE Computer Society, Feb. 25, 1980.

"Proposal for a Joint Effort in Personal Scientific Computing," Carnegie-Mellon University, Dept. of Computer Science, Aug 23, 1979

"Research in Integrated Distributed Computing," University of Washington, Dept. of Computer Science, Oct. 1979.

"PERQ," Three Rivers Computer Corp., 160 N. Craig St., Pittsburg, Pa. 15213.