Cub Workstation Design Goals

Provide a low cost "front end" for the OIS market

1. Provide an imaginal display

2. Provide Ethernet connection

3. Provide personal services (text/graphics)

4. Not to provide all services in the terminal (reach through)

cub-goals.bravo

A Discussion of Cub Architecture and A Discussion of Cub UMC estimates

May 23, 1980 R. Belleville filed on cub-may80.bravo

What is a Cub?

Cub is a new workstation under development in SDD. The purpose of the design is to provide a truly Ethernet compatible, low-cost workstation for the mainstream of the office automation market. Cub users have at their fingertips, not only a powerful personal computer, but also network access to servers and resources on the Ethernet.

Cub provides an 8086 based processor running at 4.5 MHz. This is a 16 bit microprocessor with a full range of instructions and addressing modes. Main memory consists of 262,144 bytes (8 bits data plus 1 bit parity) of dynamic random access memory. Single bit memory errors are detected by a parity system. A bank of up to 16,384 bytes of read only memory are also provided for "bootstrap" program storage. Main memory is refreshed automatically.

A full page, bitmap display is also part of the Cub workstation. An image 606 bits wide by 808 bits high is displayed on a 15 inch video monitor. The frame buffer storing these 489,648 bits is also implemented with dynamic ram and does not use any processor main memory. In all, Cub has 320K bytes of ram.

The display controller also includes a 16 x 16 bit cursor which is "ored" into the image at the x and y coordinate provided by software. Separate storage is provided for the cursor bitmap in a 16 word ram associated with the display controller.

Input/Output controllers are provided for the keyboard, mouse, and a single channel RS-232 serial interface. The serial interface can operate at speeds from 300 to 9600 baud. An 8 channel, vectored interrupt system is also included with the processor. Interrupt based service is provided for the serial interface, keyboard, real-time clock, network interface, and main memory parity error detection.

An Ethernet controller is also an integral part of Cub. The interface can send or receive packets of data, to or from main memory at speeds from 3 to 10 Mbps. Transfers are under direct memory access control, and do not require the processor during data transfer.

Cub has little "built in" functionality. All application software is "downloaded" from a server via the Ethernet. Data files created by the user are communicated to servers for storage and retreival via the Ethernet. Cub provides for no fixed or removable storage at the workstation. This approach allows the use of cost effective mass storage at the server, and frees the workstation from the cost, noise, and reliability problems of rotating storage.

Details of the organization of Cub are shown in the block diagrams below.

How can Cub be useful without a disk?

By itself, a Cub is useless. It depends on the Ethernet to communicate with servers located on the network.

The first dependency is for program loading. After powerup, Cub sends a message on the Ethernet which asks for basic system code to be loaded into memory. Cub cooperates with a server to obtain a program which I will call the user kernal. User kernal may look to the user like the Star system or perhaps like existing word processing systems like the 860; however, the code which performs the function of these systems may, or may not, actually be executed by Cub.

Cub is designed to provide "personal interactive" service to the user. For example, Cub can handle the creation and editing of text and graphics locally with little or no load on the network or its

servers. On the other hand, Cub would not run an accounts receivable program. That would happen on a server connected to the network. A Cub user might initiate the running of the receivables program through the user kernal. New data for the receivables data base might be created on Cub and the results might be inspected on the display.

Functions actually executed in Cub are "downloaded" from the network.

The second dependency is file management. The concept of a file server is simple. A file server is a computer with a relatively large, "permanent" memory (lots of disk for example) which is programmed to store and retrieve collections of bits (files or data sets in the IBM terminology) from commands received from other computers on the network (workstations and servers alike). A file server is a library and the program that runs it is a lot like a librarian. By itself a file server is not very interesting, but combined with the Ethernet it becomes a central tool in managing information which is, you will recall, what this whole thing is about.

Cub doesn't have any fixed or removable storage because, in the long run, we won't be able to manage little caches of information locked away in cabinets and on non-removable disk drives. Take the example of a large U.S. corporation with a system consisting of 1200 workstation located in 15 cities. A worker traveling to a distant city will want to use the system and access his documents. If these documents are stored as a matter of course on "his" workstation how will he access them? Will he have to leave his machine running or will he have to call someone to place them online (as I have had to do several times)? If the data is on removable media (like floppies) will he have to carry them with him?

The databases manipulated by the workstation should be shared in common servers.

The third dependency is for communication. The Cub may be used to create or read mail but the distribution would be left to the network (and probably file servers and mail servers).

The fourth dependency is for services outside the scope of the workstation. This is the central reason for the network connection of servers and workstations. Actual applications will require the user to perform actions which are outside the scope of any workstion that can be considered "personal."

Consider the Xerox Corporation's mailing list as one huge database. There are surely lists of the stockholders, employees, customers, potential costomers, etc. I know that the list is in a computer somewhere because of all the mail I get. In the OIS world this database would be a resource held in common by the company and put to countless uses by many people throughout the organization. Would we ever expect any workstation in the system to encompass and operate on that database? Very unlikely. From the workstation however, one would expect to submit a request to the machine that managed this database for a list of all customers in greater Boston. If this user was appropiately authorized, he would expect to be able to read the results at his terminal or print the result on a print server near his office.

What about Dandelion, doesn't it violate your OIS model?

Not in the short term for workstations. Nor ever for server applications.

Dandelion is a powerful computer in a small package. It is perhaps 7 times the power of Cub. Dandelion is capable of executing the full range of Mesa/Pilot software and can be extended to several important server applications. A machine of this class is clearly needed for server applications. But it is needed as a workstation too.

As the OIS environment begins, most customers will be "testing the water". They will want to buy 1 or 2 machines at first to see how they can use OIS in their applications. So each workstation will have to do some the the function that would be server functions later on. As these company find cost effective jobs, their workstations can be converted to servers and Cubs added to act as

workstations.

The concept you propose is appealing, why don't we get the cheapest terminal we can and hook a whole mess of them to Dandelion?

It takes too many cycles of the host machine to truly provide interactive service to the user. It is true that, simple, fixed font editors work relatively well at about the 10 to the PDP 10 ratio. Full multi-font work with graphics is a different story. The bottom line is that you really need a relatively powerful, full page display to do real applications and you need a snappy processor to handle both the display and the network.

There is a reliability issue too. If low power terminals are connected to a central computer and the central machine fails, then all the terminals are down. So one has to add a good bit of cost to add redundent machines or make the cluster machine more robust. Cub has the same property if there is only one machine implementing the servers; however, it is likely that systems implemented with a moderate to large number of Cubs will have more than one server processor and thus the redundency is built in to the topology of the network.

Ethernet allows us to bring together the best features of time sharing (getting personal service) and batch (sharing resources through common servers).

How would Cub be packaged?

The package would look a lot like an Alto display and keyboard. The electronics would be housed in the base of the display. Cub is convection cooled so no fans are needed and the workstation would be essentially silent.

Is Cub powerful enough?

Initial performance measurements on the feasibility model show that at least one crucial parameter, placing characters onto the display, Cub is as fast as an Alto. The 8086 processor running at 4.48 MHz is perhaps faster than an Alto for normal program execution. If the display management is fast enough without the addition of special hardware (as is seems to be) and if the execution of application code is as fast or faster than an Alto, then we have Alto power in a small, low cost package.

A Cub - Function and Performance Summary page gives more details.

Well then, maybe Cub is too powerful and we could save some cost by cutting down somewhere?

We could leave out the RS232, that really doesn't do anything in the baseline terminal. But it only costs about $6 and might be useful to control a local printer, optical scanner, or other local peripheral.

The big cost item is the memory as we shall see in the umc discussion. Dynamic ram represents 56% of the cost of the electronic portion of Cub. Main memory in the feasibility model is 256K bytes or 128K words. This size allows the design and implementation of powerful user software without the need for complex memory management schemes. Although I cannot produce an air tight argument to set the size of main memory, 256K feels right. In the UMC section, I have shown the impact of halving the memory size to 128K byte (64K words).

Cub isn't even close to being a "Princops" machine, can it run Mesa?

This is a tricky area and one where more analysis and experiment is needed. Let's look at some of the aspects of the problem and some of the solutions.

First, the basic principal of the Cub/Ethernet/Server architecture is that the workstaion provides

personal service tools to the user, not the major applications. So once the basic text/graphics/imaginal tools are written for Cub the workstation will be able to work with many kinds of server based applications. For example, a Teletype has been used for applications far beyond the transmissions of telegrams. With Cub the result is much the same, once one can create and edit text and graphics and interact directly with servers via Ethernet many different applications would use the same workstation software.

If nothing else, this split between the tasks of workstations and servers may ease some of the constraints on the methodology used to create Cub software.

Second, in the Alto/Dolphin/Dandelion/etc., we have always resorted to microcode implementations for the low level, performance critical, sections of the system software. With the 8086 the equivalent mechanism is native machine code written in assembly language. Microcode is considerably more difficult to write and maintain than assembly language code. In Dolphin for example there are more than 10,000 lines of microcode in the system and the diagnostics. Creating and maintaining, say, 50,000 lines of 8086 assembler is managable. (not a lot of fun, but do-able)

The "C" language has been used rather successfully for the implementation of systems on the PDP 11. In fact almost all of the the UNIX operating sytem and its utilities are written in C. This may be an alternative to assembly language. We are looking into this now and will know more in the comming months.

Third, we could run Mesa in one of two basic ways. We could write a program in 8086 native code that emulated the Princops machine, or we could modify the compiler to produce native 8086 code instead of the Mesa byte stream. At this point we cannot rule out either approach because it is possible that either approach might provide satisfactory performance. Much more study is needed here.

How much does it cost?

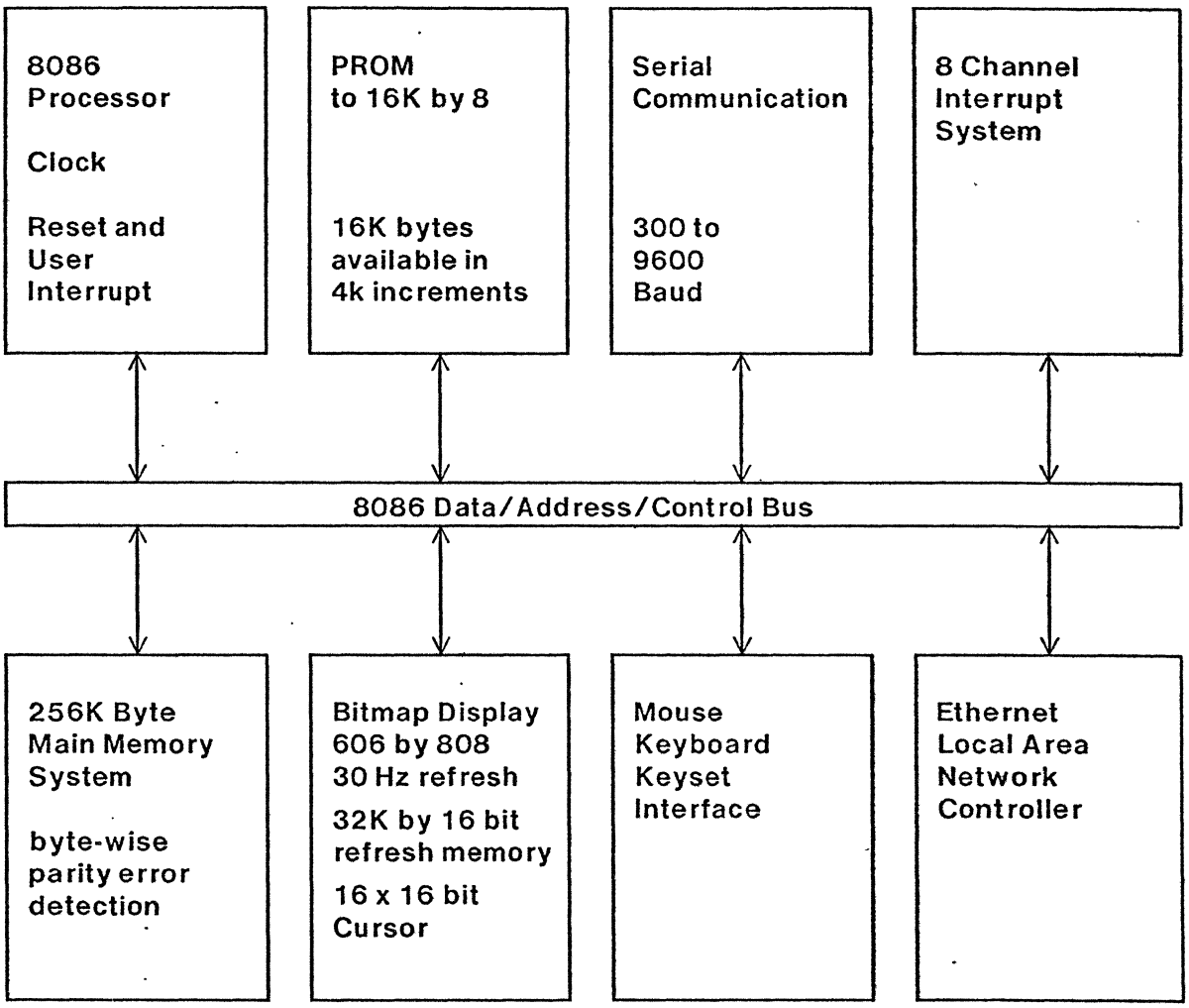
A product development UMC estimate is included in this document. There are problems with the numbers I have used for burden, etc; however, the basic cost picture can be seen. Remember that the feasibility model is actually running code and displaying pictures on the monitor, so the parts counts and types are not estimates but based on actual fact.

All three cases 0, 1, and 2 are interesting and different. Case 0 is the feasibility model implemented on two PWBAs. Case 1 cuts the memory in half and uses the Intel Ethernet controller to achieve, I believe, a very attractive cost which could be achieved with parts available today. Case 2 is, of course, the one to shoot for in a 1982 or 1983 timeframe. The price of 64K dynamic rams is the only unknown risk.

Package, power, keyboard, mouse, monitor, and misc. are between 600 and 700 dollars more.

In this kind of design, cost is king and a good deal of the development expense will be in the minimumization of cost not only in the basic UMC but also in the PLC.

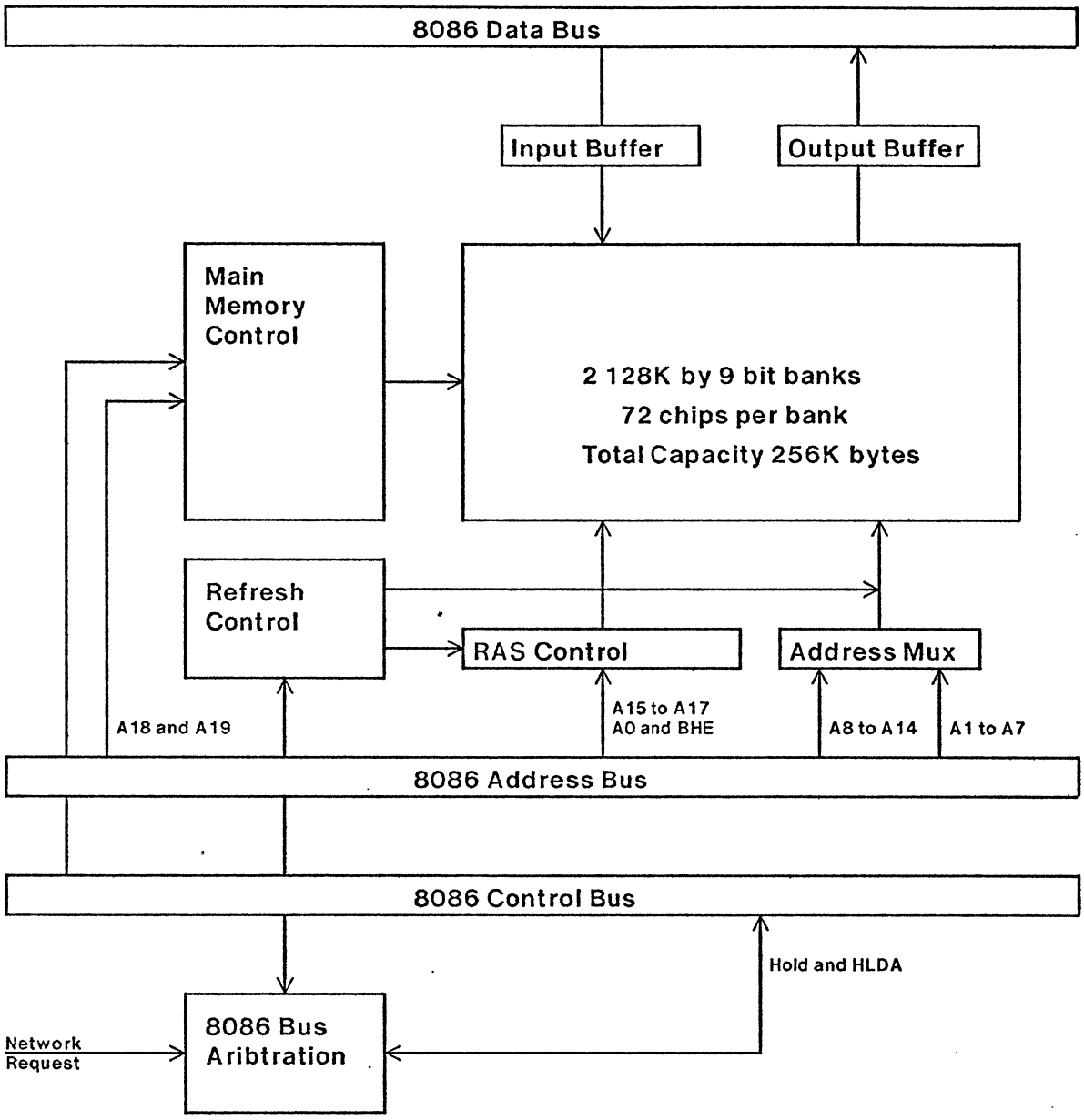
Block Diagram of the CUB Workstation



cub-cp.block
RLB 21 May 1980

Block Diagram of the Main Memory System

cub-mc.block



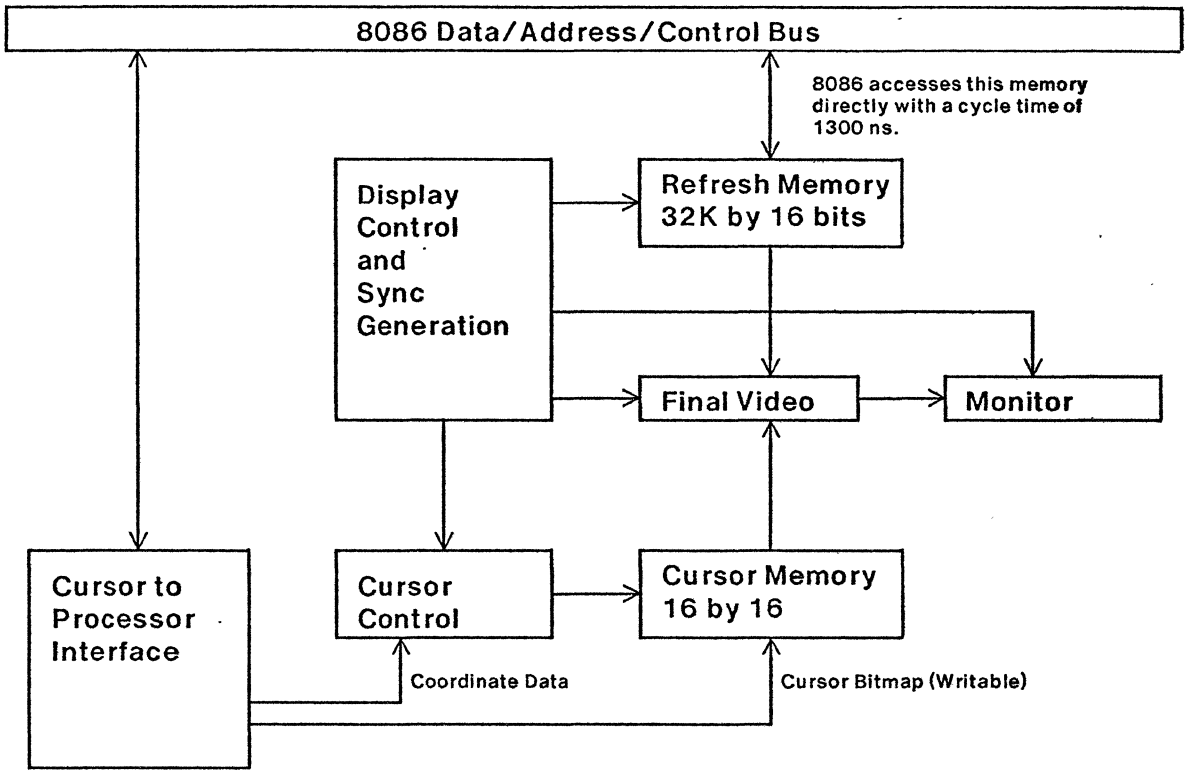
Physical Address Space Allocation

There are 16 64K byte pages

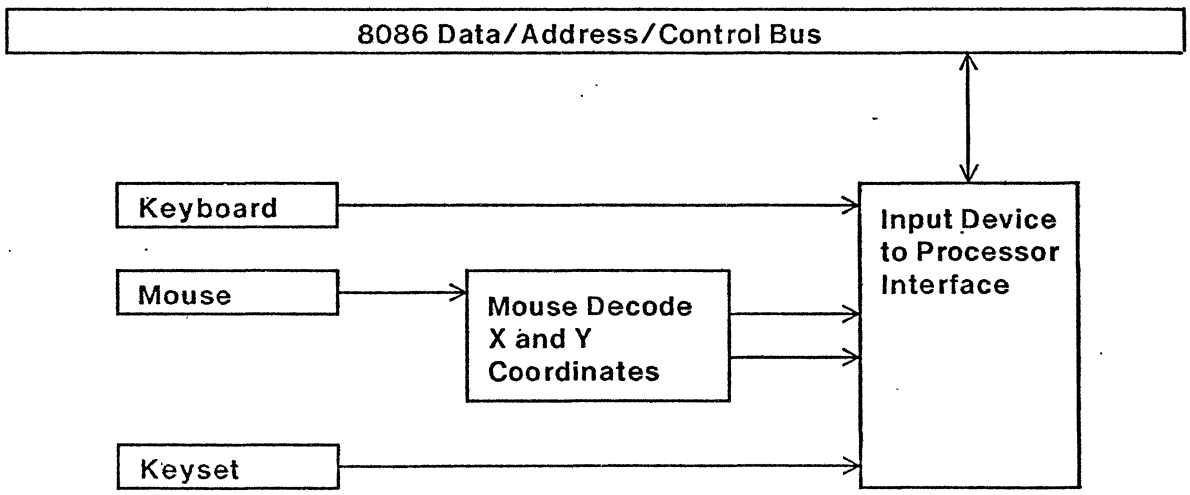
0	Implemented	4	for expansion	8	for expansion	C	for expansion
1	Implemented	5	for expansion	9	for expansion	D	for expansion
2	Implemented	6	for expansion	A	for expansion	E	Display Refresh
3	Implemented	7	for expansion	B	for expansion	F	PROM

Block Diagram of the Display and Cursor

cub-dsp.block



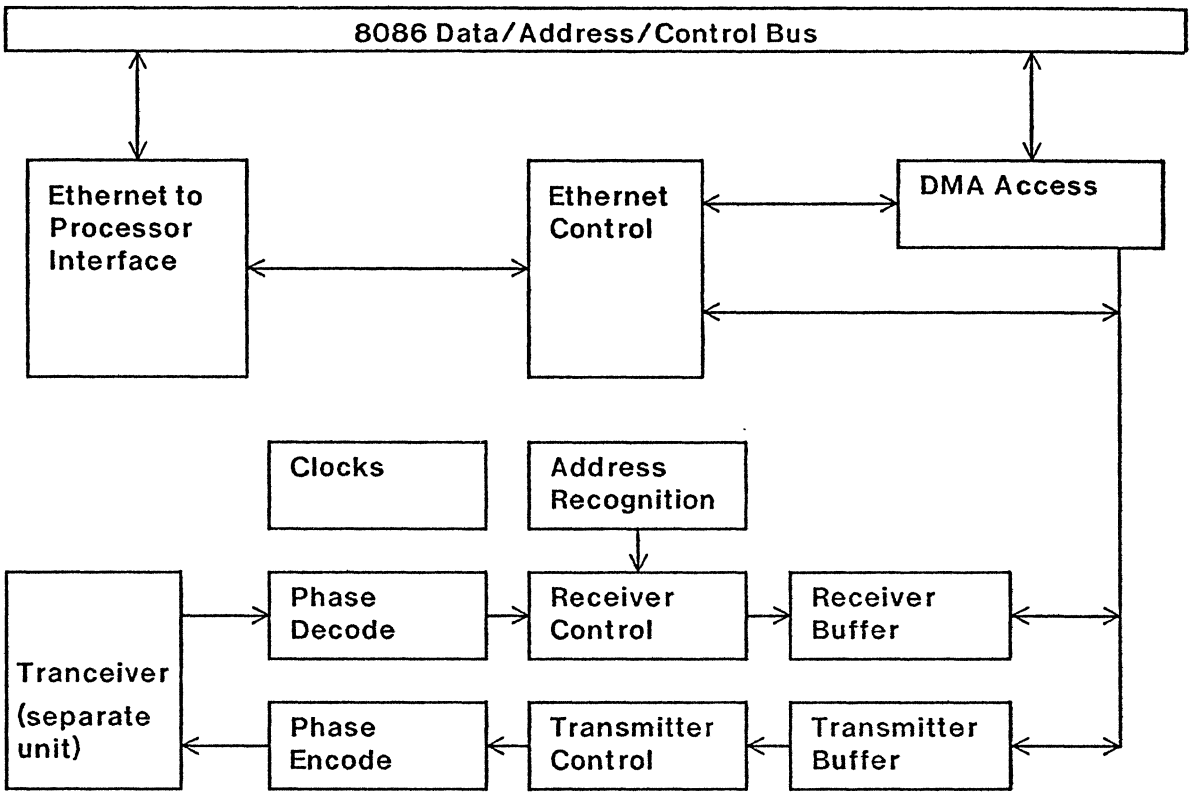
Block Diagram of the Keyboard, Mouse, and Keyset Interface



A keyset is a 5 paddle input device used with the left hand.
It is not necessary for current User Interfaces.

Block Diagram of the Ethernet Controller

cub-en.block
RLB 21 May 1980



Cub - Function and Performance Summary

Cub-f-p.block
RLB 21 May 1980

Microprocessor

8086 4.48 MHz 223 ns. clock
example timing reg. to reg move 446 ns.
local to reg 4 us (approx)
add reg. to reg 669 ns.
within segment call 2.5 us.
16K EPROM available in 4K increments (bytes)
Serial Communication RS232 300 to 9600 baud
8 interrupts

Main Memory

256K bytes with byte wise parity error detection
cycle time approx 600 ns. and the 8086 runs without wait states
available for DMA transfer from the network controller
extendable to 896K bytes
refresh of the dynamic RAMs is fully automatic and requires no processor intervention

Display

Bitmap 808 x 606 (height x width) bits active video
30 Hz refresh rate P40 phosphor
refresh memory (32K by 16) independent of the main memory
16 x 16 cursor controlled completely by hardware, no 8086 intervension
refresh memory available on the 8086 bus directly with a 1.7 us. cycle time
cursor coordinates and writable cursor bitmap on 8086 I/O bus
875 line sync generation

User Input

Keyboard, Pointer

Network

10Mbit Ethernet
Data input and output transfers are to main memory using DMA

Performance Measurements (Taken on operational feasibility model.)

Time to fill the screen with 10 point variable pitch, multifont, text - 1.5 seconds.
(Actual figures: 12 pt 1241 char/sec, 10 pt 1655 c/s, 8 pt 1909 c/s)
(This is about the same speed as an Alto.)

Cub PDUMC Estimate | 1982 Technology | cub-pdumc.block
Estimate Date: 21 May 1980 | 1980 Economics | RLB

Assumptions:

Electronic components only
Parts type and count taken from operational feasibility model
except Ethernet controller data estimated from completed but untested design
15% material burden
$37 per hour labor and labor overhead
1 hr per module direct labor assumed
parts costs taken from Dandelion estimates where possible

Baseline Cub:

Configuration:

8086 w/ 8K Eprom, 256K main memory w/ parity, 64K display, Ethernet, mouse, keyboard, and RS232 (memory stated in bytes, parity on main only)

Implementation:

16K dynamic rams, two modules, commercial LSI only, multi-voltage

Cost Breakdown:

Integrated Circuits except as below (SSI, MSI, common LSI)		
CPU/Memory control/RS232/keyboard&mouse controller/misc.		$ 57
Display controller		52
Ethernet		70
CPU LSI Group (8086, 8284, 8259A)		37
EPROM (8K bytes implemented with 4 2716)		46
Display ram (32 4116 @ $3.75)		120
Main Memory ram (144 4116 @3.75)		540
Decoupling capacitors (250 @ $0.15)		38
PWBA (6 layer @ $112)		224
	Subtotal	1184
Material burden (@ 15%)		178
2 Hours labor (@ $37)		74
	Total Case 0	1436

Variations from baseline:

128K main (72 vs. 144 and burden)		-310
LSI Ethernet controller		-58
4K EPROM		-26
	Total Case 1	1042

Major variation from baseline:

128K main (18 64K parts @ $23)	(not interesting at this level)	414
@ $11 (estimated 1983 minimum)		198
Single PWBA		112
Savings from custom LSI (guess)		-50
LSI Ethernet controller		-50
4K EPROM		-23
Savings in decoupling capacitors		-20
Costs from baseline above		420
	Subtotal w/ 1983 ram	587
Material burden (@ 15%)		88
1 Hour labor (@ $37)		37
	Total Case 2	712