CC: TO URQUHART

Rick has good points here; what are we going to do?

===== Lisa 1.75 ROM ==============================

To: Ron Johnston, Marian Catelain, Ken Okin, Gary Marten
From: Rick Meyers
Date: May 26, 1983
Subject: Lisa 1.75 ROM Organization

The Lisa 1.75 is expected to have 128K bytes of ROM and 128K bytes of RAM on the CPU board. Additional RAM will range from none (in a servant) to 768K (full memory board) and upward (given extra memory boards). In a 512K system (assuming a 512K memory board, 128K CPU ROM, 128K CPU RAM), the CPU ROM and RAM account for one third of the memory in the entire system.

It's extremely important that we use this CPU memory wisely. Careful coordination between POS hardware and software group will be required. This memo outlines some of the planning that should begin immediately.

Memory Contents and Budgets

We need to carefully select both ROM and RAM contents, and budget their use of memory. Possible ROM contents include:

- diagnostics (CPU, memory, peripherals)
- boot code (initialization, load from disk)
- hardware interface (cursor, mouse, keyboards, speaker, timers, clock, power, etc.)
- drivers (built in disk, twiggy, profile, priam, RS-232, printers)
- partial file system (for booting, keyboard tables, preferences, etc.)
- miniature dubugger
- fonts
- QuickDraw

Possible RAM contents include:

- primary screen (32K)
- optional alternate screen (32K)
- interrupt and trap vectors (defaults, soft vectors, etc.)
- hardware interface global data (keyboard mapping, keyboard queue, timers, etc.)
- QuickDraw.global data (screen size, address, etc.)
- disk buffers, caches
- debugger data area (registers, breakpoints, etc.)

Architecture

The dependencies between the possible ROM contents are complex. For example:

- diagnostics need hardware interface, fonts, QuickDraw
- boot code needs hardware interface, drivers, file system
- hardware interface needs drivers and file system (for keyboard definitions)
- debugger needs hardware interface, but hardware interface must be debugged
- QuickDraw needs hardware interface
- etc.

There is no currently no linear ordering of the ROM software components which expresses the dependencies. This makes a layered organization difficult. Unfortunately, some ordering is required, if only for initialization. We need to break free from our current organization (or lack thereof), shuffle the software around, and define easily understood relationships between the ROM components.

Calling Conventions

Calling conventions must satisfy assembly language, Pascal, etc.
How do we get Pascal libraries (QuickDraw, HWInt, etc.) to point to ROM?
If a ROM routine needs to be replaced, how do we override it in RAM?
Certain ROM routines will need domain 0 and/or priviledged mode. Mechanism?
Bad addresses as parameters to the ROM will cause disaster. How is this prevented?
Default interrupt and trap handlers will be in ROM. How are the defaults overridden?

Definitions and Interfaces

Documentation is an important part of the job. We should begin immediately assembling a notebook which contains definitions and interfaces for ROM routines. The notebook should also include the ROM and RAM budgets in detail, and the RAM memory layout.