THE MACINTOSH PROJECT

DOCUMENT 0 VERSION 11

TITLE: CATALOG TO MACINTOSH DOCUMENTS

AUTHOR: JEF RASKIN

DATE: 28 Sep-13 Feb 80

M 0.11 CATALOG*
A chronologically arranged annotated listing of all Macintosh documents. The very document that you are reading. If there is no asterisk after a document title, that document is either obsolete or is especially technical.

M 1.4 INTRODUCTION AND PRELIMINARY CONVENTIONS
The conventions for documents, their distribution and cataloging.

M 2.9 OVERVIEW OF PRELIMINARY AREAS OF CONCERN*
A list, slightly annotated, of the various questions that must be answered in designing Macintosh. It is rather comprehensive.

M 3.5 THE APPLE COMPUTER NETWORK*
Justification of and preliminary thoughts on a network. An appendix lists some names and addresses of networks.

M 4.1 THOUGHTS ON ANNIE
An old memo, from May '79, with some early thoughts on what was to become the Macintosh project. Eccentric use of English.

M 5.1 PRELIMINARY COST INVESTIGATION*
A brief rundown on the cost of the major electonic and mechanical components of Macintosh. The $500 selling price is shown to be a difficult mark to reach.

M 6.2 GENERAL CRITERIA
An expansion of the general criteria listed in M 2.8, defining the major goals of the project.

M 7.7 A MODEL OF MEMORY VS DISK CHOICES
A description of the design of a mathematical model. This is the documentation for M 9 and M 10.

M 8.1 PERSONAL AIMS
My aims in doing the Macintosh project.

M 9.6 PASCAL MODEL OF MEMORY VS DISK CHOICES
A non-interactive program that sweeps through various document and memory sizes. Superceded by M 10.

M 10.3 INTERACTIVE PASCAL MODEL OF MEMORY VS DISK CHOICES
An interactive model that allows you to easily vary parameters.

M 11.0 SUMMARY OF OCTOBER 10
A few of the main points of the project as of October 10, prepared for a meeting with Whitney, Carlson, Jobs, Markkula, Holt, Scott Roybal and Raskin.

M 12.1 CONCERNS ABOUT USING THE TELEPHONE WITH PERSONAL COMPUTERS*
An article written for magazine publication on the probable difficulties we might encounter working with Ma Bell.

M 13.1 IMPORTANT POINTS ABOUT MACINTOSH
A one page summary of the summary of October 10, prepared for the meeting of 12 October.

M 14.10 THE APPLE CALCULATOR LANGUAGE*
This is an extensive document, not complete as of this version of the catalog (22 October 79) which contains a primer of a good portion of the language, the BNF and other technical considerations for those portions described, and some of the justification for the language. (Read M16 before reading this document.)

M 15.0 MASS STORAGE PRINTER/FACSIMILE DEVICE*
A description and discussion of a low cost device, based on present thermal or electrostatic discharge printer technology, that would provide printing, data and program storage and dissemination, facsimile transmission, and digitizing abilities to Macintosh.

M 16.0 AN INTRODUCTION TO THE APPLE LANGUAGE FOR CALCULATOR USERS*
An indication of how a primer for the language of M 14 might be written.

M 17.0 REPORT ON THE HP 41C AND SHARP 5100 CALCULATORS
What we can learn from them that helps in the design of Macintosh.

M 18.0 ON THE PROBLEM OF DELIMITING STRINGS IN PROGRAMS
A justification of the string delimiting mechanism used in the Apple language described in M14.

M 19.2 THE MACINTOSH EDITOR*
The initial design of a very user-oriented, fast editor.

M 20.0 THE MACINTOSH DISPLAY*
Descriptions and some simulations of the proposed display.

M 21.0 THE ON-LINE TEXT EDITING SYSTEM*
A description of the SRI text editing system developed in the late 1960's by Englebart. Some very interesting ideas.

M 22.1 HOW CAN WE MAKE COMPUTERS TRULY PERSONAL?*
A guest editorial Raskin wrote for a magazine--not sent out as being possibly to proprietary.

M 23.0 JANUARY 1980 OVERALL SUMMARY*
A summary of the present status of the design and the project. This

supercedes all previous reports and summaries.

M 24.1 PASCAL MACINTOSH FONT GENERATOR
The program that generates the proportional font that will probably be used on Macintosh.

THE MACINTOSH PROJECT

SELECTED PAPERS

14 FEBRUARY 1980

The photographs reproduced below represent a very preliminary mock-up of the proposed Macintosh computer. The screen size is accurately presented, however it is not known if a dual disk drive as shown will be available. The case design in no way represents the final appearance of the proposed computer, but was created to help give a feel for the approximate size of such a machine. If a printer is included, it will probably by 8.5 inches in width rather than the 5 inches shown in the mock-up.

It is recommended that these documents be read in the following order:

1. M 0, the table of contents to the collection

2. M 24, the summary of January 1980

3. M 2, the list of areas of concern

and then, via the table of contents, any further items of interest.

THE MACINTOSH PROJECT

DOCUMENT 1 VERSION 4

TITLE: INTRODUCTION AND PRELIMINARY CONVENTIONS

AUTHOR: JEF RASKIN

DATE: 11 Sep 79

1. CHANGE OF NAME

To avoid using only female names on projects, it has been suggested that each major project be assigned the name of a variety of apple. To begin with, what once was the "Annie" project is now called the "Macintosh" project.

2. CATALOGING TECHNIQUE FOR MACINTOSH DOCUMENTS

As each document pertaining to Macintosh is written, it will be given a serial number. A catalog will be kept on a diskette in my files. I therefore request that any document pertinent to the Macintosh project not be released until it has been cataloged. This will permit any interested party to know what documents have been written, and will allow us to obtain and use these documents when necessary. The catalog will also be available in printed form upon request.

3. INTERNAL DESIGN OF DOCUMENTS

It would be appreciated if each item in each document be numbered, so that we may easily refer to it in other documents, for example:

Document 4, Version 3, Item 6.4.

The heading for each major numbered section will be in all caps, with subsections having the first letter of the first word capitalized.

4. USE OF EDITOR FOR DOCUMENTS

For improved communication, documents should, whenever possible, be prepared on the Pascal Editor, with appropriate control characters for Colin's formatting program. A typical heading for each document might be:

```
.fo ' 'M1.4 Page %

DOCUMENT *** VERSION ***

THE MACINTOSH PROJECT

TITLE: ***

AUTHOR: ***

DATE: ***
```

This will facilitate building a library of Macintosh documents, and easily allow modification and updating of these documents.

5. The file name of each document shall be of the form Mx.y.text where x is the document number and y is the version number. The catalog is M0.x.text.

6. The responsibility for maintaining the catalog shall initially be Jef's, but may be assigned to another member of the Macintosh team when appropriate.

7. Distribution lists and dates by which comments must be received shall be part of the cover letter, and not part of the document.

THE MACINTOSH PROJECT

DOCUMENT 2 VERSION 9

TITLE: OVERVIEW OF PRELIMINARY AREAS OF CONCERN

AUTHOR: JEF RASKIN

DATE: 11 Sep 79

1. MARKETING AND AUDIENCE
  1.1 Markets
    1.1.1 Home
    1.1.2 Business
    1.1.3 Scientific
    1.1.4 Industrial
    1.1.5 International
    1.1.6 Educational
    1.1.7 Hobby
  1.2 Contributions this product will make
    1.2.1 This may be the first (unless something else comes along) portable computer. It is a personal, not a home computer. If this point of view is adopted, then applications, such as home security, where the computer is tied to a physical location, can be eliminated.
    1.2.2 This should be a completely self-teaching system.
    1.2.3 Price vs. performance breakthrough.
  1.3 Overall strategy
    1.3.1 vis a vis competitors
    1.3.2 vis a vis other Apple products
    1.3.3 in view of unprecedented large numbers
  1.4 Sales goals
    1.4.1 Number of machines over time
    1.4.2 Dollar figures
  1.5 Major risks
  1.7 General criteria
    1.7.1 Reliability (works correctly, unbombable)
    1.7.2 Serviceablitity (long MTBF)
    1.7.3 Produceability (designed for the production line)
    1.7.4 Price low, but performance high
    1.7.5 External esthetics excellent (as is our custom)
    1.7.6 Integrity (sufficient testing of all components incl. manuals)
    1.7.7 Maintainability (short time to repair)
      Possibility of self-diagnostic programs; possibility of diagnostic through network.
    1.7.8 Documentable (easy to write manuals due to good design, esp. of software)
    1.7.9 Expandable (hardware should not limit range of applications unnecessarily)
    1.7.10 Learnability (co-ordinated hardware/software/manual design with this constraint in mind at all times)
    1.7.11 Testability (must be testable automatically at the plant)
  1.8 Product life

1.9 Profit goals

2. MAJOR CONSTRAINTS

2.1 Price constraints

The initial end-user price, for the minimal machine, should be about $500. It is our intent to have a clear path to lowering the price to $300 after 18 months, while maintaining profit margins.

2.2 Weight and size constraints

2.3 Memory size

We may wish to fix memory size (and eliminate many possible user hardware options) so that software runs on all Macintoshes. Users should not have to know about how many bytes of memory they have. This may also allow us to produce the machine for less. A user minimum of 32K bytes is suggested. This will depend on mass storage considerations as well.

3. POWER SUPPLY

3.1 AC supply from the mains

3.1.1 US

3.1.2 Foreign

3.2 Battery supply

3.2.1 Weight

3.2.2 Reliability

3.2.3 Operating time

It would be nice to obtain 6 hours without having to plug it in. What is a minimum that is still useful?

3.2.4 Potential leakage problems

3.2.5 Primary or Secondary cells?

3.2.6 Accessory pack at extra cost for battery power.

3.3 Solar cells

To keep perpetual calendar independent of other power supplies?

4. HUMAN INTERFACE: DISPLAY

The screen should be soft if possible. Image data compression should be considered to conserve memory.

4.1 CRTs

4.1.1 Home TV

We should have RS-170 and NTSC compatible output. It should work into any standard TV equipment, such as recorders. Both video and modulated outputs should be available. If cost constrains, the video would go.

4.1.2 Built-in display

There is the possibility of no built-in display and only a built-in printer. The hard-copy device, on the other hand, may be part of the display.

4.1.2.1 Conventional CRT

4.1.2.2 Flat CRT

4.1.2.3 Projection CRT

This option has much flexibility, especially in conjunction with a photographic hard copy scheme.

4.2 LCD display

4.3 LED display

4.4 Plasma display

4.5 Other technology (e.g. laser)

4.6 Size of screens

4.6.1 Number of characters (24 X 80 on CRT, 2 X 80 in portable display)
Consider foreign language fonts.
4.6.2 Graphic resolution (suggested minimum: 256 X 256)
4.6.3 Physical size
4.7 Color (probably only on external display)

5. MASS STORAGE
Some form of mass storage must be built in.
5.1 Floppies
5.1.1 Made by us
5.1.2 From outside vendors
5.1.3 Very small floppies (2 or 3 inch)
5.1.4 Compatible with any previous product?
5.2 Bubble memories
5.3 Cassettes or other tape based storage
5.4 Other technologies

6. HUMAN INTERFACE: INPUTS
6.1 Typewriter keyboard
This is probably a necessity. Atkinson points out that the halves are separable. Keyboard should be software mapped if possible (any combination should be valid).
6.1.1 Keyboard layout
6.1.2 Keyboard electronics
6.2 Microphone
6.2.1 As add-on accessory
6.2.2 As built-in. It may be possible to use speaker as microphone although separate microphone probably preferable
6.3 Audio (this and 6.2 with built-in A/D)
6.4 Photocell (perhaps in conjunction with a light pen or wand?)
6.5 Graphic input: joystick, force stick, ball, or tablet
6.6 Other transducer input: pressure, moisture, temperature etc.

7. COMMUNICATIONS
7.1 RS232
7.2 Phone connector (built-in modem and DAA)
7.3 Applenet
7.4 Other
7.4.1 WWV reciever
7.4.2 TV or radio receiver
7.5 RF link
7.6 To a paging system
7.7 IEEE 488

8. SOFTWARE
8.1 Major design critera
8.1.1 "Zero defect" programming
8.1.2 Excellence of human interface
8.1.3 Extraordinary testing
8.2 Initial software offering
There must be a large initial offering of software. Some examples might be
8.2.1 Checkbook balancing
8.2.2 Many games (it will be seen as a toy to some purchasers)

8.2.2.1 Chess
8.2.2.2 Backgammon
8.2.3 Word Processor
8.2.4 BASIC
8.2.5 Elementary communications protocols
8.2.6 Instructional programs (e.g. typing, arithmetic)
8.2.7 Daytimer (can we license use of this name?)
8.2.8 Personal 'phone book
8.2.9 Bulletin board
8.2.10 Business software
8.2.11 Telephone simulator (it has a mike and speaker)
8.2.12 Terminal simulator (also: data entry device)
8.2.13 High precision scientific calculator

Note that many of these packages could be applications of a more general DBM system with preset data and structure.

8.3 Cumulative software effort
8.4 User languages
8.4.1 BASIC
8.4.2 APL
8.4.3 Pascal
8.4.4 Other
8.5 System development languages
8.5.1 Pascal
8.5.2 Forth
8.5.3 Assembler
8.5.4 Other
8.6 Application development languages
8.6.1 Pascal
8.6.2 Pilot
8.6.3 Other
8.7 Software security
8.8 Operating system

9. MANUALS
9.1 Self-teaching, on-line manuals
9.2 Reference manuals

10. SCHEDULE (very preliminary)
1 Nov 79 Preliminary design spec
1 Mar 80 Final design spec
1 Jul 80 Engineering Prototype
1 Mar 81 Prototype for shows
1 Sep 81 Production in time for Xmas 81

11. RESOURCES REQUIRED
11.1 Personnel
11.2 Capital equipment
11.3 Supplies

12. TESTING
12.1 In house
12.2 On a wide range of potential users
12.3 Of completed systems/programs/manuals

13. EXPERIMENTATION
- 13.1 With competitive machines
  - A. HP-41C
  - B. Atari 400
- 13.2 With technologies
  - A. Attach a touch-panel to an Apple II
  - B. Attach a few small-screen CRT's to Apples
- 13.3 By programming simulations
  - A. Small screen sizes (in terms of resolution and characters)
  - B. Simple editors (e.g. my "one command" editor)
- 13.4 With services
  - A. TCA
  - B. PCNET
  - C. Other "bulletin board" services
  - D. DIALOG (Lockheed) BALLOTS (Stanford)

14. ENCLOSURE AND COSMETICS
- 14.1 Design and materials
- 14.2 Thermal considerations
- 14.3 Choice of colors and case styles

  If this is to be truly a product for the home, shouldn't we offer it in various colors? Matched to the 5 standard home appliance colors? There is also the possibility of offering options such as a wooden case.
- 14.4 Self-protecting case design (lid opens to become display, for example)
- 14.5 A handle.
- 14.6 If the design is modular, the parts should snap together electrically as well as mechanically.

15. PRINTER AND HARD COPY PICTURES
- 15.1 Built-in
  - 15.1.1 Film based (polaroid or other quick technology)
  - 15.1.2 Thermal
  - 15.1.3 Electrostatic
  - 15.1.4 Dot-matrix impact
  - 15.1.5 Ink jet
  - 15.1.6 Laser
  - 15.1.7 Other
- 15.2 External
  - 15.2.1 Letter quality

    Is it silly to try to sell $3000 printers with this machine?
  - 15.2.2 Portables

    All the possibilities under 15.1 fit here as well.
- 15.3 Width of paper (8.5 inch probably)

16. OTHER PERIPHERALS AND FEATURES
- 16.1 AC switched outlets (possibly computer controlled)
- 16.2 Speaker (with speech and/or music synthesizers?)
- 16.3 Credit card reader or HP card reader or both
- 16.4 Real time clock (essential)

  Perhaps based on a watch module, always runs.
- 16.5 Actuators (e.g. R/C servos)
- 16.6 TTL outputs

16.7 Soft "off" switch.
16.8 Plotter
16.9 Check reader
16.10 Video disk (needs interface standards)

17. THE APPLE TIMESHARING NETWORK

One of the most powerful uses of Macintosh will become viable only if a service such as TCA is available. We will have to consider setting up a nation (world?) wide set of local numbers for a number of purposes to be covered in another document. A standard protocol will have to be promulgated. Study Viewdata, Teletext, Prestel

18. FUTURE PRODUCTS IN THE MACINTOSH FAMILY

Items rejected from consideration as built-in may be moved here, as well as new ideas.

19. IMPACT ON VARIOUS DEPARTMENTS OF APPLE

19.1 Purchasing
19.2 Manufacturing
19.3 Marketing
  19.3.1 Advertising
  19.3.2 Dealers
19.4 Engineering
  19.4.1 System software
  19.4.2 Applications software
  19.4.3 Analog electronics
  19.4.4 Digital electronics
  19.4.5 Mechanical engineering
19.5 Publications
19.6 New Product Review
19.7 Service

20. ANALYSIS OF COMPETITION

20.1 Atari
20.2 Texas Instrument
20.3 Commodore
20.4 Tandy
20.5 Japan
20.6 Calculators
20.7 Mattel
20.8 Typewriters
20.9 Accounting machines
20.10 Electronic games
20.11 Video games
20.12 Other

21. CPU

It is assumed, for the time being, that memory will be byte oriented, and that the CPU or CPUs will be 8 or 16 bits or some mix of the two.

21.1 Single or multiple CPU
21.2 Off-shelf or our own
21.3 Consider 6809

THE MACINTOSH PROJECT

DOCUMENT 3 VERSION 5

TITLE: THE APPLE COMPUTER NETWORK

AUTHOR: JEF RASKIN

DATE: 11 Sep 79-11 Oct 79

1 INTRODUCTION

There are very few potential uses of the personal computer per se in the home at the present time. The question "What do you do with it?" still haunts the industry. While balancing checkbooks, playing chess and writing letters are all viable uses, it is likely that a true mass market cannot be supported on the basis of such applications. In the face of this problem, most manufacturers, seeing the hobbyist and technophile markets becoming saturated, have turned to marketing business systems. The business system market is big and legitimate opportunities abound there, but the volume can never be as large as it would be for an item that goes to consumers in general.

There is a feeling in the industry that telecommunications will become a key part of every computer market segment, and this is increasingly becoming so. Many experiments and a few successful services are in operation. Aside from long-standing timesharing systems such as GE and TYMSHARE, we have the ARPA net, Xerox's internal Ethernet, TCA (alias "The Source"), Prestel, the MECC network, and many others. Appendix 1 lists a few commercial services that may be of interest to us. A set of "underground" message centers have come into operation, for example, the PCNET. There are also a few other individuals and small groups that set up a microcomputer with an autoanswer modem and some software that allows users to leave and retrieve messages.

According to "Computer Retailer", Radio Shack and Western Union are working out some cooperative venture involving WU's "Mailgram" service whereby Radio Shack computer owners can exchange messages.

It is clear that one answer to the question "What do you do with it?" will probably be: "I use it to send birthday greetings to Aunt Tillie." More to the point there are a number of easily forseen potential uses for a network of personal computers. What is more exciting is that, as has happened with the computers themselves, there is the potential for many <u>unforseen applications.</u>

1.2 POSSIBLE APPLICATION AREAS

Many applications have been put forward. Among them are:

Time of day; News (with a boolean query data base); Stock Market (as per what we are already doing); Soap Opera Condensations; A guide to local TV programs (what's on at 9:00PM?, any westerns tonight?); Message forwarding and distribution; Fax transmission (special case of message: the bits are

interpreted pictorially); Weather Travel Info; Phone directory; Local, area or national business directory; Apple program distribution channel; Apple update distribution channel; Access to Lockheed's DIALOG or Stanford's BALLOTS systems or similar ones; A better way to answer user questions than a phone based hotline at Apple; Library of Congress card catalog; Legal precedents; Program exchange; Educational courses; Educational testing; Voting; Computer program exchange; Advertising; Computer dating; Tax information; Banking (another step to the cashless society (If taxes don't reduce us to a cashless state first)); Access to large data storage for individual needs; Access to computer power (i.e. timesharing); Insurance quotes; Credit information (what is available: what is my status); Market research; Purchasing information (who has the cheapest refrigerator model 34-aa within 10 miles); Plane schedules; Dictionary and Encyclopedia searches.

The list is potentially endless. Most come under one heading: Access to a Data Base. A few come under the heading: Communication. The remaining handful are miscellaneous.

The point of this list is that telecommunications provides a host of answers to the "What do you do with it?" question. What follows is a proposal for supplying customers with this kind of service.

2. IMPLEMENTATION OF AN APPLE TELECOMMUNICATION SYSTEM

2.1 SOME NON-TECHNICAL PROBLEMS

It is clear that setting up any kind of independent communications network runs afoul of many bureaucratic agencies, and would alarm many companies now in the communications industry. For these reasons alone, it is best to use existing communication channels. The problems of being a user of such a system are probably less than the problems of being in the industry. document M 12 discusses some of the potential problems with being a user of the telephone network.

2.2 NEED FOR PROMPT AND AGGRESSIVE POLICY

Apple cannot possibly create the data bases that are required to make this proposal viable. As we have done with Dow Jones, we will have to negotiate access for our users to existing data bases and services. Our success in this field will depend on aggressive and prompt action to secure (possibly exclusive) use (with respect to personal computers) of what we see as the most important services.

2.3 TELEPHONE BASED SYSTEM

The most universal bidirectional data path to homes in this country is the telephone. We presently have the technology to provide a low-cost computer with a 300 baud modem. Since the Carterphone decision, access to the telephone network has been assured (as least as much as anything can be assured in the communications field). There are a number of ways of providing a universal service through the telephone.

2.3.1 MANY LOCAL NUMBERS

The time sharing services have chosen to have many regional centers or data accumulation points. Dow Jones, for example, provides many local numbers. There are some disadvantages to this scheme: it requires publishing a book of telephone numbers, and some subscribers have to make toll calls if they do not live in or very close to a major population center. One advantage of this system is its redundancy.

2.3.2 A TOLL-FREE NUMBER

With one (or perhaps two) toll-free numbers, every phone in the country can have access to a central computer installation or what I will call an A-node (explained below). This has the psychological advantage of making access to the service appear "free". The number is easily advertised, again encouraging use. Research will have to be done to determine the expected density of useage and just how practical such a system would be.

2.4 LIMITED OR UNIVERSAL ACCESS

Can we, or should we try to, limit access to owners of Apple products? If we allow access to anybody with proper equipment (probably any computer or, perhaps, even some terminals) can we legally build in advantages for Apple owners? For example, the only software available will be for Apples. If we use some technological means of permitting only Apples into the service, this would require special hardware and/or software in future products, and might not be retro-fittable to the Apple II series.

It is my feeling that access should be universal, with some unique services for Apple owners. Greg Justice is devising an inexpensive modem for 1200 baud half duplex telephone line communications. This might be an Apple exclusive.

2.5 TECHNICAL PROBLEMS

If credit or any other sensitive information is involved, security measures will be necessary. Security problems are inevitable in any case since we wish to bill our customers for their useage of at least some parts of the service. Use of other parts may be paid for by the suppliers (e.g. when the service amounts to advertising) and thus seem free to the user. D. E. Denning, in her article "Secure Personal Computing in an Interactive Network" (CACM Vol. 22, No. 8) suggests a method for implementing security where the users, rather than the network, have control of security. While I question the implementation suggested there, the idea that security can come from the user's side is a good one.

Protocols will have to be carefully defined, simple enough for naive users, yet powerful enough to cope with any data base to which we subscribe. There may have to be, as with the PCNET protocols, levels of access from the beginner's character stream to the expert's auto-path-finding data compressed packet. If we are fast enough, the protocols we design may become a de facto standard.

2.5.1 INTERFACIAL SOFTWARE

One way of easing the user's problem at accessing the many data bases that

exist would be to provide, as we have begun to do for the Dow Jones services, programs that iron out the difficulties of using the various services, so that they all appear relatively uniform and easy to use.

This would become an ongoing software project for Apple, and would represent the least involved method of providing access to data bases. It would not provide a universal message transfer system, however, and thus possibly decrease the desirability of the Macintosh system. It would also not provide a single telephone number for all services, and might require each user to have multiple contracts. Users would not be encouraged to try more services as much as they might if the services were more consolidated.

If we adopt this approach, we will have to define a set of unified protocols for accessing data bases--of diverse kinds. This is no easy task, since most of them were designed quite independently.

2.5.2 ESTABLISHING A-NODES

The next level up in user convenience from providing software for stand-alone Apples to facilitate access to each data base would be to provide an A-node, or data concentration center or centers. This A-node would be accessed through a single protocol from the user's computer, and then the A-node would initiate and monitor (for billing purposes) the connection with the various data base systems. This process would insulate the user from having to be familiar with each different data base's telephone number and protocols.

The A-node would also be a message storing and forwarding center.

3. IMPLICATIONS FOR THE MACINTOSH PROJECT

This proposal grew out of attempting to answer the "What do you do with it?" question. Without some sort of network/data base service the question is going to be much harder to answer. I am proposing here that the creation of Apple communications, in some form, is part of the definition of the Macintosh computer project.

4. IMPLICATIONS FOR APPLE COMPUTER INC

We don't think of the telephone company primarily as a manufacturer of the little $40 things with dials or pushbuttons that we have in our homes and on our desks. The implications of this proposal, at one extreme, is that Apple will be seen, in the future, not so much as a builder of hardware, but as the purveyor of a service that interpenetrates the telephone network, and provides information.

On the other extreme, Apple will be in its present position, adding access to data bases on a piecemeal fashion. Message transfer will not become a useful function unless somebody else happens to start a message system that is universal enough and otherwise meets our needs. I do not like trusting to luck.

The high cost to Apple of having done our early software piecemeal and in an ad hoc manner should act as an example of why we should not let our

communications effort be similarly scattered.

The intermediate approach, using the A-node, may be the most practical. At least its requirements are not excessive in terms of hardware or software, and it can start as a mere message center (for the whole United States at first) and then grow as we contract with providers of data bases and bring them on line.

This proposal, if it is carried out, will represent an additional direction for the company. We will not be alone in the personal computer communications field, we can strive to be the best. Certain vested interests may be threatened by an Apple communications system, and there may be unpleasant pressures.

5. SUMMARY

It is my opinion that the A-nodes provide a practical solution to a good portion of the "What do you do with it?" question. If, in the next few months, a clear alternative occurs (such as might be provided by the growth of a company such as TCA) we should consider it. We must do something in the way of providing communication between Apples in locations distant from each other, and communication with sources of information.

Mike Markkula has suggested that this may be an undesireable path in that others will be doing it, and we can use their services. He further suggested that we establish a protocol early on, and publish it in order to obtain a leadership position. I agree with trying to establish the protocols early, and promulgating them, but I think we must also press ahead with a detailed study of the costs and benefits of our own system.

I am concerned that many of the existing services are either too poorly set up, or don't have the breadth of vision required. We may also need growth rates not anticipated by the existing companies. If we do not have a communications network to use, then what will people do with the million Macintoshes we wish to produce?

APPENDIX 1, SOME SOURCES OF DATA BASES AND COMMUNICATIONS

Telecomputing Corporation of America
1616 Anderson Road
Mclean, Virginia 22102
A service designed for the personal computer user. It is commonly called "The Source". Advertises in Byte.

Computserve, Personal Computing Division
5000 Arlington Centre Blvd.
Columbus, Ohio 43220
Tradenamed "Micronet", it is a less ambitious project than The Source. Advertises in Byte.

SDC Search Service
2500 Colorado Ave.
Santa Monica, CA 90406
A commercial service with no special attention to microcomputer users, mentioned here because it has a wide range of services.

The Information Bank
1719 Route 10
Parsippay, NJ 07054
A news service to the New York Times, Wall Street Journal, Business Week and some 60 other publications. No special attention to microcomputer users.

Lockheed's Dialog data base and Stanford's BALLOTS would be fine candidates, along with our existing Dow Jones service. All in all, there are many services available. A listing of such services appears in the Applications Directory of The Association of Time Sharing Users, Inc., P.O. Box 9003, Boulder, CO 80301. Almost all the services listed there provide economic, stock and commodity, market or demographic information. The two two most interesting from that list are shown above. Thanks to Tom Whitney for showing me this catalog.

THE MACINTOSH PROJECT

DOCUMENT 4 VERSION 1

TITLE: DESIGN CONSIDERATIONS FOR AN ANTHROPOPHILIC COMPUTER

AUTHOR: JEF RASKIN

DATE 28-29 May 79

[This document was written before the Macintosh project was operating under that name, and was still called "Annie". This note was written as an observer at that time not directly involved in the project. (Comments in brackets have been added on Oct. 11 79)]

This is an outline for a computer designed for the Person In The Street (or, to abbreviate: the PITS); one that will be truly pleasant to use, that will require the user to do nothing that will threaten his or her perverse delight in being able to say: "I don't know the first thing about computers", and one which will be profitable to sell, service and provide software for.

You might think that any number of computers have been designed with these criteria in mind, but not so. Any system which requires a user to ever see the interior, for any reason, does not meet these specifications. There must not be additional ROMS, RAMS, boards or accessories except those that can be understood by the PITS as a separate appliance. For example, an auxiliary printer can be sold, but a parallel interface cannot. As a rule of thumb, if an item does not stand on a table by itself, and if it does not have its own case, or if it does not look like a complete consumer item in of itself, then it is taboo.

If the computer must be opened for any reason other than repair (for which our prospective user must be assumed incompetent) even at the dealer's, then it does not meet our requirements.

Seeing the guts is taboo. Things in sockets is taboo (unless to make servicing cheaper without imposing too large an initial cost). Billions of keys on the keyboard is taboo. Computerese is taboo. Large manuals, or many of them (large manuals are a sure sign of bad design) is taboo. Self-instructional programs are NOT taboo.

There must not be a plethora of configurations. It is better to offer a variety of case colors than to have variable amounts of memory. It is better to manufacture versions in Early American, Contemporary, and Louis XIV than to have any external wires beyond a power cord.

And you get ten points if you can eliminate the power cord.

Any differences between models that do not have to be documented in a user's manual are OK. Any other differences are not.

It is most important that a given piece of software will run on any and every computer built to this specification. There must be no differences between

machines whether in terms of I/O, speed, memory size, configuration, or possible accessories.

(Speaking of accessories--off hand, the only accessory that I can see being sold is a printer. If this can be built in (on EVERY machine) then there is little cause for ever having accessories at all. This is optimal.) [So far, price constraints and the pervasive idea of a network have changed this a bit.]

It is expected that sales of software will be an important part of the profit strategy for the computer.

If it is anticipated that fewer than 100,000 of these anthropophilic computers will be sold in a 2 1/2 year period, the project should not be undertaken.

A CANDIDATE FOR SUCH A MACHINE

The computer must be in one lump. This means, given present technology, a 4 or 5 inch CRT (unless a better display comes along in the next year), a keyboard, and a disk integrated into one package. It must be portable, under 20 lbs, and have a handle. "Apple V" would not be a bad handle. It should fit under an airline seat. It would be best if it were to have a battery that could keep it running for at least two hours when fully charged.

Some things are easy to choose. Performance, in the usual computer-science sense, is not too critical. An 8-bit CPU, eight 64K RAM chips, one RS-232 interface, a telephone jack, and some 200K bytes on the diskette would be fine. There must be a clock-calendar that is battery powered.

Other things are harder to pick. Clearly, there should be BASIC available. And there should be some underlying system language, reachable through BASIC, so that OEM software houses (and our own programmers) can do what's necessary. One very small, inexpensive and compact language suitable for this application is FORTH. Having to use an external development system will hamper the growth and sales of the machine.

The end-user cost for this machine should be $500 or less, to be sold early in 1982 (or, better still, by Christmas 1981).

THE ACCESSORIES

Printers. Maybe.

SOME USER-VIEW SOFTWARE CONSIDERATIONS

The system must not have modes or levels. The user always knows where he or she is because there is only one place to be.

THE BASIC

The language should be pure interpreted. All system commands should be embedded in the language, all statements in the language must be commands. The program should be user-interuptable (and process interuptable) and

resumable even after being changed. Anything that can be done by the user can done by the program and vice versa.

Consistency is important. All names, whether file names or variable names or array names... should have the same syntax. Wherever a constant can be used, so can an expression be used. Strings should not behave differently than other arrays. All arrays should be dynamically allocated.

Declarations are taboo.

Or, rather, requiring declarations am taboo.

TOYS

Graphics must be in the language as well as sound generation via an internal speaker. The present set of sound-generating chips should not be considered.

Cursor controls should be on the keyboard, and should be used where graphic input is called for. The Apple III keyboard is not far from ideal.

RS-THIS AND RS-THAT

Standard RS-170 video output is not a bad idea, especially if it is to be used in schools. That was five two letter words in a row. But video output is not necessary.

Actually, neither is an RS-232 port. But I have a suspicion that it might be nice. Perhaps the phone jack, having two extra wires as it does, could become an RS-232 minimal (3 wire) port with an adapter. The minimum number of holes in the case through which fingers, screwdrivers (either metallic or liquid), EMI or earwigs can crawl is to be desired. I guess that adapters are OK as accessories.

Ye same olde 'phone jacke couldst be into service yprest, forsooth to accomodate divers keyboards, sych as yon organ hath.

The utility of the computer is vitally enhanced if the 'phone jacke had some personae with whom to talk, to wit, a network is an essential part of the idea of Annie. And I don't mean ABC, CBS, PBS, NBC and Mutual.

SUMMERY

That means fair, warm weather, just after spring.

SUMMARY

Let's make some affordable computers.

THE MACINTOSH PROJECT

DOCUMENT 5 VERSION 1

TITLE: PRELIMINARY COST INVESTIGATION

AUTHOR: JEF RASKIN

DATE: 27 Sep 79

1. COST

One of the primary goals for Macintosh is that it should sell for $500--with the possibility of reducing the cost to $300 with increasing quantity.

By our present selling price versus manufacturing cost ratio this means that Macintosh should cost us about $125 to build. Consider this highly speculative cost breakdown (assuming a case and board about Apple II size)

PART	1981 COST	POTENTIAL COST	
keyboard	20	10	
CPU			
6502	4	3	minimum cost
6809	12	7	likely candidate
68000	90	65	upper end
64K bytes	80	50	8 64K RAMS (@ $10 and $6.25)
video chip	15	10	
modem chip	20	15	with DAA
case	15	5	depends on size
power supply	15	10	
p c board	12	8	depends on size
misc	25	15	connectors, fasteners
SUBTOTAL	214	130	basic electronics cost
build, test	10	2	design for automation

(this last category has much room for lowering cost, and thus it is included even though it is not part of parts costs.)

disk drive	80	35	
display	40	20	

printer	50	25
TOTAL	346	212

For this very minimal estimate we assume a 6809 processor as a passable compromise between power and cost. It is readily available, allows our video techniques, requires few support chips, and is relatively efficient in terms of compiled code size.

At a 4X selling cost vs our cost ratio there is no way, even without disk, printer or display, that we could get below $850 given these prices. Fully loaded, the selling price may well be $1400. And there are many items omitted from this list, and the costs are probably optimistic. At the minimum prices that I can forsee, the base machine might be able to sell for $520. The full package for $850.

These costs may not be spot on, and the 4X multiplier is not fixed, but these are indicative ball-park figures. While the minimum price of $520 might look close to our goal, remember that it is extremely optimistic, includes no peripherals, and should (to meet our goals) be well under $300 (our "eventual" figure).

1.1 TWO POSSIBLE DIRECTIONS

Either we can continue the design of Macintosh with the present desiderata, and abandon our price goals, or we can keep the price ceiling and see what kind of machine we come up with. It seems that the higher priced machine that falls out of this analysis is not much different from a price reduced Sara. Therefore we will examine what we can do for a $500 selling price limit.

1.2 THE MACINTOSH 500

One item on which we should not compromise is a keyboard.

$ 20

Assume, for the purpose of having truly adequate software, 64K RAM

$ 80

We build our own video and modem chips

$ 35

Keeping the case quite small, it and the PC board might be lowered to

$ 25

If things are kept small, the power supply might drop to

$ 12

Add

$ 18

for miscellaneous, and a

$10

CPU to bring us to a nice round total of

$100.

A machine with no peripherals will have to have some ability in ROM. Assume 32K ROM with a word processor and BASIC (both written in Pascal and compiled to 6809 code). This adds

$25

or so. This, inflated by details and committee embellishment, gives us a chance at a $500 computer. Note the following facts:

A. No display is included
B. No disk is included
C. No printer is included.

The solution to providing these is as add-ons that attach both mechanically and electrically to the basic Macintosh product.

THE MACINTOSH PROJECT

DOCUMENT 6 VERSION 2

TITLE: GENERAL CRITERIA

AUTHOR: JEF RASKIN

DATE: 28 Sep 79

1. MAJOR CONCEPTS

The most important goal (after the operational goals set out in Document 3), in my opinion, is for this computer to have a selling price of $500 or less. If it is significantly more expensive than this, there will be little to differentiate it from some of our other personal computers. Macintosh is designed to be much easier to use than existing computers, and it must be provided with a range of pre-programmed applications that the average person will find alluring.

1.1 PACKAGING

The package must be compact (13" width, 13" depth, 5" height maximums), lightweight (under 10 lbs), and robust (it would be desirable that a three foot drop onto a solid floor cause at most cosmetic damage).

Since the cost analysis seems to indicate that a display, a disk, or a printer cannot be part of the package (Document 5), these and other peripherals should be attachable to the basic unit. It is imperative that Macintosh not evolve into a tangle of wires. Therefore it is probably necessary for each peripheral to attach to the basic unit both mechanically and electrically, allowing a daisy-chain bus and therefore an open-ended number of peripherals. Figures 4, 5 and 3 show some possible physical arrangements. The IEEE 488 bus might be considered as part of the definition for this application.

This would allow the basic computer, with video, RF, phone and one RS-232 outlet to have a very attractive price, with relatively inexpensive matching peripherals.

Markkula's intriguing suggestion that the units be connected by radio links rather than physical ones should also be investigated. It may be less expensive than the present suggestion. The problem of FCC regulations and the need for separate power supplies remain open questions.

The modular packaging concept allows us to offer, for example, a variety of displays as technology permits. A strong defeciency of modular packaging is that we would have to write software that can work in a variety of configurations. This greatly increases the cost to produce software, and presents marketing and diagnostic problems as well.

1.2 PORTABILITY

Aside from the size and weight constraints, it would be especially beneficial if the unit could be used portably. One of the add-on packages could be a battery pack with, if necessary, an inverter. The power supply itself could be in one of the add-on packs, giving the user the choice of an AC supply or a DC supply and charger.

If no display is built in, the user has the option of carrying a smaller package, and using a home TV. Most home black and white TVs can support 64 characters per line. I have tested this using the Poly 88 computer, which is a 64 character, upper/lower case machine.

1.3 LEARNABILITY

Macintosh must be easy to use. The keyboard should be typewriter-like, with no computer jargon on the key-tops. Nonetheless, it must be able to generate the entire ASCII code-set. But it will be the design of the software and manuals that will have the greatest effect on the ease with which the computer will be used--coupled with the expectations generated in the user by our advertising and the current personal computer milieu.

A separate set of documents will cover software and manual design, and yet another will have to discuss advertising strategy.

1.4 SERVICEABILITY

Macintosh must be serviceable: Wil Houde would like to give a one year unlimited warranty with the machine. What is required is a long MTBF. In the presence of the strong price constraint we cannot merely take the path of high quality components for whatever design we come up with--we must restrict the ambition of the design so that we can afford to build it well.

The modular approach taken above is both a help and a hindrance to reliability: each of the parts will be designed independently, and to different restraints, they can even have different warrantees. On the other hand, there is now a set of electrical and mechanical connections that otherwise would not be there.

Since the signal may pass through a number of boxes, a failure in an intermediate module might affect the operation (and the diagnosis) of another unit.

A single-box design puts many electronic and mechanical pieces together, thus significantly increasing the probability of a failure of the whole device in a given time period. It can be argued that if any of the modules fail in a modular design, any program that uses that module cannot run anyway, and the entire machine is lost in spite of the modularity. On the other hand, it is easier to manage the shipment and servicing of a single box. The software reliability of a single-box system is also greater.

Serviceability is also a function of the quality of the manuals. A separate document will discuss the on-line and off-line manuals

1.5 MAINTAINABILITY

Ease of repair is a function of simplicity, accessibility, modularity and the design of diagnostic software and procedures. Aside from having the design monitored by our service and repair personnel, the designs being considered seem inherently maintainable. We should consider setting up a system programming effort to write a set of diagnostic routines as part of the Macintosh project.

1.6 PRODUCEABILITY

At all times, our production engineers will be consulted on the design of the computer. The fundamental design should be single board, with an integral keyboard if possible.

2. SOFTWARE DESIGN CRITERIA

2.1 LEARNABILTY

There is one quality that software can have to inprove learnability: consistency. That one attribute is probably more important than the details of what we are consistent about. Consistency also minimizes the size of our manuals, and decreases the time it takes to write and test software. It increases the time it takes to design a system, since all aspects of the software, from command level to every application program must be considered in setting up the design parameters.

2.2 MAINTAINABILITY

All the software for Macintosh must be written under the same system and in the same language. The operating system will probably be a descendant of SOS, and the language will probably be Pascal. (This should not prejudice the choice of user languages).

2.3 RELIABILITY

We expect that we shall sell hundreds of thousands of Macintosh computers. It should be a guiding principal that software goes out as the result of a program designed in the spirit of the NASA "zero defect" program since it will become uneconomical to update programs once they are sold or delivered as part of the system.

THE MACINTOSH PROJECT

DOCUMENT 7 VERSION 7

TITLE: A MODEL OF MEMORY VS. DISK CHOICES

AUTHOR: JEF RASKIN

DATE: 2-4 Oct 79

1. A PATH TOWARD THE DESIGN

My investigations start off with the human input, the keyboard, and an application, personal (as opposed to business) word processing. With this starting point, we form a branching trail through the woods of possibilities.

1.1 THE KEYBOARD

Macintosh will have a typewriter styled keyboard with embedded numeric pad (as on the HP 2621). No separate key pad should be provided for three reasons: it shaves a bit off the cost, it makes the package physically smaller, it looks less forbidding. Such a keyboard is excellent for word processing, and we will first explore system requirements for this application. Aside from the embedded numeric pad, the keyboard layout will not differ much from Sara's excellent design.

The cost of such a keyboard will be about $20.

1.2 WORD PROCESSING

A practical word processor has either a buffer of over 20K bytes for user input, or a mass storage device whose operation is totally invisible. Since an overflow of the internal buffer can happen while a person is typing (even in the middle of a word), the time that the keyboard does not respond while the system is squirrelling away the buffer should be under 0.1 second. You don't want the system breaking the user's concentration by upsetting him or her with random pauses.

Since this application places some restrictions on storage, there are (at least) two ways of going about it: supplying sufficient random access storage, or a fast mass storage device.

1.2.1 SUFFICIENT RANDOM ACCESS STORAGE SOLUTION

This can be provided through RAM, or through some other technology such as bubble memory. For our application, we have said we need a minimum of 20K bytes for the user's area (there being no real upper limit on how much user storage can be put to use). In addition we need the word processing software, and the underlying operating system. Here the path forks again, as the amount of storage required for the software depends on the CPU chosen. If the CPU is a Lisa type, then the software will probably require

about 10K bytes, and the system need only have 32K bytes altogether. If the CPU has the power of, say, a 6809, then the interpreter will require perhaps 8K, the word processor about 10K and the operating system around 8K bytes. This CPU would require 46K bytes. If 16K bytes of RAM cost $25, (a 6809 costing about $10) the Lisa type processor would have to be under $35 to be cost effective in this application.

1.2.2 FAST MASS STORAGE SOLUTION

In the Macintosh time frame, the only viable mass storage is floppy disks. Hard disks are too expensive, and their media is not removable. Bubbles seem to be coming along a bit too slowly. Nonetheless, we will keep our eyes open.

While floppies are usually considered to be too slow for our application, they can be fast enough if we go to a multiple processor system so that spooling can go on in parallel with word processing. Assumming (and here is another branch in the path) 24 lines of 64 characters (1536 bytes) there could be a minimum 2K buffer which both refreshes the screen and is the text buffer. When it scrolls off screen, it scrolls onto a floppy. The few hundred extra bytes gives the system enough time to start up the disk drive.

Searches might, with this kind of scheme, be a bit slow and noisy. There are many trade-off points, the buffer can be expanded to 4K, or 8K, or larger, and the disk useage goes down with each increase in memory. A mathematical model can be built for this situation.

1.2.2.1 THE MODEL

Assume that the user has M kilobytes of memory at a cost of $b per kilobyte, a floppy (and associated processor and support circuitry) which costs $f with a transfer rate of T seconds per kilobyte, and a processor and memory that permits a search for a pattern match at a rate of G seconds per kilobyte in a document of D kilobytes (D*G seconds for the complete document when the entire search is in memory). We can now examine how many kilobytes per second of searching a system can achieve with disks, and what the cost is for different memory/disk tradeoffs.

If

D <= M

then no disk accesses are required, as the entire document fits in memory so that the time required for the search is

D*G

at a hardware cost of

Mb.

In the case

D > M

where there is insufficient main memory for the entire document, disk accesses occur. Assuming again a search through the entire document, the number of accesses (less one) is S which can be calculated by S = TRUNC (D/M). The time for reading in each segment except the last is L + M*T, where L is the latency time of the disk. The last segment is of length D - S*M (which is 0 if D MOD M = 0). For the total number of accesses the time is S*(L+M*T) + L + (D - S*M)*T. To this is added the in-memory search time, D*G. Simplifying the resultant expression, the total search time is

D*(G + T) + L*(S + 1)

which is easily interpreted in this form: the first major term represents the total time for transferring and searching the document, the second term represents the sum of the latency times.

The cost of the hardware is the memory cost plus the floppy cost

(M*b + f).

This model must be modified where, as on our floppies, the latency time is significantly lower if the disk is already spinning. In this case

L = U

where U is the motor-starting latency and

L = V

where V is the rotational plus average seek time latency. Usually the time V is included in U in the motor-starting case. There is also a fixed shut-down time, W.

The time for a search through a complete segment is (D*G)/(S + 1), so that if this time is greater than or equal to W, then

L = U

otherwise

L = V + W - ((D*G/(S + 1)))

which is the rotational plus seek latency, plus what is left of the motor-off time after the search is complete. This ignores the very small program overhead in the spooling algorithm.

A concrete example in the late 1981 time frame might have the floppy (and associated hardware) at $55, and memory at $1.60 per kilobyte. Assume that the transfer rate averages 0.1 seconds per kilobyte (this is the present Apple II 16 sector speed), and that a search in memory takes .006 seconds per kilobyte (this is about the speed at which Sara will work). Assume further that the latency U is 1.0 seconds, and the average latency V is 0.15 second (on the Disk II, the average rotational latency is 0.1 second, and a

track to adjacent track seek about 0.01 second).

The reader is directed to the programs M 9 and M 10 in this series of documents to play with these values.

THE MACINTOSH PROJECT

DOCUMENT 8 VERSION 1

TITLE: REPLY TO JOBS, AND PERSONAL MOTIVATION

AUTHOR: JEF RASKIN

DATE: 2 Oct 79

1. TODAY STEVE JOBS SAID: "DON'T WORRY ABOUT PRICE, JUST SPECIFY THE COMPUTER'S ABILITIES."

It is impossible to merely start with the desired specifications: it is too easy. We want a small, lightweight computer with an excellent, typewriter style keyboard. It is accompanied by a 96 character by 66 line display that has almost no depth, and a letter-quality printer that also doesn't weigh much, and takes ordinary paper and produces text at one page per second (not so fast so that you can't catch them as they come out). The printer can also produce any graphics the screen can show (with at least 1000 by 1200 points of resolution). In color.

The printer should weigh only a fraction of a pound, and never need a ribbon or mechanical adjustment. It should print in any font. There is about 200K bytes of main storage besides screen memory and a miniature, pocketable, storage element that holds a megabyte, and costs $.50, in unit quantity.

When you buy the computer, you get a free unlimited access to the ARPA net, the various timesharing services, and other informational, computer accessible data bases. Besides an unexcelled collection of application programs, the software includes BASIC, Pascal, LISP, FORTRAN, APL, PL\1, COBOL, and an emulator for every processor since the IBM 650.

Let's include speech synthesis and recognition, with a vocabulary of 34,000 words. It can also synthesize music, even simulate Caruso singing with the Mormon tabernacle choir, with variable reverberation.

Conclusion: starting with the abilities desired is nonsense. We must start both with a price goal, and a set of abilities, and keep an eye on today's and the immediate future's technology. These factors must be all juggled simultaneously.

2. WHY TIME COST ARE OF THE ESSENCE

In an article for IEEE Computer, Raskin and Whitney argued that the difference between a computer and a programmable calculator was the former's ability to handle text. The HP-41C, at $295, is a (weak) computer. Ohio Scientific has announced their C4P computer, which at $698 looks like a "good buy". In fact, it has 8K of RAM, and the cost to get one disk and an additional 16K is $1000. Nonetheless, I would like to see Apple have a computer in the $500 class, and of better value than the competition's, as soon as possible.

When a person is making a decision to buy their first computer, they do not know enough to go much beyond the advertisments and the bottom line. Since I believe the computers we sell, and are going to sell, are better and better supported than the competition, I would like to see a person be able to buy an Apple quality product for a low entrance price--with accessories available to bring it up to the level of computational power that we know is necessary before the personal computer is truly a satisfactory tool.

We may not be able to achieve a mass market unless we can educate people by selling them a system that we know they will want to expand, and letting them learn why they need mass storage, printers and all the rest. This is one of the main ways that Macintosh should be different from Sara and Lisa.

In the light of the above mentioned machines, the $795 Pet, and $698 TRS-80, the Atari 400, and some other machines now coming along, it is clear that we need a product that looks (and is) competitive. We can (and do) among ourselves point out the flaws from a user's point of view in the competitor's product--it is hard to see how anybody could put up with a TRS-80 if they have had much experience with an Apple II (At 16K the difference is only $200 between a Level II TRS-80 and an Apple II Plus). The fact is that a random customer will not have the opportunity to make the comparison beyond seeing the $200 difference. (That and Radio Shack's 6,000 or so outlets).

Therefore let's use our superior engineering, software, documentation, production, service and marketing abilities to produce an excellent low-cost computer. I am not suggesting we copy and follow our competition; I suggest that we leap-frog them.

My personal interest in small computers is evangelical, that's why I tackled manual writing first. That's why I fought for Pascal, that's why I work for Apple. My message is that computers are easy to use, and useful in everyday life, and I want to see them out there, in people's hands, and being used. My purpose will not be accomplished unless Apple continues to be one of the leaders in terms of pure numbers of customers. I don't like being second or third, I would rather Apple be number 1.

I think that one way to keep our position, and possibly get ahead, is by putting out a very low cost machine. That's why I felt honored when given the charter to start the design effort on Macintosh.

```
(*

THE MACINTOSH PROJECT

DOCUMENT 10 VERSION 2

TITLE:  VARIABLE MODEL OF MEMORY VS. DISK COSTS

AUTHORS:  JEF RASKIN AND PEGGY MILLER

DATE:  3-4 Oct 79

*)

PROGRAM TRADEOFF;


TYPE SETOFCHAR=SET OF CHAR;
     CRTCOMMAND= (ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN);


VAR M X,L,D,s,TIME,COST,f,b,T,G,U,V,W: REAL;
    ITIME,ICOST,IM: INTEGER; (*THESE NEEDED FOR PRINTING*)
    ITC: INTEGER[7];
(* The variable names have been chosen to correspond to Macintosh document
 number 7 which forms the documentation for this program*)

    CH:   CHAR;
    CRTINFO:   PACKED ARRAY[CRTCOMMAND] OF CHAR;
    PREFIXED:  ARRAY[CRTCOMMAND] OF BOOLEAN;
    PRINT, DONE:  BOOLEAN;
    REPORT : INTERACTIVE;

PROCEDURE GETCRTINFO;
(*********************************************************)
(*                                                       *)
(*  READ SYSTEM.MISCINFO AND GET CRT CONTROL CHARACTER INFO     *)
(*                                                       *)
(*********************************************************)
VAR BUFFER: PACKED ARRAY[0..511] OF CHAR;
    I,BYTE: INTEGER;
    F: FILE;
BEGIN
  RESET(F,'*SYSTEM.MISCINFO');
  I:=BLOCKREAD(F,BUFFER,1);
  CLOSE(F);
  BYTE:=ORD(BUFFER[72]); (* PREFIX INFORMATION BYTE *)
  CRTINFO[LEADIN]:=BUFFER[62];  PREFIXED[LEADIN]:=FALSE;
  CRTINFO[ERASEOS]:=BUFFER[64]; PREFIXED[ERASEOS]:=ODD(BYTE DIV 8);
  CRTINFO[ERASEOL]:=BUFFER[65]; PREFIXED[ERASEOL]:=ODD(BYTE DIV 4);
  CRTINFO[RIGHT]:=BUFFER[66];   PREFIXED[RIGHT]:=ODD(BYTE DIV 2);
```

```
  CRTINFO[UP]:=BUFFER[67];        PREFIXED[UP]:=ODD(BYTE);
  CRTINFO[LEFT]:=BUFFER[68];      PREFIXED[LEFT]:=ODD(BYTE DIV 32);
  CRTINFO[DOWN]:=CHR(10);         PREFIXED[DOWN]:=FALSE;
END;


PROCEDURE CRT(C: CRTCOMMAND);
(*****************************************************************)
(*                                                               *)
(*  CRT COMMANDS ARE: ERASEOS ERASEOL,UP,DOWN,RIGHT,LEFT.        *)
(*                                                               *)
(*****************************************************************)
BEGIN
  IF PREFIXED[C] THEN UNITWRITE(1,CRTINFO[LEADIN],1,0,12);
  UNITWRITE(1,CRTINFO[C],1,0,12);
END;


PROCEDURE PROMPTAT(X,Y: INTEGER; S: STRING);
BEGIN
  GOTOXY(X,Y);
  WRITE(S);
  CRT(ERASEOL);
END;


FUNCTION GETCHAR(OKSET: SETOFCHAR): CHAR;
(*****************************************************************)
(*                                                               *)
(*  GET A CHARACTER, BEEP IF NOT IN OKSET, ECHO ONLY IF PRINTING *)
(*                                                               *)
(*****************************************************************)
VAR CH: CHAR;
    GOOD: BOOLEAN;
BEGIN
  REPEAT
    READ(KEYBOARD,CH);
    IF EOLN(KEYBOARD) THEN CH:=CHR(13);
    GOOD:= CH IN OKSET;
    IF NOT GOOD THEN WRITE(CHR(7))
      ELSE IF CH IN [' '..'}'] THEN WRITE(CH);
  UNTIL GOOD;
  GETCHAR:=CH;
END;


FUNCTION YES: BOOLEAN;
BEGIN
  YES:= GETCHAR(['Y','y','N','n']) IN ['Y','y'];
END;
```

```
PROCEDURE INIT;
BEGIN
  GETCRTINFO;
  GOTOXY(0,0); CRT(ERASEOS);
  PROMPTAT(10,20,'DO YOU WANT THE RESULTS PRINTED?  ');
  PRINT:= YES;
  F := 55.0; (*FLOPPY DRIVE COST*)
  B := 1.6; (*MEMORY COST IN DOLLARS PER KILOBYTE*)
  T := 0.1; (*DISK TRANSFER TIME PER KB*)
  G := 0.006; (*MEMORY SEARCH TIME PER KB*)
  U := 1.0; (*MOTOR START-UP LATENCY*)
  V := 0.15; (*MOTOR ON LATENCY (ROTATIONAL + AVERAGE SEEK)*)
  W := 0.75; (*MOTOR SHUT-DOWN LATENCY*)
  D := 18.0 ; (*DOCUMENT SIZE IN KB*)
  M := 16.0; (*INITIAL MEMORY SIZE IN KB*)
  X := 64.0; (*MAXIMUM MEMORY SIZE IN KB*)
END; (*INIT*)


PROCEDURE TIMECOST;
BEGIN
  IF D <= M
     THEN
       BEGIN
         TIME := (D*G);
         COST := (M*b)
       END
     ELSE
       BEGIN
         s := TRUNC (D/M) + 1; (*for convenience, s = S + 1 in document 7*)
         IF (D*G/s) > W
           THEN L := U
           ELSE L := V + W - D*G/s;
         TIME := D*(G + T) + L*s;
         COST := M*b + f
       END;
 END; (*timecost*)

PROCEDURE MENU; (* DISPLAY OPTIONS FOR CHANGES*)
    VAR VALUE:REAL;
BEGIN
  REPEAT
  PROMPTAT(10,20,'Enter letter or press RETURN to continue  ');
    CH := GETCHAR(['F','B','T','G','U','V','W','D','X','Q',CHR(13)]);
    IF CH IN ['F','B','T','G','U','V','W','D','X']  THEN
      BEGIN
        PROMPTAT (10,22,'Enter new value  ');
        READ (input,VALUE);
        CASE CH OF
          'F': F:= VALUE;
          'B': B:= VALUE;
          'T': T:= VALUE;
```

```
          'G': G:= VALUE;
          'U': U:= VALUE;
          'V': V:= VALUE;
          'W': W:= VALUE;
          'D': D:= VALUE;
          'X': X:= VALUE;
        END;
    END;
      GOTOXY(10,20); CRT(ERASEOS);  (*CLEAR PROMPT LINES*)
  UNTIL ( CH= CHR(13)) OR ( CH = 'Q' );
  IF CH = 'Q' THEN DONE:= TRUE;
END;


PROCEDURE HEADER (FILEID:STRING) (* PRINT PAGE HEADINGS FOR REPORT*);

VAR I:INTEGER;

BEGIN
  CLOSE (REPORT);
  RESET (REPORT,FILEID);
  WRITELN (REPORT,CHR(12));
  FOR I:= 1 TO 5 DO
  WRITELN (REPORT);
  WRITELN(REPORT,'                    ',
           'VARIABLE MODEL OF MEMORY VS. DISK COSTS');
  FOR I:= 1 TO 3 DO
  WRITELN(REPORT);

END;

PROCEDURE DISPLAY (FILEID:STRING);  (*DISPLAY RESULTS*)
BEGIN
  CLOSE (REPORT);
  RESET(REPORT,FILEID);

  WRITELN(REPORT,'F   ',f:10:4,'  FLOPPY DRIVE COST typical value: 55.0');
  WRITELN(REPORT,'B   ',b:10:4,
          '  MEMORY COST IN DOLLARS PER KILOBYTE typical value: 1.6 ');
  WRITELN(REPORT,'T   ',T:10:4,
          '  DISK TRANSFER TIME PER KB typical value: 0.1');
  WRITELN(REPORT,'G   ',G:10:4,
          '  MEMORY SEARCH TIME PER KB typical value: 0.006');
  WRITELN(REPORT,'U   ',U:10:4,'  MOTOR START-UP LATENCY typical value: 1.0');
  WRITELN(REPORT,'V   ',V:10:4,
          '  ROTATIONAL + AVERAGE SEEK LATENCY typical value: 0.15');
  WRITELN(REPORT,'W   ',W:10:4,'  MOTOR SHUT-DOWN LATENCY typical value: 0.75');
  WRITELN(REPORT,'D   ',D:10:4,'  DOCUMENT SIZE IN KB: 6.0');
  WRITELN(REPORT,'X   ',X:10:4,
          '  MAXIMUM MEMORY SIZE IN KB typical value: 64.0');

  WRITELN(REPORT,'Q                QUIT PROGRAM');
  WRITELN(REPORT);
  WRITELN(REPORT,'         MEMORY  TIME          COST       TIME/COST');
```

```
  WRITELN(REPORT,'          SIZE     (HUNDREDTHS)  DOLLARS     * 10000');
END;


  PROCEDURE CALC(FILEID:STRING);  (*CALCULATE RESULTS*)
  BEGIN
    CLOSE (REPORT);
    RESET(REPORT,FILEID);

    M := 16.0;  (*RESET INITIAL MEMORY SIZE*)
    WHILE M <= X    DO
      BEGIN
        TIMECOST;
        ITIME := TRUNC ((TIME)*100.0);
        ICOST := TRUNC (COST);
        IM := TRUNC (M);
        IF COST<> 0
          THEN ITC := TRUNC((TIME/COST)*10000.0)
          ELSE ITC := 0;
        WRITELN(REPORT,'      ',IM:8,ITIME:14, ICOST:9, ITC:13);
        M := M + 16.0;
      END; (*WHILE*)
  END;(*CALC*)



BEGIN (*MAIN PROGRAM*)
  INIT;
  DONE:=FALSE;
  REPEAT
    GOTOXY(0,0); CRT(ERASEOS); (*CLEAR SCREEN*)
    DISPLAY('CONSOLE:');
    IF PRINT THEN
      BEGIN
        HEADER('PRINTER:');
        DISPLAY('PRINTER:');
      END;
    CALC('CONSOLE:');
    IF PRINT THEN CALC ('PRINTER:');
    MENU;
  UNTIL DONE;
END.
```

THE MACINTOSH PROJECT

DOCUMENT 11 VERSION 0

TITLE: SUMMARY OF OCTOBER 10

AUTHOR: JEF RASKIN

DATE: 10 Oct 79

1.0 APPLICATIONS

1.1 TEXT EDITING

Some form of text editing should be part of the firmware, and will form a major marketing feature. This editing ability should also be at the heart of the operating system, and will be available at all times. A proposal for the design of the text editor will be part of the specification.

1.2 COMMUNICATIONS

It is agreed by nearly all authorities that the personal computer will not achieve a mass market until communications networks are available to them. To help make this project a success, Apple will have to provide software that will ease access to various services, and perhaps adopt some stronger strategies, for example:

1.2.1 INCLUDE CONTRACTS WITH SERVICE VENDORS

The purchase price of Macintosh might include a limited time contract with some vendor of personal computer network and data base services. Vendors might wish to do this as a "come-on".

1.2.2 ESTABLISH APPLE NODES

There is the possibility of building "nodes" (See document M 3) that provide a uniform interface to a number of services, and which handle billing. This is a value added network.

1.2.3 BUILD AN APPLE NETWORK

This would be a communications network, not a data base. It would have access for personal computers as well as ports to other networks and data base services.

1.3 CALCULATOR

A simple calculator language should be provided. This will be explored in a separate document.

2.0 MANUALS

Since this computer is going to a larger and more diverse audience than ever before, the quality of the manuals will have to be especially high. All applications programs should have self-teaching sections, and any languages should also have computer based tutorials.

The manuals should be well constructed physically, typeset, and either wire-O or hard bound. They should make use of color and graphics to a much greater extent than our present manuals.

3.0 SOFTWARE

Macintosh software will have to be written to the highest levels of quality of human interaction. Updating programs in the field, considering the number of units we anticipate selling, will be nearly impossible. A "zero-defect" atmosphere will have to be maintained in software development.

3.1 LANGUAGES

3.1.1 PASCAL

This is our standard language, and it should be provided, on disk, for Macintosh. Since system development is not one of the main application areas for Macintosh, Pascal should be a purchasable item. In spite of the fact that Macintosh is not primarily a programming tool, it should be possible to generate the system software on the machine itself. This "desert island" philosophy assures that we will not build any essential weakness into the software.

for vendors only?

3.1.2 BASIC

It is not yet possible to offer a personal computer without BASIC. We will use whatever BASIC is implemented in Pascal for Lisa and other Apple products, perhaps with the elimination of heavily business-oriented features.

3.1.3 APPLE

The name "Apple" is proposed for a programmable calculator style language that will be the subject of a future report. This language, along with the editor, may be part of the firmware.

3.2 APPLICATIONS

comm, home software.

4.0 HARDWARE

4.1 INPUT AND OUTPUT

There will be a keyboard (upper/lower case, similar to Sara's and Lisa's but with embedded numeric pad), one RS-232 port, one telephone port with modem and daa (auto answer is important, dialing would be nice), a video output, a modulated video output, and some kind of extension bus. There will be no expansion slots per se. A few lines of LCD alphameric display should be an option. It would be advantageous to allow joystick input.

4.2 CPU AND MEMMORY

The CPU choice (at present) is a 6809. There seems to be little advantage to

going to our own processor at this time. Memory is fixed at 64K, consisting of eight 64K dynamic RAMS.

4.3 CASE

Since the requisite circuitry can fit onto a 48 sq in board, it is possible to have the computer not much larger than a keyboard alone. Figure 7 shows one possible case, and figure 6 shows how it might be arranged internally.

4.4 POWER SUPPLY

The power supply will be external via a wall-mounted transformer thus allowing the option of a battery supply.

4.5 DISPLAY

If a configuration such as figure 7 is used, a LCD panel should be mountable in the lid. A separate monitor, in a matching and perhaps attachable case should be provided, as in figure 5.

4.6 DISKS

It is important to open a project to produce a low-cost, even if relatively low performance diskette drive. One way it could be integrated into the system is shown in figure 5. It is essential that some form of mass storage be made available to this system.

4.7 PRINTER

Again see figure 5. It is essential that a printer be made available to this system.

THE MACINTOSH PROJECT

DOCUMENT 12 VERSION 1

TITLE: CONCERNS ABOUT USING THE TELPHONE WITH PERSONAL COMPUTERS

AUTHOR: JEF RASKIN

DATE: 9 Oct 79

The ordinary telephone lines are the only bi-directional electronic communication links between most people. This is not only true for the United States of America, but for a significant portion of the rest of the world as well. The telephone lines are therefore the most obvious and accessible means for implementing inter-computer communications at low cost.

The technical problems have been solved, and electronic interfaces between computers, terminals, data acqusition and display devices and the phone line are inexpensive, often costing as little as a few months telephone service. Nothing special is required of the telephone service in order to use these devices: the telephone system does not distinguish voice from digital signals encoded as sequences of tones (or combinations of tones). In fact, it uses certain tones itself to establish connections, and even to do some bookkeeping.

The bandwidth of the telephone service imposes some limit on the speed of transmission. Inexpensive interfaces operate at a maximum of 30 characters received or transmitted per second. More expensive interfaces could run at, say, four times this rate over the same lines. Such faster interfaces for personal use are still a few years away.

The main technical difficulties in using the phone network for personal computer communications is in adopting protocols that will allow computers to speak to one another. This problem is being addressed by a number of groups such as the PCNET, to name one among many. Assuming that this problem can be solved to a point where such communication becomes commonplace, or if the current time sharing and data base services proliferate to the point where individuals begin to use them as individuals (instead of using them exclusively in connection with their employment or studies), we find another potential problem. The telephone system might start to move to disallow such use.

At first it is not clear why the telephone system might oppose personal computer communications. It would seem that it would only mean increased use, and thus increased revenue. Dr. Gammill, of the Rand Corporation (in his Position Paper on Personal Computers in the 1980's) and others have suggested that the telephone company might seek to limit or control computer useage in order to maximize income by charging a higher rate for computer transmissions. Dr. Gammill points out that "from the point of view of the phone companies, personal voice communication is under-charged due to regulation of that market", and since the tarriffs only apply to voice communication, new tarriffs, at (presumably) higher rates would be applied to digital communication. This would require that the phone company have special

equipment that can distinguish between the two grades of service.

I would like to suggest that there is a technological reason that the telephone companies might be concerned with digital use of a system intended for human communication. The phone system is based on a statistical model of phone use. There are not enough lines and interconnections so that all possible non-conflicting calls can be made at once. The amount of equipment actually installed is based on assuming a certain percentage of the possible calls are being made at any time, and that calls have a certain distribution of lengths. Some calls last just a few seconds: "Hi, Jean?" "Hello Mary." "I'll be over in five minutes." "See you then, bye." "Bye." Others last longer.

The amount of equipment and personnel the telephone company needs depends on the maximum acceptable number of calls that cannot be completed due to lack of equipment, the total number of calls, and some statistics on the length of those calls. The problem with allowing digital transmission probably has little to do with the increased number of calls due to such use. In the next few years the number of personal computers in use will remain under ten million, with only a percentage of these being equipped for phone transmission. But there are over 100 million phones in daily use. So the number of calls made for the sake of a computer connection will remain insignificant for the time being. The same cannot be said for the length of those calls.

As a very simple example, assume that there are 30 calls per hour (at random times) on a system that can handle one call at a time. Also assume that the system requires no time at all to reject a call when the line is busy. If each call lasts one minute, then the probability that a given call was placed without waiting is about one half (it depends on some other factors such as the delay between retries). As these calls double in length, the probability that a given call was placed without waiting gets extremely small. If the calls are longer than two minutes in this example, they won't all fit--some calls can never be made. Notice that the number of calls has stayed the same.

I suspect that this phenomenon is one of the things the telephone companies are concerned about. It is not the number of calls, but the large increase in average length that may well cause problems.

What must not happen is that the users of personal computers get into a cat and mouse game with the phone companies. A possible senario is this: a phone company sets up a special, higher rate for computer use. They add a circuit that detects the usual modem (the modem is the device that attaches a computer or terminal to a telephone line) frequencies and charges accordingly. At the same time they apply to the FCC to make those frequencies mandatory (ostensibly to help promulgate standardization and the free interchange of data). The computer manufacturers make a modem that "sounds" to the phone company more like the voice, so their detectors don't work. The phone company builds a better detector, and begins to throw in random .15 second pauses that interfere little with speech but play hob with digital transmission. The computer buffs respond with error-correcting codes that correct for small pauses, and make still more voice-like modulation. The phone company could respond with a rate scheme that vastly increases the cost per minute after 10 minutes... This could be an expensive and counter-productive war.

If we have the various utitilty commissions and the Federal Communications Commission involved it may be years before true personal computer networking gets under way. There are various strategies that may be applied-now: attempts to have legislation passed that will not allow the telephone company to discriminate based on the content of a telephone call (if they start with computers, will they eventually get the right to charge differently for, say calls with good news and calls with bad news? Do they have the right to listen in on a call at all to determine its content?) One might argue that deaf people can communicate via terminals over the phone and that they should not have to suffer a higher rate.

There might be an attempt to get the telephone company to give a policy decision on the matter--although this could possibly help accelerate their coming down on what we might see as the "wrong" side. Apple Computer is, rather naturally, interested in this situation, and would like to hear from interested parties.

THE MACINTOSH PROJECT

DOCUMENT 13 VERSION 1

TITLE: IMPORTANT POINTS ABOUT MACINTOSH

AUTHOR: JEF RASKIN

DATE: 12 Oct 79

1. The design assumes the existence of a network allowing nationwide communications. Macintosh is a communications device.

2. The cost of the main unit shall be $500, with hopes of lowering that cost to $300 in three years.

3. The design shall have peripherals that attach mechanically as well as electrically, making a unified package.

4. Some functions will be available in ROM, in particular, the network protocols, some word processing, and possibly a simple programmable calculator style language.

5. It will contain a modem/daa, an RS-232 port, a real-time clock, speaker and video and modulated outputs.

6. Disks, printers, a TV monitor, speech recognition and synthesis devices, and battery power supply are examples of possible peripherals, and will not be part of the main unit.

7. RAM size will be fixed, and probably 64K bytes. The processor will be a commercially available product, possibly a 6809.

THE MACINTOSH PROJECT

DOCUMENT 14A VERSION 10

TITLE: THE APPLE CALCULATOR LANGUAGE ~~PRIMER~~

AUTHOR: JEF RASKIN

DATE: 13 Oct 79

{Note: material in braces is notes to myself, or notes to the advanced reader. The reader might well comment that the following language seems similar to the work done by K. Iverson over eighteen years ago. Apple is based on a re-spelling of Iverson's work. Even the name "Apple" might seem derivative from the name he chose for his language.}

CHAPTER 1

IT'S A SIMPLE CALCULATOR

To begin with, this language only uses the numbers, the signs for simple arithmetic, and the large key over on the right side of the keyboard marked with the word "RETURN". Press this key whenever you see the word "RETURN" in a box. Later we will use the other keys too, so as not to be wasteful.

You can tell that it is your turn to type whenever you see an exclamation point (!) sitting at the left edge of the screen. As soon as you begin typing, the first character that you type replaces the "!". The exclamation point is called the "prompt" character, because it prompts you to type something.

Now you know when you can type something. Type

5+2 [RETURN]

The computer responds by showing the result

7

We must say a word about fixing typing errors. You can correct a typing error by backspacing over it and typing the correct information. To backspace, hold down either button marked "SHIFT" and press the space bar.

Subtraction is indicated by the usual minus sign

5-2 [RETURN]

3

{Negative numbers are indicated by preceding them with an underscore (_), e.g. _45.4}

In this manual, any line not followed by [RETURN] is produced by the computer,

in this case, the answer 3.

Multiplication is indicated not with an "X" but with an asterisk (*).

3*4 [RETURN]

12

Division is indicated by a slash (/)

4/2 [RETURN]

2

7/4 [RETURN]

1.75

You can do more than one arithmetic operations in the same line, for example

6/3+2*5 [RETURN]

20

A combination of a number of operations such as this is called an "expression". The rule for evaluating an expression is very simple: Start at the left and move to the right. The expression 6/3+2*5 starts out as 6, the leftmost number. It is then divided by 3. That's 2. Then you have to add 2: that makes 4. Now multiply the result by 5: the result is 20.

If you are familiar with simple pocket calculators, you will recognize that this is exactly how they work. You put in a number, then an operation, then a second number and press the button with an equal sign. In this language you press [RETURN] instead of the equal sign. Evaluating an expression is exactly like doing a "chain calculation" on a pocket calculator where you don't bother to get intermediate results.

If you are familiar with other computer languages, or do a lot of algebra, you might find this strict left-to-right scanning a bit unfamiliar. Actually, if you think about it, this method is more consistent and simpler. This is one of those cases where the beginner with no prior computer experience has the advantage.

If you wanted to add 6/3 to 2*5, you could write the expression

(6/3)+(2*5) [RETURN]

12

Parentheses are used to group items that are to be evaluated together and subsequently used as a single entity.

What would be the value of

55/11+1+12/6*2-3

as interpreted by the computer? The answer isn't 13. The answer isn't 7. In fact there are a whole lot of numbers that the answer isn't.

{ The answer is 3}

Powers of numbers, such as 2 to the tenth, are easily obtained.

2 TOTHE 10 [RETURN]

1024

And to get a square root, you could write

2 TOTHE .5 [RETURN]

1.41

Notice that the answer comes out to two decimal places. This is the standard, or default number of decimal places. You can get almost any number of decimal places you want. Up to a limit of {say, 18}. For example, to get seven places, you would type

-7: PLACES [RETURN]

2 TOTHE .5 [RETURN]

1.4142136

The answer is rounded to seven places. This only affects what the computer shows: inside it knows the truth and remembers as many decimal places as it can. It will continue to show answers to seven places until you give it some other number of places. Here is another example:

10: PLACES [RETURN]

22/7 [RETURN]

3.1428571428

1: PLACES [RETURN]

22/7 + .6 [RETURN]

3.7

0: PLACES [RETURN]

22/7 + .6 [RETURN]

4

{There might be a WIDTH specification as well. Together with the PLACES specification this controls precision.}

CHAPTER 2

CLUMPS OF NUMBERS

Here is an easy one to figure out

17 [RETURN]

17

The simplest expression is just a number. It is not much more complicated to have a clump of numbers, separated by spaces.

34 5 67 [RETURN]

34 5 67

A clump of numbers acts pretty much like a number in an expression. When you add a number to a clump, you add it to each number in the clump. For example

34 5 67 + 3 [RETURN]

37 8 70

The way to figure out how to handle a clump is to start from the left (as always). First you find the 34, then a space, then the 5. Since there was no operation between them, you know that they must be part of a clump. Then you find another space, followed by the 67. Since you have encountered no operation, the 67 is also part of the clump. The next thing you find is a space followed by a plus sign, indicating an operation. This is not a number, so the clump is finished. The operation symbol you just found tells you what to do to the entire clump, in this case you add three to each member of the clump.

Here is an example with multiplication.

1 2 3 4 5 *6 [RETURN]

6 12 18 24 30

Just one more example of using a clump.

2 5 1 -2 [RETURN]

0 3 _1

Remember how negative numbers are indicated. There is a very good reason for distinguishing the operation of subtraction from negative numbers. For one thing, it is never good to use one symbol to represent two different concepts. Furthermore, if we didn't distinguish these ideas, how would you put a negative number into a clump?

Now let's evaluate

7 + 2 .5 1 * 6

Starting at the left you find a 7. The next thing you find is a plus sign, so a clump isn't being formed. Continue to the right and you find a 2. Now you can perform the addition and add 7 to 2 to make 9. The expression is now equivalent to

9 .5 1 * 6

Clearly 9, .5 and 1 form a clump, and then you find a multiplication sign, so the clump is done. Now multiply each element in the clump to get the answer

54 3 6

Parentheses are quite useful. For example you can use parentheses to write

7 + (2 .5 1) * 6

The seven is added to the entire clump, to give

9 7.5 8 * 6

which evaluates to

54 45 48

All this makes the calculator much more convenient for calculations involving a whole bunch of numbers. For example, to convert 32, 50, 100 and 212 degrees Fahrenheit to degrees Celsius, you could write

1: PLACES [RETURN]

32 50 100 212 -32*5/9 [RETURN]

0 10 32.8 100

(We didn't have to limit it to one decimal place, but we did, for appearances sake.) To see how this works inside the computer (and how you can figure out the answer yourself), the first part of this expression is equivalent to (32 50 100 212)-32, or 0 18 68 180. This new clump is then multiplied by five, yeilding 0 90 340 900. Lastly, it is divided by 9 (remember, just work from left to right) to yeild 0 10 32.8 100.

CHAPTER 3

AUTOMATIC CLUMPS

Some of the most useful clumps of numbers are just consecutive integers, for example the thirteens multiplication table can be obtained by

```
1 2 3 4 5 6 7 8 9 *13 [RETURN]

13 26 39 52 65 78 91 104 117
```

You can abbreviate a clump of consecutive integers by use of what is called, in English, the ellipsis. The ellipsis is a kind of punctuation in a class with such things as periods, commas, semicolons...

Those three dots are the ellipsis. To save you a bit of typing, the computer uses two dots. The thirteen's table can be produced by the expression

```
1..9 *13 [RETURN]

13 26 39 52 65 78 91 104 117
```

Of course, you can count backward

```
5..2 +3 [RETURN]

8 7 6 5
```

{Interestingly,

```
5..7..2 [RETURN]

5 4 3 2 6 5 4 3 2 7 6 5 4 3 2
```

Why is this? Because 5..7 is 5 6 7, so the expression is equivalent to

```
(5 6 7)..2
```

which is 5..2 6..2 7..2

Just remember to do things strictly from left to right. Also notice that if real numbers are used where integers are expected, as in 3.6..7.2, they are truncated to integers 3..7}

So far we have done arithmetic between a single number and a clump. When clumps are the same length, we can easily do arithmetic between whole clumps at a time. It is done element by element.

```
(1 2 3 4)+(4 3 2 1) [RETURN]

5 5 5 5
```

But remember, without parentheses, you have to work things through from left to right, element by element.

1 2 3 4+4 3 2 1 [RETURN]

5 6 7 8 3 2 1

This can be checked by starting to form a clump 1 2 3 4. Then you find an operation, which applies to the whole clump. The operation is to add four to each element of the clump. That gives you 5 6 7 8. There is no operation before the next number, so you must still be clumping. This explains the given answer.

{This also shows that juxtaposition indicates concatenation of output.

Another example is

2..6 3..1 [RETURN]

2 3 4 5 6 3 2 1

You might like to try

1..4+4..1, which is equivalent to

(1 2 3 4)+4..1, but this is

(5 6 7 8)..1 or

5 4 3 2 1 6 5 4 3 2 1 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1

Now consider

(1..2)+(1..3) [RETURN]

2 4 3

These two clumps are of different lengths. The shorter clump is always "padded" to make it equal in length to the longer. In this case it is padded with zeros. {The identity element is always used as padding. For addition and subtraction it is 0, for multiplication and division it is 1. A more complete list of padding elements is presented later.}

One last thing to try

3/0 [RETURN]

YOU MUSTN'T DIVIDE BY ZERO

This is an example of a "error message" which tells you that you have done something beyond the pale.

CHAPTER 4

SOME OTHER OPERATIONS

It is sometimes convenient to be able to get just the remainder of a division. In accord with common practice, this is indicated by MOD

```
3 MOD 2 [RETURN]

1
```

If you divide 3 by 2 you do, indeed, get one left over. This next example is educational

```
1..16 MOD 7 [RETURN]

1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2
```

You can get the greater or lesser of a pair of numbers.

```
3 MAX 38 [RETURN]

38

3 MIN 38 [RETURN]

3

38 MAX 3.09 [RETURN]

38

17..21 MAX 19 [RETURN]

19 19 19 20 21
```

With clumps of equal length you can easily do this

```
1 2 4 8 16 MIN (1 3 6 9 12)

1 2 4 8 12
```

And, similarly, you can compare numbers. In this language the number 0 is used to indicate that an answer is false, and 1 indicates that an answer is true. While this may seem a bit strange, we will be able to use these values in some rather neat ways later.

The sign ">" means greater than.

```
1642 > 31.008 [RETURN]

1

5 > 7 [RETURN]
```

0

The equal sign (=) we use unblushingly to mean equals.

3 = 4 [RETURN]

0

6.4 = 6.4 [RETURN]

1

We use "<" for less than, but won't bother with any examples, but go on to use "<=" for less than or equal to and ">=" greater than or equal to.

18 <= 3 [RETURN]

0

7 >= 7 [RETURN]

1

Lastly, in this group, we use "<>" for not equal to, since it has the meaning "greater than or less than" which is the same idea.

5 <> 5 [RETURN]

0

Observe that you can compare to a clump

1 2 3 4 5 6 > 3 [RETURN]

0 0 0 1 1 1

For those who need them, we have the functions AND, OR and XOR.

0 AND 0 [RETURN]

0

1 OR 0 [RETURN]

1

1 AND 0 [RETURN]

0

in all the familiar combinations. For example

0 1 0 1 OR (0 0 1 1) [RETURN]

0 0 0 1

Why didn't we write 0 1 0 1 OR 0 0 1 1?

{Because that would be 0 0 0 0 0 1 1 and not show all the combinations.}

Why didn't we have to write (0 1 0 1) OR (0 0 1 1)?

{Because we evaluate from left to right, but it might not be a bad idea to sometimes use redundant parentheses in a situation like this to make the expression clearer.}

1 XOR 1 [RETURN]

0

CHAPTER 5

SOME SCIENTIFIC CALCULATOR ABILITIES

For those of you that like your calculators scientific, we have sines and cosines and the like. On the typical calculator, if you have a number in the display, and want to find its sine, you press the button marked "SIN" (no religious implications intended). Similarly, in our left-to-right scheme, after you have a number you just type "SIN" and the result is calculated. For example

3.141592654 SIN [RETURN]

0

Here, we are working in radians. If you'd rather work in degrees, you have an option, the way you did with the number of decimal places. You can set the option "RADIANS" to true or false. If it is true, you are working in radians; if it is false, you are working in degrees.

From the sine value above you can conclude that you were working in radians. If you want to work in degrees, you make RADIANS false. It works very much like PLACES did.

0: RADIANS [RETURN]

30

To get back to working with radians, you would type

1: RADIANS [RETURN]

Incidentally, if, at any time you want to find out to how many decimal places the computer will display, you can just type

PLACES [RETURN]

4

Or, to find out if you are working in radians or degrees, you can type

RADIANS [RETURN]

0

The trigonometric functions you have available are

SIN COS TAN ARCSIN ARCCOS ARCTAN

Also you have LOG (which is base 10) and LN (which is base e).

There is even a constant

PI

with the usual value. You can say

PLACES: 3 [RETURN]

1..4 * PI [RETURN]

3.142 6.283 9.425 12.566

Incidentally, what would you get from PI * 1..4?

{3 4}

{The base of the natural logarithms is available. Its name is "E".}

We also have two simple functions, FLOOR and CEILING. FLOOR gives you the greatest integer less than or equal to the given number; CEILING gives you the least integer greater than or equal to a given number.

5.1 17 -5.1 FLOOR [RETURN]

5 15 -6

5.1 17 -5.1 CEILING [RETURN]

6 17 -5

In the same vein we have

22 22.1 22.5 22.51 22.8 ROUND [RETURN]

22 22 22 23 23

and we can truncate

22 22.335 22.8 -5.43 -4.9 TRUNCATE [RETURN]

22 22 22 -5 -4

And here is a strange, but enjoyable function. You'll have to puzzle it out from these examples.

3 PICK [RETURN]

3

4..8 PICK [RETURN]

5

4..8 PICK [RETURN]

4

4..8 PICK [RETURN]

4

4..8 PICK [RETURN]

7

It gives a random choice among the clump presented to it.

There is also one logical operator among these functions

0 NOT [RETURN]

1

1 NOT [RETURN]

0

If you apply NOT to numbers other than 0 and 1, the results may seem strange, so it is not recommended. {The logical operators applied to integers result in bit-wise operations. If the numbers are not integers, they are first truncated.}

Here are some puzzles. Fill in the computer's part.

0 NOT 3 [RETURN]

{1 3}

0 1 NOT [RETURN]

{1 0}

3.1 ⁻5.6 ⁻5.2 3.8 +.5 CEILING [RETURN]

{This evaluates as follows 3.6 ⁻5.1 ⁻4.7 4.3 CEILING which is 4 -5 -4 5}

CHAPTER 6

SAVING FOR A RAINY DAY AND WITH WORDS WE LEARN TO PLAY

Many calculators have a way of storing a number, then later recalling it. This is usually called "memory". In this Apple Calculator, there is also memory. For example, to save the value 5 you merely give it a name, in this case we'll call it "fingers".

5: fingers [RETURN]

Notice that the computer types nothing back to you (except the prompt, which we never show in this manual. It now sits there awaiting your command. And it remembers that the value of fingers is 5.

You have seen that

5 [RETURN]

5

Well, now that you have defined fingers you can type

fingers [RETURN]

5

Of course

13 84 56 72: hike [RETURN]

hike [RETURN]

13 84 56 72

hike -20 [RETURN]

-7 64 36 52

So you see that you can store numbers and clumps of numbers into cubbyholes that have names. Names can be as long as you like; just remember that you will have to type them so you might not want to make them too long. The rest of the rules for names are simple: they must begin with one of the twenty six letters of the alphabet in upper or lower case, they may contain letters of the alphabet, digits, and periods. Blanks are forbidden. It is recommended that names be all lower case. For example

flavor.number.58

is a legal name. It is customary to write names in lower case characters so that they are easily distinguished from keywords, such as SIN or TRUNCATE. You may not use any keywords (which are always upper case) as names.

This flexibility to use names that we devise ourselves makes this calculator a little bit easier to use than most pocket calculators. It sure is easier to remember that you've put the checkbook balance into a cubbyhole named "balance" than to remember that it is in "register 4".

Our calculator doesn't only deal with numbers, but with letters as well. As you know a cat has four legs, but the word "cat" has no legs at all. It has three letters. We distinguish, in English (and most other phonetic tongues) between the way the word is written and its meaning by using quotes. This is true of the Apple language as well. Here are some examples.

"cat" [RETURN]

cat

It should be clear that

fingers [RETURN]

5

"fingers" [RETURN]

fingers

One very useful ability is to be able to find the number of characters in a string.

"fingers" LENGTH [RETURN]

7

Incidentally, this function also works with clumps of numbers

1 2 3 LENGTH [RETURN]

3

3..9 LENGTH [RETURN]

7

Is this answer correct?

{Sure is, even though 9-3 is 6.}

As we have seen, just writing two clumps one after the other gives a longer clump

2..6 7..5 [RETURN]

2 3 4 5 6 7 6 5

the same thing happens with letters

"had","dock" " smells" [RETURN]

haddock smells

This is called "concatenation", the bringing together of two or more objects side by side. Notice that the space between the quote and the word "smells" is not accidental.

Letters between quotes are called strings. You can concatenate strings even if they have been given names.

"abcd":alpha [RETURN]

"efghij":bet [RETURN]

alpha bet [RETURN]

abcdefghij

Every name, whether you use it or not, has a value.

elephant [RETURN]

0

That is because any name that you haven't given a value to is zero. If the name is used as part of an expression involving strings, and it hasn't been given a value, then its value is a string with no characters. This is called the "null string" in the jargon. In this example the name "alphabet" has not been given a value, so it is the null string and has no effect when concatenated to the string named "alpha".

alphabet alpha [RETURN]

abcd

CHAPTER 7

AN INTERLUDE: A SUMMARY AND RE-PRESENTATION

The prompt character is "!", and it is overridden by the first typed character. User input is terminated by [RETURN].

The dyadic operators (those that come between two quantities and do something involving both of them to result in another qantity) are

+ - * / TOTHE .. MOD MIN MAX < = > >= <= <> AND OR XOR INSERT

0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0

The numbers under each operator is the padding value in case the operation is attempted between clumps of different lengths. Strings are padded with nulls.

The special variables (names that have a meaning to the system) are

PLACES RADIANS PI E

The monadic operators (those that come after a quantity and do something involving it) are

SIN COS TAN ARCSIN ARCCOS ARCTAN NOT LOG LN FLOOR CEILING ROUND TRUNCATE PICK LENGTH

Names may begin with any letter, and may have letters, digits and underscores. The assignment operator, which should be a right-pointing arrow, is the colon because ASCII doesn't have a right-pointing arrow. Expressions are evaluated strictly from left to right, but the order may be modified with parentheses. Any printing characters, space, and return may occur in strings. Much more is to come.}

CHAPTER 8

A FEW MORE STRING OPERATIONS

Strings can be converted to clumps of numbers and vice versa. A table {the standard ASCII codes} shows what numbers go with what characters. For example

```
"ABC" NUMBER [RETURN]

65 66 67

68 69 70 LETTER [RETURN]

DEF
```

If you give the LETTER function a number too high to represent any letter in the clump, then it gives back the null character.

{It is nice if

```
"" NUMBER [RETURN]

0
```

and vice versa}.

You can compare two letters

```
"a" > "b" [RETURN]

0
```

You can compare strings for alphabetic order

```
"jello" <= "hello" [RETURN]

0 1 1 1 1

"jello" > "hello" [RETURN]

1 0 0 0 0
```

just as you could for clumps of numbers.

It is sometimes useful to deliberately change the value of a numerical quantity into a string, for example

```
34 STRING "b" [RETURN]

34b
```

And to go the other way, there is the VALUE function

```
"2" VALUE +1.1 [RETURN}
```

3.1

Single quotes are used when the string has double quotes in it

```
'He said, "Do not shoot, I have a cold".' : line.from.movie [RETURN]

line.from.movie [RETURN]
He said, "Do not shoot, I have a cold".
```

said, Don t shoot, I have a cold

CHAPTER 8

INSERTING OPERATIONS INTO CLUMPS

Let's say you wanted the odd numbers between 1 and 19 inclusive. It is easy to get the even numbers between 2 and 20

```
1..10 * 2 [RETURN]

2 4 6 8 10 12 14 16 18 20
```

if you then subtract 1, you get the odds (in this example they are first stored under the name "odds" and then displayed.)

```
1..10 * 2 - 1: odds [RETURN]

odds [RETURN]

1 3 5 7 9 11 13 15 17 19
```

Now, say you wanted to find the sum of all the odds in this range. You could write the expression

```
1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + 19
```

which does the trick. But there is an easier way. You can automatically insert any operator that works on two entities (called a "dyadic" operator) between all the elements of a clump. The operation is called "INSERT". After the word "INSERT" you place the operation you want inserted into the clump. For example, the sum of all the odd numbers above can be easily obtained by

```
odds INSERT + [RETURN]

100
```

You can get the largest of a clump of numbers by using INSERT MAX

```
34.667 92 3 _45.09 INSERT MAX [RETURN]

92
```

The smallest of a clump?

```
odds INSERT MIN [RETURN]

1
```

To demonstrate another application of INSERT, we introduce the monadic function ODD.

```
11 ODD [RETURN]

1
```

said, Don t shoot, I have a cold

34 ODD [RETURN]

0

ODD determines if a number is odd. Here is a clump

·· 12 34 55 18 67 31 24 : bunch [RETURN]

For convenience, we have given the clump a name. It is simple to find how many odd numbers there are in the clump.

bunch ODD INSERT + [RETURN]

3

This tells us that there are three odd numbers in the bunch. Step by step, it works like this:

bunch ODD

evaluates to

0 0 1 0 1 1 0

and then

0 0 1 0 1 1 0 INSERT +

evaluates to

0 + 0 + 0 + 1 + 0 + 1 + 1 + 0

which is

3

How many even numbers in the clump?

bunch LEN - bunch ODD INSERT +

Or, you can use the handy EVEN operator

bunch EVEN INSERT +

We can ask, for example, how many examples of the letter "e" occur in a text.

"The quality of mercy is not strained." = "e" INSERT + [RETURN]

3

ABOUT CLUMPS

1. ETYMOLOGY AND PHYLOGENY OF CLUMPS

Clumps are a data structure. I searched about for a while to find this name. The best English word for them is probably "lists". A shopping list is an excellent model:

One Useless

Three boxes of Dreadful

Two packages of a dozen Expensives each

A pound and a half of Unobtainium

A Gimcrack

A dozen Shoddies, or a bag of Overpriced, whichever is cheaper

However, the word "list" has been pre-empted by computer science to mean a clump with pointers between the items in the clump. A clump is something like a vector, but the word "vector" is scary to many beginners, besides, the terms "vector" and "array" in most technical parlance usually imply elements of like kind. The same problem applies to "matrix". A clump is not a set, since it's elements are ordered: it is an ordered set, but that name is too clumsy. I can't use many other English words, such as "group" or "conglomeration" or "accretion" on the grounds of overuse, lack of euphony, or excess length. Thus, "clump". This word also has the feeling of "sticking together", which is how clumps behave in expressions. (The verb "to clump" means to put things together in generally irregular ways.)

The word "clump" has a certain informality about it. This corresponds nicely to the way they are used. Clumps are not declared, they just happen as they are needed. Clumps are not fixed in size or composition. However, they are ordered; each clump has, at any moment in a program's execution, a fixed number of elements, each associated with an ordinal number that gives its position. ~~Clumps are linear~~.

A clump may have an element that is, itself, a clump. There is no null clump. If you create a clump, it is assumed to be something, depending on context. The smallest clumps are things like the null string, a single numerical value (possibly zero), or just one other clump. Clumps are the only named objects in this language. Programs are clumps.

I fear a strong abreaction from structured programming devotees. I wish to say to them: live and let live. Programming languages and structures are tools, and each tool has its rightful place. Computers are not only to be used by carefully disciplined coders in phalanx. They are also to be used by undisciplined non-programmers who care to solve a problem in the quickest, most haphazard way they can. You cannot argue with human happiness, and a

formal, structured, declared approach is not always appropriate. If you think that it is, then you have a bit of the martinet or despot in you, and would have everybody toe the line and write magnificent, portable, durable code. If you are true to form, you probably won't let anybody write a quick note in pencil to a friend, but require them to have it typeset, on 100% rag paper, and sent by liveried messenger.

Apple is an informal communication between a user and his or her computer, it is quick, impermanent, friendly, and useful. Do not fret, my structured friend, you will not be put out of business, nor threatened by this any more than I, a professional writer by trade, am threatened by universal literacy. Such literacy just means a wider audience for my professional talents. So will it be for programmers. The more people know how to program and use computers (the two are not synonymous) the more customers there will be for good programs. The more a person knows about programming, the more he or she realizes the quality of a fine program. Apple is no impediment to the writing of fine software.

2. THE ELEMENTS OF CLUMPS

The elements of clumps are of two kinds

A. Numbers

A number is a sequence of digits and, optionally, a decimal point. There is no notation for powers of 10 such as 34.2E14. Such numbers can be represented externally using the usual notation for powers. The output routines will do this when necessary. There is no problem on input, since expressions can be used wherever a constant may be. The internal representation of numbers is not a really a concern of the language design, except that it is expected that there will be at least 18 decimal digits of precision. Numbers will generally be stored as integers unless a real representation is required. The precision of the representation may also change.

B. Characters

Characters are those codes that can be generated by the computer. Some of them (at least the usual set of 96 ASCII/ISO characters) can be shown on a display or printed. They are:

ABCDEFGHIJKLMNOPQRSTUVWXYZ (26 uppercase letters)

abcdefghijklmnopqrstuvwxyz (26 lowercase letters)

12345678890 (10 digits)

\:-@^[;],./|!"#$%&'()0*=`~{+}<>?_ (33 special characters)

(1 space)

Strings are no part of the language. There are just clumps of characters.

3. A FORMAL, BEEN EFF, DEFINITION OF CLUMPS

A formal item in braces means that it may appear zero or more times.

digit := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

decimal point := .

integer := <digit>{<digit>}

number := <integer> | <integer><decimal point> | <decimal point><integer>

| <integer><decimal point><integer>

character1 := <any of the 96 characters shown above except the single quote>

character2 := <any of the 96 characters shown above except the double quote>

letter := <any uppercase or lowercase letter>

name := <letter>{<letter> | <digit> | <decimal point>}

space :=

string := "{<character2>}" | '{<character1>}'

element := <string> | <number> | <name>

clump := <element><space><clump>

Note that integers and strings per se do not appear as separate concepts in the user's view of the language, and that exactly one space is specified between clumps.

4. EXPRESSIONS INVOLVING CLUMPS

We first define the assignment operator which has the form

<clump> : <name>

Thereafter the name stands for the clump. The value includes type and structure information. This is different than most languages where this information is associated with the name, rather than the value.

Every clump has a length L. L >= 1. This length is the number of elements that comprise it.

There are two kinds of operators in this language, monadic and dyadic. Operators may be distributing, expanding or reducing. A distributng operator operates on each element of a clump and leaves a different clump of the same length. A reducing operator reduces a clump to a smaller clump, usually an element. For example, the LEN operator reduces a clump to a number, the assignment operator (:) reduces a clump and a name to a name. The sequence operator (..) takes two numbers A and B and expands them into a clump of (B-A+1) elements.

A monadic operator occurs after the clump on which it operates. A dyadic operator occurs between two clumps.

monadic operator := LEN | (to be filled in)

dyadic operator := + | - | * | / | TOTHE | MOD | : | .. | (to be filled in)

expression := <clump> | <clump><monadic operator> |

<clump><dyadic operator><expression>

said, Don t shoot, I have a cold

CHAPTER 9

SELECTING ELEMENTS FROM A CLUMP

By this time, you are quite experienced, so a few examples should suffice.

"abcde" [1] [RETURN]

a

"abcde" [2] [RETURN]

b

"abcde" [5] [RETURN]

e

"abcde" [2 5 1 4] [RETURN]

bead

12.5 99 63 [2] [RETURN]

99

12.5 99 63 [5] [RETURN]

0

"abcde" [_3] [RETURN]

This last example evaluates to the null string.

ABOUT CLUMPS

1. ETYMOLOGY AND PHYLOGENY OF CLUMPS

Clumps are a data structure. I searched about for a while to find this name. The best English word for them is probably "lists". A shopping list is an excellent model:

One Useless

Three boxes of Dreadful

Two packages of a dozen Expensives each

A pound and a half of Unobtainium

POSTSCRIPT AND DIATRIBE

To the Reader:

To those of us not familiar with APL, this language may seem a bit odd. To those of us who know APL, this language may seem a bit odd too, but in a different way. One of the major aims of the designer of general purpose computer languages should have is to improve the quality of life of the programmers who will use it. The important item here is the programmer or user, not the system. The whole idea of a higher level language is this: programming time is very dear, let's use the computer to minimize it. As computers become cheaper, this strategy becomes ever more valuable. Computers are now very cheap. In the room in my house where I am writing these words (on a computer system) are no less than three computers. The most expensive piece of equipment, on an hourly basis, is me.

It is with the utmost trepidation that I dare to attempt to design a new computer language. New computer languages are plentiful and easy to come by, many old language are adequate and well entrenched. My motivations are simple, and relatively pure. The advent of personal computers gives me sufficient excuse, and a new potential audience. None of the major languages was designed for use by untutored individuals using computers in their personal lives. And, as I shall attempt to show, none of them is truly suitable to this arena.

My main motive is to give people as much power for as little effort as I can conceive. I have been programming computers for most of my life, and of the many languages I have used, one stood out as permitting me to accomplish, in a given amount of programming time, more than any of the others. Subsequent studies by others have shown that I was not unique in my appreciation of this language. But before going on, perhaps a brief examination of the broad characteristics of computer languages, from the point of view of programmer effort, and why each language is popular is in order.

A VIEW OF EXISTING LANGUAGES

One language is "higher" than another if its primitives are constructs in the other. Sometimes "higher" depends on the task. For many processes involving lists and grammars, LISP is a very high level language. It is not quite so high level a language for some engineering applications. The major directions in higher level languages might well be summed up in these words: ALGOL, BASIC, COBOL, SNOBOL, LISP, APL.

FORTRAN and PL/1 are derivatives or fellow travelers of ALGOL, as is Pascal. As declared languages they are inherently poor at simulating human trains of thought. I am well aware of the advantages of declarations, and take joy at the improvement in control structures of, say Pascal over FORTRAN. But these are all improvments of a detailed, nit-picking sort. What languages should do is remove the nits, and minimize the detail you must think about when solving problems. What Pascal's data structures, declarations, and control structures have done is to make those details more explicit, and thus cleaner. It is a real improvement, but unfortunately leans in the direction of discipline and rigidity.

BASIC derives from the same traditin as FORTRAN, standing out only because of its interactive environment. BASIC is a very weak language, and is being shored up in many ad hoc ways. Any student of programming languages can easily poke holes in it. Nonetheless, it will continue to be used for many years, which is a tribute to its environment, not its internal design. BASICs well- deserved popularity demonstrates that minimizing programmer effort and time is often more important, when it comes to purchasing a language, than the language's features as described in the spirit of traditional computer science.

COBOL, with regard to general problem solving, is weak and inefficient of programmer time. Its widespread use is proof that money and marketing strategy can be as important as rigor and technical perfection. I have never, in any study of the design of computer languages, seen a chapter (or even a paragraph) devoted to the importance of marketing to the general adoption and use of a language. When COBOL was introduced, computers were "IBM machines", and IBM pushed COBOL for business applications. It actually has features that are advantageous in some business applications, but it never would have succeeded on pure merit. It is not the only product in the world that owes its success to PR.

SNOBOL, which has some novel features, was first designed for string processing, at which it is a very high level language. Like many other languages, it will retain a small and devoted following. Besides the string processing features, SNOBOL is rather consistent in design, and its program structuring technique is interesting. I mention it only because it exemplifies a language that was designed for a special, limited purpose which later (because of its excellent design) became more widely used than one would have expected given its intended audience.

LISP has been given lisp service above. It is an interesting corner of the programming universe, and is relatively efficient of programmer time. It suffers from an overdose of recursion and a lack of approachableness for common, everyday activities.

WHY APL IS SINGLED OUT

Most of my praise and commentary must be reserved for APL. In terms of stating many algorithms, APL is a much higher level language than the ALGOL group. For example, APL builds most loops into its expressions, whereas the ALGOL group of languages (as well as most others) require the programmer to pay attention to the details of each loop. APL was designed with an interactive human environment as part of its specification. It was years ahead in this regard.

Pascal is often pointed out as a modern, well designed language. It is, to be sure, tolerably efficient. Its efficiency is due to many design choices aimed at making life easier for the compiler writer and system programmer. The user is often ignored. The programming environment is not even the least part of the design (although the assumed environment had a subtle influence). This brands Pascal as practically reactionary rather than forward looking. Wirth, Pascal's designer, invented it (in his words) to foster a "systematic discipline" of programming that would be implemented so as to be "reliable and

efficient on presently available computers". Do we want to impose a "systematic discipline" on the purchasers of personal computers? The very opposite is true. Are we really concerned with saving the computer occasional microseconds? No, we are concerned with saving the human hours of thought and work.

APL, from its very inception, was interactive as far as a Selectric terminal would allow. The human being was treated as supreme, the machine as his or her servant. I, being a human and not a machine rather prefer this orientation.

In a number of ways, such as consistency, elegance, and power at handling array and matrix structures (of numbers or strings) APL is unmatched. On the other hand, APL uses a eccentric notation replete with Greek characters and a hodge-podge of unique symbols that make it look very strange indeed. Iverson was trying to invent a new notation for mathematics, and mathematicians never quail at creating obscure symbols. This notation has been a millstone about APL's neck. There is also a feeling that APL works from right to left, which people find strange. This accusation is true, but Iverson was only being consistent where most of us (and most of our programming languages) are inconsistent. For example, look at expressions such as

```
SIN (TAN (COS (3 * (PI/2))))
```

To evaluate this, you start nearly at the right, evaluate PI/2, then multiply it by 3, then take the cosine, then the tangent and finally we get to the left and take the sine. See! you do work from right to left.

Iverson, realizing how troublesome the rules of precedence are (he is so right, as anybody who has taught programming knows), and being consistent, has everything evaluated from right to left in expressions. And APL is almost all expressions. As a consequence, APL is easy to teach, easy to learn, easy to parse, and looks most peculiar to people brought up on other computer languages. APL is also notoriously slow to execute, and until Abram's work, it seemed impossible to make it efficient.

With these black marks against it, it is only APL's extreme learnability, interactive environment, great power, and human programming speed that have kept it alive at all.

Another comment that one often hears is that an APL program is unreadable, undocumentable, and only can be used by the person who wrote it. All this is usually true. Of course, if the main use of a personal computer is for "throw-away" programs that are used but once, or perhaps only a few times (and this is often the case) then these objections have little force.

These problems are not inherent to the idea of using expressions as does APL. They are partly due to the style in which APL is taught, and mostly due to the particular embodiment of the language, not to its concepts. Most APL programmers take little trouble to make programs readable, as there is a temptation to do as much as possible in as few symbols as possible. The mechanism for introducing comments is extremely difficult to type, and its use is not encouraged by any books on the subject. Most of Iverson's pamphlets don't even mention comments.

Given the good and bad of APL, Apple is an attempt to keep the good, and ameliorate the bad. Apple uses the common subset of the ASCII and ISO character sets rather than Iverson's special character set. Many unique symbols are replaced by readable keywords. A typical example is that Apple uses SIN and ARCTAN rather than 1o and -3o to represent the sine and arctangent functions respectively.

I must admit that English keywords diminish a languages international appeal to a slight degree--BASIC, FORTRAN and Pascal usually keep their English keywords in most non-English speaking countries.

Out of context, some of Iverson's conventions may seem counterproductive or even stupid. This is not the case. If you had to sit in front of a 15 character per second terminal, you would welcome shortening a function name by even a single letter. The personal computer's fast screen gives the present design much freedom that Iverson did not have. Even the comments made above about Pascal should be seen charitably: Wirth was countering languages that did not aid in teaching rigourous, professional programming practices and that included the concept of structured data types. His primary environment was a large, card-entry, batch processed computer center. Pascal, too, is a work of considerable brilliance. In the kinds of world Wirth was considering for Pascal, where hours separate runs and computer time is precious, Pascal is superior to its predecessors (including APL). Just observe that even when it isn't explicitly mentioned, the programmer's environment conditions the design of a language to a great degree.

SOME DETAILS OF APPLE, JUSTIFIED

Jean Sammet, in her book "Programming Languages", after some quite positive comments, said of APL, "I cannot become enthusiastic about a language that has this notational complexity " James Martin, in his book "Design of Man-Machine Dialogues" observed that "It was easy for a person who was not a professional programmer to become hooked on APL." These statements are in contradiction. If the notation of APL is truly complex, then why is it so easy for non-programmers? Because they have no prejudices to work against. Apple is an attempt to cure the disease of notational complexity that Sammet complains about, while retaining and enhancing the features that make APL so addictive to novices.

One of the easiest things to do with APL to make it more useable is to reverse the order of evaluation. As this document attempts to show, this should make Apple especially easy for the millions of people who have used pocket calculators. The next thing that was done was to change the peculiar symbols into keywords, which had the beneficial side-effect of totally eliminating the tedious requirement in APL for backspacing and overstriking to create new symbols.

Iverson's confusing use of a single symbol to represent two related (sometimes very ingeniously related) functions, one of which is monadic and the other dyadic has been eliminated in favor of simply having two separate function names. A bit of mathematical insight is lost in favor of much mnemonic convenience.

A ROSETTA STONE

A number of ideas from other languages have been adopted. The alternating use of single and double quotes from SNOBOL, the range notation from Pascal, and others that the knowledgeable reader will notice. It might be interesting to extend the example on page 64 of Martin's "Design of Man-Machine Dialogues" for inclusion here. The same simple program is presented in a number of different languages. In each case the program produces the average of a list of real numbers. In all but Apple and APL, the list has an limit of 100 items. In Apple and APL there is no limit beyond the amount of available memory.

I have taken the liberty of modernizing the BASIC example, and have written examples in Pascal and Apple.

FORTRAN

```
      DIMENSION X (100)
      READ (5,10) N, (X(I), I= 1,N)
10    FORMAT (I5, (E15.2))
      S = 0.0
      DO 9 I=1,N
 9    S = S + X(I)
      A = S/N
      WRITE (6,20)A
20    FORMAT (E15.2)
      END
```

BASIC

```
 20 S = 0
 30 READ N
 40 FOR I = 1 TO N
 50 READ X
 60 S = S + X
 70 NEXT I
 80 A = S/N
 90 PRINT A
100 DATA {the data on which the program would operate goes here,
110       and perhaps on the next few lines.}
120 END
```

Pascal

```
PROGRAM AVERAGE;
VAR ITEM,N,SUM: REAL;
SUM := 0.0;
READ (N);
FOR I := 1 TO N DO
  BEGIN
    READ (ITEM);
    SUM := SUM + ITEM
```

```
  END;
WRITELN (SUM/N);
END.
```

PL\I

```
DCL  X (100)  INITIAL (0);
GET LIST  (N, (X(I) DO I = 1 TO N));
PUT LIST (SUM (X)/N);
```

APL

+/X÷⍴X < □

(Using p for rho, : for the division sign, <- for the left arrow, and [] for the box we could write it +/Xp:X<-[]. This is done just in case the program segment above has not been filled in by hand with the actual APL symbols.)

APPLE

```
INPUT: many.numbers
many.numbers INSERT + / (many.numbers LENGTH)
```

COMMENTARY ON THE LANGUAGES

Which ones are immediately readable to you depends on what you are used to. I remember being told that a Pascal program was really easy to read, even if you didn't know Pascal. At the time, I didn't know Pascal, and I couldn't read it, or even figure out much about it. Now it looks as clear as good English to me. So do all the other examples.

One obvious difference in the examples is their lengths. For the purposes of these counts, I have not counted extraneous spaces introduced for the purpose of clarity, and all variable names are counted as one character in length. The DATA statements in BASIC have also not been counted. Taking a BASIC program as being of length 1, then FORTRAN is 1.5, Pascal 1.2, PL\I .85, Apple algorithm being represented. Pascal and PL\I improve strongly as things get more complex. FORTRAN, APL and Apple (again, relative to BASIC) improve slightly with increasing complexity.

SOME ADDITIONAL FEATURE OF APPLE
(a potpourri of notes to myself)

Compared to APL, string operations are given more prominence in Apple. Graphics are an integral part of the language, although they do not appear at all in APL. Some notations from other languages have been substituted for Iverson's, especially notable is the use of Pascal's abbreviated ellipsis for the iota. This also allows expressions to be more readable when the range does not begin with 1. (Suggestion of R. Kelly). The PROGRAM function is based on APL's evaluate. Apple makes decisive use of upper and lower case.

THE MACINTOSH PROJECT

DOCUMENT 15 VERSION 0

TITLE: MASS STORAGE PRINTER/FACSIMILE DEVICE

AUTHOR: JEF RASKIN

DATE: 22 Oct 79

1. INTRODUCTION

On the 14th of this month I was trying to find a less expensive mass storage device for the Macintosh project. The bar-code reading wand is one of the least expensive computer input devices, but is limited by the operator's dexterity. I was also considering the possible requirement for a printer--it is agreed that it would be best, although possibly uneconomic, if Macintosh could have both a mass storage device and a printer. Any low cost printer would have to be a dot matrix based device given the present state of the art. I realized that such a printer could print bar codes.

My next thought was to eliminate the need for operator dexterity. First I thought of a plastic guide with slots that the wand could run along. Then, considering the sweeping motion of the hand moving the wand through the slots, it became apparent that the printer head is already making such a motion, and that the wand's sensor should be incorporated into the printer's head. It occured to me that the design could also affect read-after-write checking (a rarity with printers), and that it could also be used as a facsimile machine--which fits in wonderfully with Macintoshe's proposed communications abilities.

The possibility of providing the public with a very low cost computer that includes both mass storage facilities and a printer at very low cost is what makes this idea intersting at all.

2. DATA PACKING DENSITIES

Since, by using pin-feed or by some feedback from the sensors, the motion of the head across the paper can be done with some accuracy, it would be possible to have more than a simple bar code, and the head could read a number of bar codes in one character height--perhaps even read each of, say, eight dots by means of eight photo sensors (and, possibly, a bar-shaped LED to provide constant local illumination). This provides (given a 5 by 8 character matrix) 6 bytes in the space of one character. Given compression due to the fact that pages of a document are not filled uniformly, this means that a document could be stored in 1/8 the number of pages it would take to print it in natural language form.

If 80 characters can be stored across the 8 1/2 inch page, and 68 lines of such characters fit into the 11 inch height, then 32640 bytes can be stored on a standard page, with reasonable margins. This means that the 64K memory of the Macintosh will fit on a bit over two pages. If, as Victor Bull indicates, a few more characters can be put on a line, 64K bytes could fit on two pages.

3. DATA TRANSFER RATES

While the data packing densities indicated here may not be achievable in practice, they do give us a believable upper limit. They also give us an upper limit on data transfer rates. Without any changes in the mechanism, the data in and data out rates will be equal. At one line a second, that's 480 bytes per second, or 28,800 bytes per minute. It would take 2 1/2 minutes at this rate to dump or fill the entire 64K memory.

4. POSSIBLE OTHER TECHNOLOGIES

At present the simplest printers are thermal and electrostatic discharge (ED). I considered the possibility of reading the aluminized paper produced by an ED printer by merely sensing the change in resistance caused by the burning away of the aluminum coating. Using an ohmmeter, and a probe made of #26 wire in a suitable holder, I found that there is an increase of resistance on the burned areas. However, it is a spotty effect, and seems an unreliable basis for a mass storage device.

It is possible that this technology, which is potentially faster than thermal printing, given a head designed to optimize resistance readability, could result in a superior storage device. Interestingly, the same head could possibly both read and write the data. This could be the lowest cost approach, excluding the cost of the paper.

5. USE OF PRINTING PRESS FOR DISTRIBUTING SOFTWARE

It is no new idea that once a printable, machine readable code is available, the printing press now becomes an instrument for the dissemination of software. The use of a resistance reader becomes more difficult in this case (although by no means impossible), but the benefits of an optical reader are increased. This possibility may well be the strongest motivation to using a printer/reader, since software duplication and distribution is a major problem at the present time.

6. LIMITATIONS ON ITS USE

It is not clear whether such a device, which does not even have the power of a floppy disk, would be seen as an inducement to purchase a Macintosh class product. The potentially low cost is attractive, but operation of a printer/reader would be clumsy compared to a disk. To switch from writing to reading would require removing the blank paper and inserting the written paper. Changing back would be as difficult, except that the paper could be fed through instead of pulled back. Making the design largely self-loading could ease this problem, although paper handling is apparently not an easy engineering problem, considering some of the printers I have seen.

Unlike a disk, it would be complicated to use a printer/reader to store intermediate results (although not impossible, given bidirectional paper feed), and it would probably be unreliable compared with a disk. It would be most practical in the roles of archival storage, and program and data distribution.

The major question is: would the cost advantage, software distribution advantages, and facsimile ability overweigh, in the user's mind, the difficulties caused by having to use the device as the sole mass storage medium in a personal computer system?

7. MECHANICAL DIFFICULTIES

There are a number of engineering questions that have to be answered before it is known if this technology is practical. The maximum data transfer rates and packing densities cited above, if achieved, are practical. Much less than that, even by a factor of two, and the concept becomes too clumsy.

7.1 Accuracy of tracking, between writing and reading.

At 12 pitch, the width of a character, including the intercharacter space, is 0.083 inches. The width of a single dot is 0.014 inches. Assuming a 5 by 7 character in a 6 by 8 matrix, assuming 1:1 aspect ratio for the dots, and assuming a reading circle of 0.005 inches, the maximum absolute positioning error is plus or minus 0.01 inches both horizontally and vertically. While these assumptions may not be exact, this is certainly in the right range. It is also in the range where the humidity coefficient of expansion of the paper becomes a problem. Accuracies on this order are not possible with a friction feed paper drive.

7.1.1 Pin feed

With pin feed, absolute paper positioning within 0.01 inch is possible assuming dimensionally stable paper. Unlike plotters, which re-draw over the same surface during a short time interval, it can be expected that records written with the printer/writer may be read from minutes to months later. Humidity and temperature changes over longer time periods can change the dimensions of the paper to a sufficient degree that even pin feed may not assure 0.01 inch absolute positioning.

7.1.2 Optical feedback tracking

It should be possible to place a horizontal line segment at each end of every printed line. If a stepping motor was used for paper feed, controlled by the sensors on the head, then absolute vertical alignment would be assured. Horizontal tracking can be obtained a number of ways. If there was a ninth dot, then it could be used in a number of ways. It could be used as a parity bit. Another interesting possiblity is that it could be alternated on and off, providing an optical clock track that would assure accurate tracking horizontally.

Paper skew is probably eliminated to a sufficient degree by pin feed. If separate right and left paper motions were allowed (or there were a controllable clutch between the left and right sides of the feed) then the horizontal alignment marks could be used, along with a sweep of the print head, to adjust the paper for any small skew.

7.2 Accuracy of tracking, between different units.

Given a feedback mechanism for vertical and horizontal tracking, then

information should be transferrable between different units. The only mechanical restraint is that the heads produce the same size dots, within rather broad limits, and that the vertical spacing between dots be constant, to narrow limits. These last requirements are easy to meet.

Any non-feedback scheme would probably make the printing and transfer of software difficult, or require a much lower bandwidth. The lower bandwidth can be obtained by printing codes as bars across the entire height of the character, or by some number of vertical bars (each composed of two, three, or four dots, for example) in the height of a character.@

7.3 Error rates

Error rates depend on noise introduced at five points.

A. Flaws in the paper

B. Writing errors

C. Misalignment of the paper

D. Misreading of a correctly placed dot

E. Errors made by the electronics

I have not made any tests or done any research on the quality control of thermal paper. I have examined some samples, and found occasional dark marks that might be mistaken for a signal. With a read-after-write scheme, and software that marks errors and re-writes the correct information, it seems likely that this source of errors can, for the most part, be overcome.

Writing errors can also be caught by the read-after-write scheme, and corrected similarly. A protracted period of writing errors might cause the system to increase the writing current, if this is under software control.

Misalignment would probably cause gross errors in reading the data. Either the parity (if parity was used) or an abberant clock track (if a clock track scheme was used) would alert the system, and a message requesting that the paper be reinserted issued.

A dirty or failing light source, or an occluded or defective photo sensor might cause reading errors. Checksums, and other data checking schemes are the only protection against this kind of error.

Errors caused by electronic malfunction can be caught in the same manner.

I can give no estimates of the probable error rates with and without each of the various safeguards mentioned. I suspect that empirical studies are the best way of determining actual error rates.

7.4 Patentability

It is unclear how broad a patent might be obtained on this concept, if it were decided that Apple Computer Inc. had an interest in pursuing it. I have been

told that Sphere computers was thinking of a device with a sensor on the printer head, probably for facsimile use (but, apparently, not as a data storage medium). The combination of printer, data storage, read-after-write checking on a printer, and facsimile seems to be unique. Details of execution of the device, of course, may be patentable.

8. WRITE PROTECTION

It is important to have write protection. In the case of a themal, read-before-write design, it would be sufficient to disable printing if the paper already had marks upon it. In the case of ED designs, it might be possible to do a sense-before-burn.

9. COMMENTS SOLICITED

Comments and suggestions are welcome, especially on the marketing desireability and the technical feasability of this project.

THE MACINTOSH PROJECT

DOCUMENT 16 VERSION 0

AUTHOR: JEF RASKIN

DATE: 20 Oct 79

TITLE: An Introduction to the Apple language for Calculator Users

1. APPLE IS JUST LIKE A CALCULATOR (but, oh, what a calculator).

Picture a small calculator. There is one number that you can see displayed at the top at the top of the calculator: we will call that number the "result". On a calculator you press the equal sign or a button marked "ENTER" to let it know that you are finished typing for the moment and want to see a result. On the computer, you press the button marked "RETURN".

Since this is a written document and not an animated presentation, we will need some way to show when you press the return button. So, whenever you must press it, we will have the symbol

[RETURN]

When the computer presents an result, we will just write the result on a line by itself. For example, if the answer was 98.6, we would show it as

98.6

You can tell that the computer produced this number since it is not followed by a [RETURN]. We are now ready to begin.

Unless the computer is in the process of displaying a result, it is waiting for you to type your request. Like a calculator, it is always ready, unless it isn't plugged in.

For reassurance, the computer displays an exclamation point (!) when it expects you to type something. A dialog between you and the computer will be presented in this document in the style of this example:

2+3 [RETURN]

5

In this example you typed the first line, and the computer responded by typing the single number 5.

The simplest example is where you type a number, and the computer responds with the same number. This example shows that decimal points are permissible.

56.21 [RETURN]

56.21

If you want to add 2 to this, you can type

+ 2 [RETURN]

and the computer will respond

58.21

You can do subtraction

- .21 [RETURN]

58

and multiplication

* 20 [RETURN]

1160

and division

/40 [RETURN]

29

Let's add one

+1 [RETURN]

30

and then, since this is a scientific style calculator, take the sine of this quantity

SIN [RETURN]

.5

Or, you could do this all at once, instead of a step at a time

56.21 + 2 -.21 * 20 / 40 + 1 SIN [RETURN]

.5

2. A COMPARISON WITH SOME OTHER COMMON METHODS OF WRITING EXPRESSIONS

An essay for experts.

In ordinary computer languages, a combination of operations and numbers (or names that stand for numbers) is called an expression. The same is true

here. The difference is that the expression above would be written

SIN (((56.21 + 2 -.21) * 20 / 40) + 1)

in most computer languages. The reason it is different here is that this language is much simpler, and works like a calculator. You do not have to remember any rules but one: START AT THE LEFT, AND WORK TO THE RIGHT. You can use parentheses if you wish, but there are no "hierarchies of operators" to remember. To write an expression to solve a problem, just ask yourself what is to be done from first to last, and then type it in--in that order.

In RPN (Reverse Polish Notation), used in some calculators, the expression is

56.21 E 2 + .21 - 20 * 40 / 1 + SIN

where E is the "enter" button.

In APL (Iverson's "A Programming Language") it is (since this font doesn't have the proper division sign, forgive us for using $ for division in this case)

lo(((56.21 + 2 - .21)X 20 $ 40) + 1)

lo is how APL indicates the SIN function.

The usual hierarchical schemes begin to collapse when the number of operators gets great. For example, in Nicklaus Wirth's Pascal language, the expression

3 + 4 > 8

is ill-formed, since the logical operators have higher precedence and you can't add 3 to FALSE. But in most BASICs (Beginner's All-purpose Symbolic Instruction Code, by Kemeny and Kurtz) that same expression is legal, and evaluates to FALSE since arithmetic operators have higher precedence. The only precedent (using the word in its legal sense) for hierarchy in operators comes from mathematics where there is only the precedence of multipication and division over addition and subtraction. It arose there only as a shorthand. As RPN, APL and this Apple language have found, the elimination of hierarchies of precedence contributes greatly to readability, simplicity memorability and consistency of a language. It also makes expressions shorter, and easier to type.

3. Document 14 contains the full details of the rest of the Apple language. This document demonstrates how easily the Apple language can be introduced to users of calculators, and gives a comparison of various methods of expressions.

An article by Backus in a recent Communications of the ACM (Backus of BNF fame, now employed by IBM in San Jose) points out that expressions are easier to comprehend than programs. He suggests that the languages of the future will have their structure embedded in expressions, and points to APL as being a step in that direction. Apple does not extend the power of expressions beyond the scope of APL's uses of expressions, and does not forward Backus' ideas. It just makes these ideas more accessible.

PROJECT MACINTOSH

DOCUMENT 17 VERSION 0

TITLE: REPORT ON THE HP41C AND SHARP 5100 CALCULATORS

AUTHOR: Jef Raskin

DATE: 27 Oct 79

1. HP 41 C

The HP 41C is a general purpose, hand held, primary battery operated scientific calculator. It sells for $295, or less. It is slightly smaller than their original HP35, and the styling is still the classic wedge.

1.1 DISPLAY

The display is a 24 character LCD 14 segment alphameric display. Generally it is very easy to read, however some lower case characters, such as "e" are unrecognizable until you have a bit of experience interpreting them. This suggests that a segment type display is probably inadvisable for a general purpose computer system where upper and lower case letters must both be displayed.

The visual clarity is excellent, with the letters being a dense black on a silver-white background.

Information can be scrolled right and left.

1.2 KEYBOARD

The keyboard is typical HP, with excellent feel. It is simpler than some of the other keyboards of equally complex products, since many of the alternative uses of the keys are under software control. Pressing a key and holding it reveals on the display, in letters, the functions it will perform.

The complete uppercase alphabet appears on the keyboard, along with some punctuation. I found it difficult, without practice, to form words. This is especially true due to the irregular spacing and size of the keys, which have been optimized for calculation.

The fact that the keys have many functions, depending on your programming, and what modules have been inserted, makes this calculator the least mnemonic of any of the HP series. On the other hand, its power is considerably greater. Legends appear on the top and front of the keys, as well as on the keyboard panel.

1.3 ALPHAMERICS

The HP 41C is clearly a digital computer in almost any sense of the word. You can give variables, subroutines and statements alphanumeric labels (which is more than you can do in BASIC!). You can compare strings for lexicographic

order, and have messages printed in English. You can request alphameric input. With a bit more memory and a full-size keyboard, the HP 41C, with its compact and powerful language, would be a practical personal system.

1.4 PERIPHERALS

If it didn't look like a computer system already, the addition of a printer which can support graphics and a user-definable character set, a card reader/writer (backwards compatible to the HP 67/97), program libraries on ROM, additional RAM, and a bar-code reading wand would be a convincing argument for its "computerness".

1.4.1 PRINTER

The 2 1/4 inch, 24 character, thermal printer, which seems large compared to the built in HP printers, e.g. in the HP 97, has variable intensity control, and a number of modes. The modes allow tracing of a program, printing only on demand, or printing at key moments in a calculation. Placing the mode switch on the printer, as well as the PRINT X function, frees up a number of keys on the calculator. The printer, unlike the calculator, operates from rechargable NICADs , or from the AC line.

The blue colored printing is quite clear, definitely superior to previous HP calculator thermal printers. The calculator uses a 5 X 7 dot matrix to form its characters, with lower case descenders. It is very easy to read the entire character set. The edges of each dot are relatively sharp, but this may be due to the very fine surfaced paper that HP provides. It may be worthwhile, as an experiment, to try a sample of this paper in our printer.

1.5 LANGUAGE

I continue to be impressed by the consistent and logical way HP has extended the simple idea of RPN to make it into a programming language. As shown in the "Rosetta Stone" portion of the postscript to document M16, it compares in efficiency for problems involving scalars with the best current computer languages. It is, at present, weak in handling any data structures.

The technique of "sneaking" in programming by presenting the language as a calculator has been very effective, and many people learn to program a calculator quite painlessly due to this subterfuge. It is powerful from both a marketing and a pedagogical standpoint.

1.6 SOFTWARE

At release, the HP 41C came with a ROM "Math-Pac", and a book with many application programs from a wide variety of fields, including the inevitable game of Hangman, this time complete with real words appearing on the screen. It would not be difficult to draw the scaffold on the printer. HP can be probably be relied upon to provide translations of their usual software offerings (e.g. Surveying, Navigation, Astronomy, Games, Electrical Engineering, etc.) rather quickly. As it comes, the HP 41C card reader has a built-in program which translates (as far as is possible) HP 67/97 magnetic card software to HP 41C notation and format.

1.7 HARDWARE AND PACKAGING

I did not physically examine the construction of this device since it is the personal property of Woz. Interestingly, the connectors for the ROM's and peripherals seem to be part of a flexible PC board, as are the battery connectors. I suspect that the number of connectors required inside is very small, and that relatively full advantage is taken of the flex PC board.

There is some clever user-level packaging here, and each component has a place or carrying case. Even the ROM packs and the spare slot covers have a little book with formed places for them.

1.7.1 EMI

The calculator disturbs some channels of TV reception if it is within three feet of a top-quality SONY TV. The interference sometimes has the unusual apperance of a few lines of widely spaced white dots. An AM radio can pick up its operation from a distance of up to 4 feet on some frequencies, and it hardly disturbs an FM radio at any distance beyond an inch or two. The case is plastic, but it was not opened to observe the amount, if any, of shielding.

Since HP typically uses CMOS, low EMI would be expected compared to TTL.

1.8 MANUALS

There are quite a number of manuals, all printed on heavy coated stock, with at least three colors on each page. The instructions are complete and very accurate, and the wire-o binding is convenient. The manuals are HP's usual 5 1/2 by 8 1/2 (as are Apple's--which copy HP in this respect). There are no errata sheets or other loose pages, except for the handy summary cards, carefully plasticized.

While these manuals continue to be complete and of very high quality in layout and appearance, the writing style has lost some vigor compared with the earlier manuals. In some places, the choice of words shows a lack of care, which has not been evident in earlier HP calculator manuals.

1.9 SUMMARY

In many ways, this is the first handheld computer--the distinguishing feature being alphamerics. It is well thought out, with the main annoyance being the small number of characters on the screen, and the need to type long subroutine names (sometimes as many as six characters) to summon certain functions--although those functions you use frequently can be reassigned to whatever key you wish. Labels are provided for relabelling the keyboard.

There is some real genius in this machine, and it probably competes for one tiny corner of Macintoshes intended market. I am still more concerned about any improved versions, and HP's competitors.

2.0 THE SHARP EL5100 CALCULATOR

For $99, the user gets a 7 by 2 3/4 by 1/2 inch scientific calculator. It is distinguished by a large number of keys (60) with many unguessable function

symbols, and a large display window (4 3/8 by 1/2 inches). It comes with a hard, plastic carrying case--probably a necessity since its form factor and construction make it a likely victim of back-pocket destruction, which they duly warn against.

It looks tawdry. Especially since there is a special (kludged) RESET button on the back, the use of which is not clearly spelled out in the manual. I suspect that the software can get wedged, and that in some circumstances this button is the only way out.

2.1 THE DISPLAY

This calculator was purchased primarly to examine and gain experience with the feel of a relatively large, high density LCD display. It is an extraordinarly easy-to-read 24 character window, with each character position being a 5 by 7 dot matrix. Each dot is a precise square. The space between dots is about 1/3 the width of each square. The height of each character is 3/16 (.019) inch, for a dot size of .02", and a dot-with-space size of about .027". The characters give the impression of being very large, since they correspond to about 14 point type. Books are typically 8 or 10 point.

Clever use of the dot matrix allows Sharp to present a wide variety of characters, including a few (carefully selected) in bold face. The display scrolls right and left, and allows insertion and deletion of charaters.

After those who wish to evaluate this unit have done so, I would like to disassemble it to examine how the display is electically connected to the remainder of the circuitry.

2.2 KEYBOARD

The keyboard is a hodge-podge with a four-function calculator in the center, with a scientific function panel off to the left, and the first ten letters of the alphabet, memory and editing functions off to the right. Many of the keys have strange labels, such as "Exp" where the "E" is bold face, the "x" in italics and the "p" in simple lower case. It allows you to enter the exponent of 10 in scientific notation. Other such as PB, CD, F<>E, and TAB leave you guessing as to their use. TAB, by the way, specifies the number of digits to which numbers will be rounded!

The keys are brown and silver, with one red and one yellow key. Thelegends are white on brown, and either black or blue on silver.

The keys have a definite "fall through" although it is not as sharp or definite as that on the HP's. Nonetheless, it is better than the majority of calculators. The keys' contacts make direct contact with the single, phenolic based, two-sided PC board, as does the mode switch. The keyboard labels are typographically clear, and the grouping does make each key easy to find when you need it.

2.3 ALPHAMERICS

All symbols are clear and distinct, even if inscrutable as to function at times. Only the letters "A" through "J" can be typed, and no confusion with

other symbols is possible. The limited portion of the alphabet prevents this device from being used as a note pad. The letters are much easier to type than on the HP, being on the tops of the keys--this shows a problem with multiple key designators.

2.4 PERIPHERALS

The EL 5100 uses primary batteries, and has no provisions for any peripherals.

1.5 LANGUAGE

Here is were this calculator falls apart. There are 10 levels of priority, plus parentheses, and some auxilary rules. For example you have to remember that the fifth priority is "Multiplication cleared of "X" instruction located just before memory or pi." which is three priorities higher than ordinary multiplication, but lower than a single term function when preceded by numerals, but higher than a single term function followed by numerals.

While this calculator is far less powerful than the HP 41C, it is far more difficult to understand and use. Or, at least, it seems that way. The totally inadequate manual aids and abets the crime.

Using it as a scientific calculator with "normal" mathematical style expression input is not difficult, the trouble is that there are many obscure gotchas.

Another example of the kind of rules you have to memorize is the one that says "Provided that functions shown in item (5) (6) [fifth and sixth priority] above are successively designated in an algebraic formula, calculations are performed from the right to the left. The other functions are calculated from the left to the right."

With all this complexity, the machine is not programmable.

1.6 SOFTWARE

Without programmability, there is no software. The functions on the calculator, besides the usual scientific functions, include combinations, factorial, cube root, and the hyperbolic functions; besides the usual statistical functions is a correlation coefficient; angles are in degrees, radians and (as everybody seems to do) grads.

1.7 HARDWARE

The construction is conventional, and extremely cost conscious, using lands on the PC board as switch contacts. The case is metal, resulting in no TVI, and interference to a sensitive AM receiver can be detected only within inches of the front of the calculator.

1.8 MANUAL

The manual is dreadful. This may be partly due to translation--we should have some mechanism to make sure that Apple does not look this stupid in other languages--and partly due to a design that makes documentation difficult. The

booklet has the same form-factor as the calculator and is 100 pages in length. It is poorly organized and has no index. It is typeset and printed in brown ink.

This is not the worst manual I have ever seen, but it renders many features of the calculator opaque, even to an experienced calculator and computer user. It points out to me the importance of clarity.

1.9 SUMMARY

Except for the display, and its relatively sophisticated editing (delete and insert) for a calculator, there is little instructive about it.

THE MACINTOSH PROJECT

DOCUMENT 18 VERSION 0

TITLE: ON THE PROBLEM OF DELIMITING STRINGS IN PROGRAMS

AUTHOR: JEF RASKIN

DATE: 3 Nov 79

1. INTRODUCTION

The problem is how to delimit a string in a program. A string, by definition, can include any displayable symbols, which usually include the string delimiters. If a string is delimited by quotes for example, then it is obvious that a quote that occurs as part of the string may be mistaken for a close quote.

2. THE SNOBOL SOLUTION

SNOBOL uses two kinds of quotes, single and double, on the theory that any string containing one kind would be delimited by the other. To include a string such as

"Don't shoot," he said, "don't! I'm unarmed."

in a program you have to represent it as the concatenation of the following substrings

'"Don'

"'t shoot,"

'" he said, "don'

"'t! I'm unarmed."

'"'

Not a pretty sight. The problem is not quite solved by adding a third kind of quote since one can then contrive an example (say, where people are discussing his new kind of quote) that is as messy as the example above. Further quote symbols merely exhaust the character set without solving the fundamental problem. But see section 5 below.

3. THE PASCAL METHOD

The solution chosen by the designers of Pascal (and some others) is to have but one kind of quote and to use it twice in succession to represent a quote within the string. The messy example above becomes

'"Don''t shoot," he said, "don''t! I''m unarmed."'

One could do the same designating the double quote (") rather than the single quote as the standard quote symbol (Pascal made the incorrect choice, since the usual character sets use the same symbol for both the single quote and the apostrophe--which occurs far more often in strings than does the double quote. Thus Pascal has to use the awkward repeated symbol more often than it should. The most annoying aspect of this notation occurs when you are trying to produce nicely formatted output, in which case the number of characters is distorted, so that line lengths do not appear in the program as they will upon output.

To represent, with this notation, n quotes in a row you use n+1 quotes.

4. FORTRAN'S HOLLERITH DEVICE

FORTRAN manages to both avoid the need for quotes and keeps the length of strings constant between program and output. It represents the test string above as

45H"Don't shoot," he said, "don't! I'm unarmed."

Where the 45 before the "H" means that the 45 characters following it are a literal string. This works without fail, but requires that you count every character--and there is no room for error. It is flawless from every point of view except that of the programmer.

The "H" stands in memory of Herman Hollerith who was instrumental in the development of punched card accounting machines.

5. COMPOUND QUOTE SYMBOLS

One can create a large hierarchy of quotes by using a symbol that includes a number--in this case the compound symbol is '34'.

'34'"Don't shoot," he said, "don't! I'm unarmed."'34'

In case a string happens to contain a particular compound quote symbol, or even a whole bunch of them, there is another quote symbol you can use. It is unlikely that a string could be so long that it could contain all possible quote symbols.

6. USING NON-PRINTING CHARACTERS

A string could be delimited by non-printing characters. This works in all regards except that it makes it impossible to tell, by looking at the program, if the delimiters are present or not. You might define a symbol that can only appear in programs, and not in output. This is weak, for example, you can't write a program that writes programs (or even have examples to instruct a user in the use of strings). Both of these schemes have to modified by one of the methods already discussed so that quoted strings can be quoted.

7. USING PAIRS OF ASYMMETRICAL SYMBOLS

The problems with quotes do not occur with parentheses in expressions for two

reasons: paretheses have syntactic function so that well formed expressions cannot have random or unexpected parentheses; secondly, parentheses come in pairs. It is possible to use symmetrical pairs of quotes. The ASCII character set has both " ` " as well as " ' ". This allows nesting of quoted strings, but since these symbols can occur without structure inside strings, an additional mechanism is required.

If it could be assured that the quote delimiter symbols would occur only in matched pairs inside strings, then there would be little problem.

8. SUMMARY AND CONCLUSION

SNOBOL's solution is unsatisfactory in its lack of depth and potential operational complexity. The Pascal method seems too ad hoc, and the FORTRAN Hollerith notation is unattractive in light of the need for an easy-to-use language. Non-printing characters are unacceptable, and the use of asymmetry does not gain us any advantage. A dual solution presents itself: to use simple quotes (") when no problem is engendered by their use. When a more complex situation occurs, use the compound quote symbol shown above.

Formally, the syntax for strings is as follows, where <character> is any displayable symbol: (here, the braces are used to indicated zero or more occurrences of a syntactic element)

<string> := "{<character>}" | '<integer 1>'{<character>}'<integer 2>'

The syntax requires that <integer 1> and <integer 2> be identical, and that

'<integer 1>'

not occur in the characters comprising the content of the string.

Such a syntax can be "fooled" by an occurrence of

'<integer 2>'

in the string, so that it is the programmer's responsibility to choose the appropriate integer. A compound quote can be parsed easily:

A. When a single quote is found, not within a normal quoted string, store the integer following it. This integer, called X, is delimited by another quote. If not, it is a syntax error.

B. Every character thereafter is stored as part of the string unless it is a single quote followed by a non-integer, or an integer other than X in which case continue from B.1 otherwise continue from step B.2

B.1 The quote is part of the string. Continue from step B.

B.2 In this case the quote is followed by the integer X delimited by another single quote so that the first single quote is not part of the string, and the string is complete. Parsing continues from the first character after the second single quote.

THE MACINTOSH PROJECT

DOCUMENT 19A VERSION 2

TITLE: THE MACINTOSH EDITOR, PART A

AUTHOR: JEF RASKIN

DATE: 03 Nov 79

1. INTRODUCTION

In a personal computer system meant for the general public the design of the text editor must be especially approachable and easy and quick to learn and use. To this end, an editing system has evolved that is somewhat different from the main stream of word processor design.

2. ORGANIZATION OF THE SYSTEM

When the computer is first turned on, it is executing some program. As supplied, it powers up in the editor. There are a few editor options that make it "smart" with respect to the task that it is to do--for example in editing a program--but its capabilities remain basically the same. The options might include some of: the "Apple" calculator language, BASIC, Pascal, the Personal Communicator and the Personal Assistant. You can make any of these or any application program the default upon system power up.

The editor abilities are common to the entire system (except during execution of application programs).

There are two special keys on the keyboard. On Sara they are Apple 1 and Apple 2. For now we will call them HELP and CHANGE.

3. THE EDITOR, A FIRST LOOK

The design is based on the Bannister & Crun text editor. It is aimed at the personal creation and editing of documents rather than the typing and editing of documents created by others. Most commercial word processors are optimized for the latter task.

3.1 INITIAL APPEARANCE TO THE USER

The system is very easy to learn and to teach since it merely simulates a rather smart typewriter. It requires no commands to be used as a simple editor. You type, and what you type appears on the screen. Shift Space is the destructive backspace, and the keyboard has auto-repeat. With no further information than this, you can create documents more easily than with a conventional typewriter.

3.2 INITIAL STORING OF A DOCUMENT

What is kept on the mass storage device is not a copy of the document as you

see it on the screen, but a keystroke trail of the document. At the cost of (usually) very little extra space, this is a far more powerful item to store. It not only represents the current state of a document, but every previous state as well. You will have the option (as we shall see later) of storing the final result of following the trail, as well as the formatted version of the file. (Note that a document that is built out of many copies of a set of items is stored more compactly in trail form than in literal form.)

Another useful fact about the keystroke trail is that it never need be re-written but only appended to. This is be done automatically as soon as a given buffer fills. What you see on the screen, of course, is not the trail, but the current state of the document.

As far as the you are concerned, no thought at all need be given to the storing of a document. Given the "soft off" feature, and the automatic dumping of the keystroke buffer, the current document is always placed into non-volatile memory. Only a power failure (or other act of God, such as the dog chewing up a disk) can cause loss of data--and a power failure will lose at most one buffer (probably about ~~4K~~ 1K bytes).

You will have to give some thought to filing things away when a new document is to be started. Pressing the HELP button obtains a menu (at the bottom of the screen, called the "menu area", which is scrolled up to accomodate it). One item on the menu during editing is to file a document. The system asks for a name, and a description (which is optional). When given the name (and the description) the keystroke trail is squirreled away. Later we will discuss retrieving files, and keeping versions straight.

3.3 INITIAL PRINTING OF A DOCUMENT

There is a default formatter in the system. Another item on the HELP menu during edits is to print a document. With no further instructions, standard margins, pagination and font selection will be operative. The document presently being edited (what UCSD Pascal would call being "in the Workfile") is the document that will be printed. Later we will see that it is possible to format and/or print other documents.

3.4 SUMMARY OF FIRST USER INTERACTION

With but a few minutes of instruction from the computer (or, at worst, a few minutes reading the manual), you (or even a naive user) will be able to edit, store and print a document satisfactory for many purposes. To get further options you will have to press HELP rather than RETURN after selecting an item such as PRINT from the edit HELP menu.

In general, the menus will be very simple, and at each stage selecting an item and pressing RETURN creates an action. Selecting an item and pressing HELP gets a sub-selection or further details on the item in the original menu if no sub-selection is possible.

Note that all Apple-supplied programs for this computer will have to adhere to this style of user interaction.

4. THE CHANGE COMMAND

During editing, you can press the CHANGE key. The bottom line immediately shows this format--the cursor position is indicated by []

CHANGE [] INTO

The cursor in the text stops flashing, and the cursor in the bottom line flashes.

4.1 USING THE CHANGE COMMAND WITH LITERALS

You type the word or phrase you wish changed (called the "pattern"), and then press the CHANGE key again. The screen shows (for example)

CHANGE frugile INTO []

Now you can type what you want the string changed into (called the "replacement"). The cursor in the text moves to the point at which it found the example for which you were looking, and highlights it.

[The CHANGE instruction is operative, by using SHIFT CHANGE, even during a change.]

The search proceeds from the current text cursor position and proceeds backwards, wrapping around if necessary to complete the search. You may think it strange that the search is backwards, however in creating a document, this is the direction most often needed.

If you want to change the instance of the pattern (continue searching), you use the HELP key which has this item on the menu. Usually you will include enough context to avoid needing this feature (which should not be introduced at first).

Once you have found the desired instance of the pattern you can complete the command, which becomes

CHANGE frugile INTO fragile

and is terminated by a (third) press of the CHANGE key.

(Were you expecting, maybe, "frugal?)

When the instance of the pattern in the text is found (it was, you remember, highlighted) it is replaced by the replacement string. One HELP option is to do it again, another is to do it for every instance. Again, note that you need not use any HELP options in order to do most editing, nor do you need them to use the simpler method.

If the INTO field is empty (it is terminated by a press of the CHANGE key) then this command is a deletion. This is most natural, and avoids having a separate instruction or button for deletion.

4.2 MOVING THE CURSOR

When an instance of a pattern is replaced, the cursor is left just after the replaced section. Typing will appear at the cursor location--it is an implicit insert. The instruction

CHANGE run INTO run

serves to move the cursor to the most recent instance of the string "run". (It might be into the middle of the word "brunch" for example. If you wanted to find the word "run" you might type " run " for the pattern.)

The empty string is understood to occur at the end of the text, so that entering no string for the pattern (i.e. pressing CHANGE twice in a row) places whatever would be put as the replacement at the end of the text. It thus becomes quickly learned that three presses of the CHANGE key moves the cursor to the end of the text. Two presses does the same thing, except that the text you want to add appears as the INTO portion of the bottom of the screen.

The menu area of the screen can grow as needed until it is the entire screen or more if necessary.

4.2.1 THE EFFICIENCY OF THIS METHOD OF MOVING THE CURSOR

Interestingly, experience with an editor of this design has shown that you can unabiguously specify most instances of patterns in natural language text with three characters. With the wild cards in 4.4 below, this means that moving the cursor is usually faster using this method than with cursor motion commands.

In most programming languages, the text is more repetitious, and you will often need to specify three or four tokens to move the cursor. A compact language such as that suggested for the calculator language is nearly ideal for this editor. (See document M14)

The four normal cursor control buttons or some equivalent device is also expected to be available.

4.3 USING RANGES IN THE CHANGE COMMAND

A mechanism for specifying and moving large portions of the text is provided. The syntax is much like that of Pascal. (This concept has appeared in the text editor described recently in CACM [find reference], however it was first used by the author some ten years ago, albeit with three dots instead of two.)

4.3.1 USING THE CHANGE COMMAND WITH RANGES

Changing a large body of text can be tedious unless a range specifier is allowed.
CHANGE you m..ker INTO expletive deleted

Will take the most recent occurrence of "ker" and then find the first

occurrence of "you m" prior to that.' The periods that appear in the pattern are obtained via the HELP key, and are not ordinary periods.

If the INTO field is empty, this mechanism allows deletion of large portions of text. Since the text is highlighted, and since commands can be aborted by means of the HELP key, errors can be avoided.

4.3.2 USING A RANGE AS THE REPLACEMENT

When a range is specified as the replacement, the text specified by the replacement range is moved into the place specified by the pattern. This allows blocks of text to be easily moved. A HELP option for the pattern allows the movement to be to the current cursor position.

There are a number of techniques that users quickly develop that make this command structure more useful than it might appear to people unused to it.

THE MACINTOSH PROJECT

DOCUMENT 19B VERSION 0

TITLE: THE MACINTOSH EDITOR, PART B

AUTHOR: JEF RASKIN

DATE: 03 Nov 79

4.4 USING THE WILD CARDS

4.5 SCANNING THROUGH A DOCUMENT (and grammatical positioning)(to beginning and end)

4.6 UNDERLINING AND OTHER MODIFICATIONS OF CHARACTERS

4.7 OUTPUT FORMATTING (& widows and orphans)

4.8 ABILITY TO UNDO MISTAKES(and the lack of need to protect areas of text)

4.9 THE MENU AREA (and enviornment setting: headings, footings, margins, page numbers, footnotes, paragraphs)

4.10 HANDLING USER ERRORS

4.11 TABS AND COLUMNS

4.12 MACROS

4.13 DIFFERENT PRINTERS

5.0 PROPERTIES OF THE EDITOR

This is certainly one of the "what you see is what you get" variety editors. It has roughly the same power as the UCSD and Apple Word editors. Each of these editors has some "features" the other lacks, but they can all do about the same job. Apple Word is easier than our other editors to use, and I suspect the present editor may be easier still.

5.1 SIMULATING THE B&C EDITOR ON AN APPLE II

This system can be simulated with CTRL C and CTRL D for the CHANGE and HELP keys. On most sytems CTRL H is the destructive backspace, if SHIFT SPACE is not available.

5.2 MANUALS, BOTH ON- AND OFF- LINE

6. IMPLEMENTATION

6.1 PERSONNEL

6.2 IMPLEMENTATION METHODS

6.3 SCHEDULE

7. FUTURE EMBELLISHMENTS

8. ENGELBART STRUCTURE OF A TEXT

THE MACINTOSH PROJECT

DOCUMENT 20 VERSION 0

TITLE: THE MACINTOSH DISPLAY

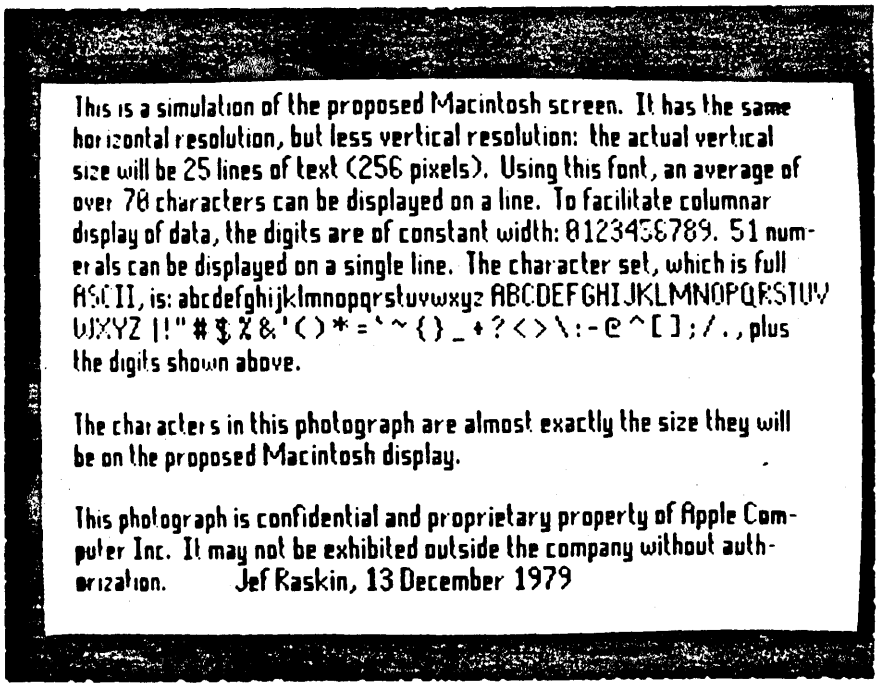
AUTHOR: JEF RASKIN

DATE: 12 Dec 1979

ABSTRACT
Many display technologies have been studied. Regretably, all but the CRT are as yet either not ready or too expensive to consider. Human factors studies suggest that a screen should have from 25 to 30 dots per centimeter to make the displayed characters appear continuous. More is uneconomical. Fewer make the dots visible to the average reader at normal reading distances. Programming considerations show that a 256 by 256 display is desirable in terms of program speed and efficiency. These factors fit nicely with a small CRT with a viewing area of from 8.5 by 8.5 cm. to 10 by 10 cm.

The photographs on the next page show a simulation of the proposed screen at actual size. In printer's terms, the characters are in an 8 point font, which is larger than the print you are reading now.

Portability and cost constraints, combined with the expected application areas of Macintosh, preclude color.



This is a simulation of the proposed Macintosh screen. It has the same horizontal resolution, but less vertical resolution: the actual vertical size will be 25 lines of text (256 pixels). Using this font, an average of over 78 characters can be displayed on a line. To facilitate columnar display of data, the digits are of constant width: 0123456789. 51 numerals can be displayed on a single line. The character set, which is full ASCII, is: abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ !"#$%&'()*=`~{}_+?<>\:-@^[];/.,plus the digits shown above.

The characters in this photograph are almost exactly the size they will be on the proposed Macintosh display.
This is a simulation of the 256-dot wide Macintosh screen. Since it is being simulated on an Apple II, the full 24 lines cannot be displayed. On this screen, using the font at which you are now looking, an average of over seventy characters per line can be displayed in normal English text applications. The numerical font is constant width: 111122223333444
12345678901234567890123456789012345678901234567890
As demonstrated here, 51 numbers can be displayed across the full width of the screen. abcdefghijklmnopqrstuvwxyz,./;[]^@-:\
!"#$%&'()*=`~{}+?><ABCDEFGHIJKLMNOPQRSTUVWXYZ_ This comprises the complete ASCII character set. Here is some Pascal code:

due to the CRT aspect ratio, this paste-up shows only 23 lines rather than the 24 lines (at 11 lines/char) or 25 lines (at 10 lines/char) that should appear

```
function GCD(firstnumber, secondnumber: integer): integer;
var remainder: integer;

begin
   if firstnumber < secondnumber
      then
         GCD := GCD(secondnumber, firstnumber)
      else
         begin
            remainder := firstnumber MOD secondnumber;
            if remainder = 0
               then
                  GCD := secondnumber
               else
                  GCD := GCD(secondnumber, remainder)
         end
end;
```

how a program might look

1. INTRODUCTION

The various questions about the nature of Macintosh's display have been largely resolved. This document summarizes the research done to date.

1.1 BUILT-IN DISPLAY

In Document M2, a number of areas of concern about the display were brought forward, notably absent was any mention of predictability. One of the main weaknesses of the Apple II was the display limitations imposed by the variability of external displays that might be attached. We were forced to design for the lowest common denominator: a poor color TV. In order that Macintosh be self-contained and to give us control over the quality and specifications of the display, it will be mechanically and electrically part of the main package.

1.2 BIT-MAPPED ARCHITECTURE

To permit both graphics and text, a bit-mapped display has been chosen. Experience has shown it to be both a cost-effective technique with few limitations in applications. In fact, the bit-mapped display on the Apple II is certainly one of the reasons it has remained so popular.

1.3 MARKETS SERVED

The display resolution proposed below will satisfy the needs of most major marketing areas such as personal use in business, science, industry, and the hobbyist. A small problem with some potential educational applications is discussed below.

2.0 DISPLAY TECHNOLOGY

A number of technologies have been explored. It is clear that Apple must follow and perhaps even develop one or more of these (or even other possible) display technologies.

2.1 LIQUID CRYSTAL

Two avenues have been explored, direct LCD viewing and a small LCD "slide" using transmitted light onto a screen. The problem of multiplexing an LCD matrix of the required resolution at any size has not been solved at the present time. When and if this technology matures, its small size, light weight and low power consumption will inspire many products--including a redesigned personal computer.

2.2 PLASMA AND SELF-SCAN PANELS

At present this attractive technology suffers from excessive cost. A 256 by 256 display without electronics can be purchased for $100. Paul Baker has suggested that we could produce the panels (with moderate resolution) in-house at reasonable cost, but at present this option has not been studied.

2.3 LIGHT EMITTING DIODE ARRAYS

These have just become available, and are being applied by the military. They are too expensive to consider at the present time. There are some concerns about power requirements as well.

2.4 FIELD EMISSION DISPLAYS

While a promising technology, it is unlikely to be ready in time for the present project.

2.5 ELECTROLUMINESCENT PANELS

Sharp and Hitachi, among others, have exhibited such displays. They are not yet available commercially.

2.6 LASER SCANNERS

An optomechanical scanning device was considered but abandoned on the grounds of mechanical difficulty and questionable maintainability.

2.7 CRT

CRTs are undesirable on grounds of size, weight, fragility and the high voltage necessary. Nonetheless, nothing approaching their low cost and ready availability exists. Two approaches have been considered: direct view and projection.

2.7.1 PROJECTION CRT

Our consultant, Alan Stein has been investigating this possibility. At present there are possible difficulties with brightness and contrast, and it is not clear just how available are the extreme brightness CRT's required. In any case these projection CRTs are about 7 or 8 cm in size, which is quite close to the size of the direct view CRTs being proposed. Of course a projection CRT would give us a larger screen, but in that case more resolution would be required to meet the goal of 25 to 30 dots per cm. (see section 3 below).

Use of a projection scheme is very attractive since it allows a folding display and thus a compact, lightweight package. On the other hand, it requires more mechaical assembly, optics and other parts that would increase maintainance problems. Keep in mind that a projection CRT system has all the complexity of a CRT, and in addition has optical and mechanical components.

2.7.2 DIRECT VIEW

A direct view CRT has the advantage of simplicity. Such a CRT may either be a flat screen display or a conventional (and awkward) bottle type CRT.

2.7.2.1 FLAT CRT

Allen Stein has been investigating these, such as the Sinclair rectangular tube. As with many other promising technologies, this one is, at present, merely promising. It does not seem that such a CRT will be prepared in time and at a competitive price, although it is by no means out of the question

that given adequate incentives it could be done.

2.7.2.2 CONVENTIONAL BOTTLE

There is no doubt that the conventional CRT is an excellent candidate for Macintosh's display.

2.8 HARD COPY AS A DISPLAY MEDIUM

Mike Markkula has suggested that the possiblity of having no volatile display be investigated--or, if one could be invented, a volatile hard copy display be devised. The requirements for too many applications require a fast-changing display, and no known technology (unfortunately) will permit updating a piece of paper or other hard copy medium at anything near the required rates.

3.0 HUMAN FACTORS

The main factors with which we are concerned are resolution, flicker, size, color, brightness and contrast.

3.1 RESOLUTION BOTH GRAPHICAL AND OPTICAL

We have two resolutions, the program resolution or number of dots comprising the picture, and the resolution of the screen as seen by the human eye. For a given screen size, fixing one determines the other. Experience and a number of studies have shown, as stated above, that characters composed of dots seem continuous at between 25 and 30 dots per cm, and their edges seem smooth at above 80 to 90 dots per cm. Careful inspection of a character will reveal jagged edges until a dot density of 150 to 200 dots per cm. is achieved. (these studies assume round dots just touching at their edges in a square packed tesselation).

For our purposes, the formation of characters that appear continuous, we should strive for a screen dot density of 25 to 30 dots per cm. If we choose, for the purposes of portability, a 4 or 5 inch CRT, then the desired density is obtained with between 200 and 300 dots across the tube. Programming considerations on a byte oriented machine suggests the convenient figure of 256 dots.

3.2 BLACK ON WHITE VS. WHITE ON BLACK

The work with a number of commercial word processors (e.g. CPT, experienceat Xerox PARC, and also the photographs on page 2, suggest that black on white, as used nearly universally on the printed page, is to be preferred. I suggest that this not be made a user option.

3.3 FLICKER

A light-colored background makes flicker more noticable than does a black background. User preference indicates that a white phosphor is to be preferred over the slower green. 50Hz seems to be a bit slow, so it is suggested that a higher rate be used. If, as proposed here, no attempt be made to have standard video, and we do not have to be PAL or NTSC compatible, than any refresh rate above 50Hz would be acceptable, and we can choose some

convenient sub-multiple of the system clock.

3.4 COLOR

Considerations of cost and portability practically exclude color as even an option on Macintosh. The areas in which it is intended to be used, fortunately, do not rely as heavily on color as would a home or game oriented machine.

4.0 COST

Gary Baker estimates that a CRT system that can display 256 by 256 dots might cost in the $20 to $30 range.

5.0 SIZE AND WEIGHT

The CRT will be about 6 inches in width, 5 high, and between 6 and 8 inches in length. The entire CRT and associated components will weigh between one and two pounds.

6.0 MEMORY REQUIREMENTS

A bit mapped 256 by 256 display requires exactly 8,192 bytes of memory.

7.0 PROBLEMS WITH NON-STANDARD VIDEO

While non-standard video has some advantages, such as extra vertical resolution and electronically convenient refresh and scanning rates, we lose the ability to use inexpensive slave monitors. This hurts us most in the educational arena where large monitors are especially useful. It also makes dealer display more difficult.

On the other hand, the enhanced quality of the image and lower price permitted by non-standard scanning may be a greater asset.

8.0 POWER CONSIDERATIONS

A conventional CRT and associated electronics adds a load of between 9 and 12 watts to the system.

9.0 FONTS, DOMESTIC AND FOREIGN

As shown by the photographs, a small CRT, with the appropriate fonts, can be extremely readable. The large number of lines will allow the special characters and diacritical marks demanded by non-English languages. The software engendered by this approach will be very flexible in these regards.

10.0 APPENDIX: FONT GENERATOR PROGRAM

This is a simple program that generates and allows primitive editing of a proportional font. The one shown is called "Two-bit Gothic Proportional Condensed" and was designed by the author to demonstrate how readable a truly compact font can be.

DOCUMENT 21 VERSION 0

TITLE: BEYOND WORD PROCESSING: THE ONLINE TEXT SYSTEM

AUTHOR: DAVID CASSERES

DATE: 3 October 1979

The purpose of this paper is to stir up some interest in the possibility of an Apple implementation of a unique and powerful personal computer tool, the Online Text System (OTS).

OTS is a personal system for entering, editing, and studying text. It differs radically from today's concepts of "editors" and "word processors" as explained below, and is optimized for the user who wants to use text in large quantities as a primary intellectual tool.

As explained below under "History," the essential concepts presented here are not new. They have been implemented in a research prototype system and exhaustively tested in a work environment for several years by a team of up to twenty technical, clerical, management, and documentation people.

There is reason to think that someone besides me is thinking about implementing these ideas on a personal computer.

OTS IS NOT ANOTHER WORD PROCESSOR

Most existing text processors fall into two categories:

> Editors: Direct or indirect descendants of hacks developed (often decades ago) by programmers who needed to edit source code.
>
> Word Processors: Elaborately integrated systems to be used by a clerical operator. These systems are intended for production of business documents from letters up to (occasionally) manuals. The function of formatting the text for hard-copy production is the heart of these systems. Text entry is assumed to be done from some kind of draft provided to the operator by someone else. Documents are assumed to be small. Editing is assumed to be simple, and it is assumed that if you want to study a document you get a hard copy.

The proposed OTS is more like an editor than a word processor, in that it is to be operated personally by the creative worker. Also, it does not incorporate hard-copy formatting at all -- this is done by any of a whole family of external processes with various specializations. All formatting done by OTS is screen formatting, and OTS is heavily optimized for this.

For example, while almost all editors and word processors are at least partly line-oriented, OTS is line-ignorant. It breaks statements into lines according to current display parameters, and does not store lines as such except when the user explicitly wants it to.

CONCEPTS

OTS is designed with large, complex texts in mind. The problem in dealing with such texts on a screen is the difficulty of orienting oneself to the structure and content of the text. When you try to study, you get lost very quickly. Anyone who has tried this can understand the problem. It has never been addressed in a meaningful way by any production text system.

OTS does address this problem. By letting hard-copy formatting be an external function, OTS is able to have a study function as well as the traditional capture and modification functions of an editor. The same features that optimize OTS for studying the text also optimize it for ultra-efficient editing (since study is a key part of editing).

Existing systems treat the computer as a tool for developing hard copy, under the assumption that hard copy is THE medium for text. Such systems are based on partial simulation of hard copy on the CRT. OTS treats the computer/CRT as a new medium for text in its own right.

TEXT STRUCTURE

The key idea of OTS is that a text has an implicit structure which is known to the system. The structure used is the classic tree. Nodes in the tree are called "statements," and the content of a statement can be any string. Typically, an OTS statement is used to contain a paragraph, a heading, or a source-language statement. (Anyone who finds it hard to visualize a document as a tree structure need only think of the conventional "outline form" we learned in high school.)

So every statement not only has a certain position within the sequence of statements, but it also has a "level" in the hierarchy of the tree. The system knows about levels. For example, when you create a new statement the system offers you a default level which is the level of the preceding statement. You can adjust the level down or up.

For a first glimpse of the power this affords, imagine a huge text -- equivalent to an entire corporate planning document, for example. Each chapter heading is a top-level statement; each section heading within a chapter is a second-level statement, and so forth.

If you loaded up such a text in a conventional editor or word processor, you would then be in bad trouble if you wanted to find something that might be near the middle of the text. With tree

structure, you can command OTS to display only first-level and second-level statements. The second-level statements are displayed indented. Instant outline... Now you find the section you want, and move it to the top of the screen; at the same time, you specify "opening up" another level, with only the first line of each statement on the display. In this manner, you can very quickly find what you are looking for (in most cases).

The tree structure is chosen because it is natural for both documents and computers. It is also an easy structure to understand. Writers who use it as an organizing principle soon learn to love it, and documents written this way are merely obeying the ancient high-school teaching: start by making an outline. It is not limiting, since the user is perfectly free to write all statements at the first level. This results in a "conventional" structure, i.e. linear.

Using a system of this kind to develop and study the source code of a large program written in a block-structured language is a revelation. Moreover, it turns out that if you use such a system as an editor for developing large source codes, it encourages structured programming.

USE OF TEXT STRUCTURE IN EDITING

The editing function falls into two parts: statement editing (the familiar manipulation of strings, words, and characters) and structure editing. Structure editing is the manipulation of structural entities within the tree. For example, a "branch" is a subtree -- a particular statement, all its substatements, all their substatements, etc. You can delete branches, move them, copy them, etc.

Other structure entities are defined: a "plex" is all the branches that are subtrees of a common parent; a "group" is a set of contiguous branches.

The display normally shows structural relationships by indenting statements according to their level. Also, statements implicitly have numbers like

```
1 xxx

   1a xxx

2 xxx

3 xxx

   3a xxx

      3a1 xxx

      3a2 xxx

      3a3 xxx

   3b xxx

   3c xxx

4 xxx
```

The statement numbers are generated automatically and displayed on command. Of course, statement numbers change automatically whenever structure editing is done.

USE OF TEXT STRUCTURE IN STUDYING

The study function is supported by commands such as the J(ump set: [The prompt notation of UCSD Pascal is used here as an example, it should probably not be used in an Apple implementation of these ideas--Jef]

```
J(ump
   S(uccessor: next statement at same level
   P(redecessor: preceding statement at same level
   U(p: parent statement
   D(own: first daughter statement
   O(rigin: root of tree, first statement in text
   L(ink: statement referenced in textual link
   N(ext: next statement in linear order
   E(nd: last statement in text
   R(eturn: statement jumped from to get current state
```

A J(ump command operates by placing the specified statement at the top of the screen. The syntax also allows for the entry of various one-character codes called viewspecs. The viewspecs set the display control parameters for such things as the number of levels displayed, number of lines displayed per statement, whether statement numbers, labels, links, etc. are displayed, and so on.

Another important viewspec enables or disables display filtering.

Display filtering is OTS's equivalent of the "string search" capabilities of a typical text editor. OTS offers the user a powerful "string specification language" (SSL) for specifying content characteristics of statements to be displayed; only statements with the specified type of content will be displayed when filtering is enabled.

SSL is essentially an extended BNF. It contains a subset that is easy to use and approximates the customary string search function. For the more experienced user, a specification written in SSL can be stored as statement text and used by means of the command E(xecute S(tatement.

SSL is itself a subset of a larger language which allows specification of primitive string-editing functions. This is String Search and Editing Language or SSEL. Users who are interested can also have all of SSEL, and can execute their own automatic editing sequences via E(xecute T(ext.

This accesibility of detailed primitive functions via high-powered languages, and their integration into the text via E(xecute T(ext, is an important part of the design philosophy of OTS. Another step in the same direction is the O(utput to C(ompiler command, which outputs a text of high-level source code (say Pascal) to a pre-compiler which converts it from OTS tree-structured file format to a compilable file. At this point OTS becomes reasonably serious as a software development/maintenance tool.

ANOTHER KIND OF STRUCTURE

Although the tree is about the most general and useful of the regular graph structures, it cannot do everything. Often it is desirable to have a linkage between two statements entirely independent of the tree structure. OTS allows you to establish such a linkage by giving a label to a statement and giving other statements links to that statement. The system knows about the meaning of links, and can "follow" them on command, displaying a statement according to a reference in another statement.

Labels and links are part of the content of a statement, but can be suppressed from the display on command. Notice that in high-level source code, labels and links can have the syntax of procedure names and procedure references. Also, a link can point outside of the current text.

In summary, a text has an inherent tree structure and may also have a user-imposed arbitrary structure based on links. The link structure can point out of the text and into another text. The two structures are entirely independent of one another and both are actively supported by the user system.

HUMAN INTERFACE

The success of any OTS implementation depends critically on the beauty of its human interface. The interface must feel extremely responsive. Speed of human input and speed of system response are both important.

Human input is of two kinds: command mnemonics and cursor positioning.

Cursor positioning is done by means of an analog device such as a joystick, forcestick, graphics pad, or mouse. Cursor control keys are absolutely not adequate, and it is critically important to make cursor positioning as easy as possible. [I suggest that all of these means of cursor positioning are relatively slow, see M19--Jef]

It is desirable to have two control buttons available on the cursor-positioning device: one is used for a "command accept" function, and the other for "command delete."

Most command mnemonics are single characters, and the command set is tree-structured. The flavor of OTS commands is best given by example; the D(elete commands are a good example:

```
D(elete
   B(ranch
   P(lex
   G(roup
   S(tatement
   W(ord
   V(isible
   I(nvisible
   T(ext
   C(haracter
```

These commands illustrate the general philosophy of OTS editing commands: one character (D) specifies a kind of action, and the next character specifies the kind of entity to be acted on. The entity W(ord, for example, means a string of alphanumerics bounded at both ends by non-alphanumerics (or statement boundaries). After typing DW, the user positions the cursor to any character in the word and types a "command accept" character (which might be ctrl-C). The word is deleted; and OTS is smart enough to delete a space along with it, if the space delimits it. Alternately, instead of "command accept" the user may select another word; the meaning is to delete the two selected words and all between them.

A V(isible is simply any string of non-blank characters; an I(nvisible is any string of blank characters. T(ext is an arbitrary string defined by pointing to its first and last characters.

In general, when OTS has a traditional function such as editing, it beats the competition by being smarter. For example, M(ove and C(opy are distinct editing commands, and there is a T(ranspose command as well. The entity W(ord is handled correctly in all cases, as regards

punctuation and spacing after an editing command is executed. Such features are not frills; they are essential to the "hot" interface that any system like OTS must have if it is to succeed.

NOTES TOWARD IMPLEMENTATION

The file structure of an OTS text is something to think about. Ideally, there would be random access to statements; and ideally a text could extend over many diskettes. Hard disks are a natural thing to think about here.

Speed of response is important -- particularly the time required to reformat the screen.

MARKET

Who knows? My own position has always been that the potential market for this kind of tool is far, far greater than anyone has yet guessed -- at least if it is as cheap as it now can be. Many of the capabilities have to be used to be fully appreciated, and my guess is that to sell a full-dress OTS (or even half-dressed) would require some serious educational efforts. The implication is that if an OTS is worth doing, it's worth doing right.

HISTORY

At the 1967(?) Fall Joint Computer Conference in San Francisco, Dr. D. C. Englebart of SRI presented a system called NLS (for oN-Line System). NLS was presented as a tool of very wide scope for "augmenting the human intellect," but in its specifics it was a system for editing and viewing text information. It offered the individual text-oriented worker far more power than anything before or since.

People who saw the FJCC presentation were overwhelmed -- as much by the presentation as by its content. Englebart sat on stage at a futuristic workstation custom-designed by Herman Miller, and operated the system in Menlo Park via microwave. The CRT image was projected on a theater-size screen behind him; TV images of his face and of busy workers at the Menlo Park site were intercut with the NLS display. It was a media triumph, but unfortunately one of the main impressions people got was that NLS was a monstrously expensive, extravagant, esoteric superwhizzie for use in a futuristic laboratory by the most rarefied geniuses. I think that many went away with their eyes bugging out, talked about it for a while, and then forgot.

At that time, NLS ran on a timesharing XDS-940 dedicated entirely to serving NLS users. About half a dozen users could be served efficiently, up to about 20 with serious degradation of response.

Later, SRI's Augmentation Research Center operated NLS as a tool for the ARPA Network Information Center (NIC). This incarnation of NLS ran on a PDP-10. Another incarnation was delivered to the Air Force.

The engineering that went into NLS was extremely impressive. For example, a high-level machine-oriented language called MOL940 was developed for the XDS-940. It served as the mainstream language for developing NLS; but it was also used to develop a metacompiler which created compilers for a family of Special-Purpose Languages (SPL's). The SPL's included a string search and editing language, a tree manipulation language, etc. Some of the SPL's, or subsets of them, were directly available to the NLS user via NLS itself. Eventually all the NLS software was contained in NLS structured files, cross-referenced to each other by the actual procedure calls in the source code and also by text links to the object code files. It was easily the most elegant large software system around.

Hardware engineering was equally impressive; for example, the "mouse" device was invented in connection with this project. Some of the technical people on the project were Bill English, Jeff Rulifson, Bill Paxton, Chuck Irby, Don Andrews, Dave Hopper, Bill Duvall, Mimi Church, and Bruce Parsley, many of whom later went to Xerox PARC.

At the time of its development, NLS was strictly a demonstration of concept, as far as any public beyond the ARPA community was concerned. It was very expensive to use, its human interface depended on then-exotic hardware, timesharing systems were unreliable and inefficient. The concept of computer text processing was not thought to involve anything better than justified margins on a Flexowriter. Eventually funding dried up, most of Englebart's technical people went to Xerox PARC where they were forever walled up in ivory, and NLS was forgotten -- by most people.

I was until 1969 the technical writer for the project, and have been carrying some of the ideas around ever since. I always thought it was a great pity that such a spectacular tool for writing and studying text was forgotten. It is now clear to me that future Apple systems are ideal for a modern descendant of NLS, optimized for an individual user. OTS is such a descendant, and I believe that if Apple does not eventually implement something like OTS, someone else will.

THE MACINTOSH PROJECT

DOCUMENT 22 VERSION 1

TITLE: HOW CAN WE MAKE COMPUTERS TRULY PERSONAL?

AUTHOR: JEF RASKIN

DATE: 10 FEB 80

What is currently considered "state-of-the-art" technology is being applied in many ways that affect our lives. While anything that affects our lives has an impact upon us as individuals, and is "personal" in that sense, I wish to distinguish one special set of application areas of technology from the rest.

This distinction is easily made by observing some existing technologies, each of which is over a century old: the telephone, home central heating, and the office desk. Consider the desk first. In most circumstances it is assigned to a particular individual who will be the only person to use it--sometimes for many decades. Many sociological rituals and taboos surround the desk. Only the owner and a few other people, such as the owner's secretary or boss or other immediate "family" within the company feel at ease looking through it. In some instances there may be no other person who is authorized to go through your desk. In spite of this close attachment of the desk to an individual, it is rarely considered personal property.

This example shows that an item, though closely connected with one individual, may not be personal in that it is neither owned nor controlled by the individual (the company may reassign the desk, say to move in a new one, without violating any taboos). It only affects the individual at work, and is rarely (if ever) thought of outside of one's place of employment.

The example of central heating may be taken to stand for the welter of tangible property that we own (or rent) and that is associated with our homes. There is no doubt that this is all personal property, but none of it is essential personal technology in the sense being discussed here, even though it affects our everyday lives and is often technologically based. I exclude these items because they are attached to the house and not to the person. When we move from place to place, we do not take the furnace and radiators or ductwork with us. While they may belong to us, they equally well belong to the home. Other items, such as a console television set, will move with us if we move, but they are primarily used at home. It is rare to find a 25 inch TV at the beach or in the office. Rather than personal technology, these items are appliances or fixtures. A small portable TV may be an instance of personal technology.

The telephone instrument itself is attached either to home or to the working place (or is a public phone), and is thus not essential personal technology, but the telephone system is a different matter entirely. It affects every phase of our lives, in matters both personal and involving our jobs. We

carry telephone numbers with us, in books and in our heads and feel free to use them at almost any time. The telephone system, though not owned by any of us, and not physically within our command, is an example of essential personal technology in the sense I am trying to develop.

The general characteristics of an essential personal technology include its applicability to work, homelife and play; portability or wide geographic access or both; its importance or significance to the user in all these roles; affordability; and options to make the technology aesthetically acceptable to the owner.

If a technology does not impact all phases of the user's life, that technology is not personal; if it is limited as to where and when it can be used, a technology is not personal; if the user cannot purchase the technology or its use, it is not personal; if the user does not feel comfortable using the technology it is not personal; if it does not matter to the individual if the technology is always available or nearly so, it is not the kind of essential personal technology being discussed here.

Calculators are an example of a technology that moved from non-personal to personal technology in the past decade. For many individuals it is used at work (say, to figure out sales tax), at home (to balance the checkbook) and at play (to average bowling scores). The pocket calculator is independent of power lines, is small and cheap, and comes in a wide range of styles to appeal to the consumer's illusion of making a significant choice. It became personal when microelectronics made it small and inexpensive.

The same technology spawned the "personal" computer. However, in spite of their name, they are not yet truly personal. The so-called personal computers fail the tests proposed here on a number of grounds. Because computers, much more than most other technologies, have a wide and unpredictable range of application areas, the argument is more complex, but the basic factors are the same.

The personal computer of today (such as the Apple II on which I write this) fails nearly every test that I propose. I can use it at work, and do so every day. But then, I am a system designer, programmer and writer by trade. I do use it at home, but just to continue my working day at a different location. My calculator balances checkbooks nearly as well, and a lot more conveniently. I can force the computer into my avocations, but it is a square peg in a round hole. If the computer is used, as many have proposed, to control my house (e.g. adjust the heat, the windows and shades so as to take maximum advantage of solar heat and minimize thermal losses) then it becomes a fixed appliance and cannot be moved without disrupting the house.

The present personal computer is neither truly portable nor widely available. Even though my computer is very light, rugged and small (about 5 Kilograms, and the size of a portable typewriter) and attaches to the ubiquitous television, by the time you bring along the mass of cables, the disk drives, and (heaven forbid) the printer, you need a wheeled cart or pet octopus to help you out. In addition, it must be plugged in before it can be used, so that it is difficult to use while travelling or even on a desk in the middle of a classroom--both places where the calculator is as handy as ever.

My Apple is very significant to my life. I quail at the prospect at ever having to use as clumsy a tool as a typewriter again. Yet it is not significant enough to be carried along wherever I go, nor do I feel a pressing need to have corner computer booths as we now have telephone booths. Thus its significance to me is not enough for it to classify as an essential personal technology. Only technocrats will find it as useful as I do.

There is some question as to whether personal computers are affordable. With the accessories required to make them practical for a wide range of roles the cost is certainly over $2,000. This is too much by a factor of at least two. A realistic system, including sufficient quantities of mass storage (which stores information and not mass!) and means for making this information available to the user (such as a screen or a printer) will cost much more than that. My system, to be practical just for the writing tasks I perform (such as this article) has a retail cost of about $7300. To be personal it should cost under $1000.

The system also fails the test of comfort of use and compatibility with the dictates of personal style and interior decorating. While I could build (and many have built) a special enclosure to hide it all, the many separate pieces and their interconnecting cables have a mad scientist air about them. And my computer happens to be one of the neater ones on the market--most are worse. The computer comes in only one color, a neutral off-white. Again this is better than many others, but even the telephone company offers a choice of colors and styles for their electronically identical instruments.

Computers are not comfortable to use. A parallel comes to mind: The first scientific calculator used a notation called "RPN" (Reverse Polish Notation) which while truly superior to the way most later calculators operate, does require a few minutes of thought and practice. As a result, only a small percentage of calculators use RPN because it was not immediately comfortable. Similarly, to get the full benefit from a computer, and to exploit its inherent flexibility the user must program it in some form (although many benefits are available without programming).

Programming, as a human activity, rates with torture in the popularity polls. As presently constituted programming requires study and practice. The low proportion of scholars and violinists in our population attests to the avoidance of study and practice by most people.

It may be possible to provide programmability in a convenient and painless form, but it has not yet been achieved.

If a computer is not programmed by the user, then it must be employed via programs written by someone else. Whether such programs are comfortable to use depends on the sympathy and insight of the programmer or program designer. At present, programs more commonly reflect the difficulties encountered in programming than the real needs of the user: like the spines of a sea urchin, the awkwardness of most computer systems comes through in spite of many attempts to make the programs smooth and humane.

The instructional books provided with programs (and with most so-called personal computers) all too often do not provide the cushion they might against the stiff frames created by the programmers and system designers.

Thus present personal computers fail the test of comfort as well, and present owners must put themselves out in order to use them.

The personal computer will come of age when it goes the way of the calculator or the telephone, or probably both. It will be small and portable, affordable, come in many styles, be easy to program when programmability is desirable, it will have features and software that make its use natural and convenient for an individual in the multiplicity of functions that each individual performs. In other words, it will become a nearly indespensible companion like a Swiss Army Knife becomes to certain people, and its owner will feel a bit helpless when she or he leaves it behind.

THE MACINTOSH PROJECT

DOCUMENT 23 VERSION 0

TITLE: JANUARY 1980 OVERALL SUMMARY OF THE MACINTOSH SYSTEM

AUTHOR: JEF RASKIN

DATE: 12 Jan 80

0. Concept

The purpose of this design is to create a low-cost, portable computer so useful that its owner misses it when it's not around--even if its owner isn't a computer freak. [M2, M4, M6, M8, M13]

The notations, such as [M0] refer to existing documents giving further details.

1. Hardware

Macintosh is intended to be a complete, self-contained, portable, personal computer. It does not have to be attached to anything other than a power source in order to operate. An optional battery pack will be available.

Macintosh is designed to sell for about $1000 [M5], and will have a disk drive (with consideration of the possibility of a dual disk drive) and 7 inch CRT. A lightpen for graphic input is being considered and the computer may contain a small printer [M15],. Macintosh will have a full alphanumeric keyboard, without separate numeric pad (an embedded pad will be used instead). If we have dual disks, they will share drive and head positioning motors. The size should be about 12 inches wide, 14 inches deep (maximum), and 6 1/2 inches high. The weight should be under 22 lbs.

The electronics are conventional but streamlined: a 6809E processor, eight 64K RAMs (no expansion provided), 6845 video generator, a modem/DAA, ACIA, and supporting circuitry. We have not begun to approach the limits of what can be done with such an architecture.

One central concept in the design of the hardware is that the programmers must have a fixed environment. This will help insure reliable user level programs--the emphasis in selling Macintosh will not fall on the hardware as much as the tasks that can be done with it. There must be no hidden "gotcha's" in the design (this program works only if you have a super-duper board in slot 3, and have a jumper from pin 5 of IC 56 to your left pinky) [M6]

The display is bit-mapped, black-and-white, 256 by 256 resolution with a 10 by 10 cm (4.5 inch) display area. A speaker and microphone will be built in, along with 8-bit D/A and A/D circuitry as necessary. Portability and cost constraints preclude color--in addition, the software should be compatible with future display technologies, which are likely to be monochromatic at

first.

There will be a battery-supported real time clock.

The computer may operate on 12 to 15 VDC, allowing operation from a wall transformer (minimizing thermal problems as well as simplifying UL approval), a battery pack, or from an automobile battery.

2. Software

At the user level there will be a combined text editor-data base manager, a calculator based language, and a disk oriented BASIC.

The text editor is designed to be especially easy to learn and very fast to use. It is conceptually quite different than existing text editors, and a demonstration is being prepared. Tests have shown that many simple operations require one third to one tenth the time (for the operator) when compared to current Apple-based editors. The same mechanisms that allow the user to search through and modify text will be designed to allow searching and modifying a data base. [M19, M21]

Text is displayed in a proportional font, allowing an average of 72 characters per line; there will be at least 24 lines of text.

The calculator based language is, again, extremely easy to learn. It is designed to sneak the user into programming, and yet provide powerful and immediate commands without creating programs. It is at the same "level" as the text editor and will operate without system commands. [M14, M16]

The BASIC should be an ANSI standard BASIC. It will be disk resident.

At the system level will be Pascal and a macro assembler. The operating system, invisible to the user, will be partly based on the concepts found in the Sara operating system.

3. Network

Macintosh, however nifty its hardware and software, will not sell unless it does something useful. The number of useful things a personal computer can do with a network is vastly greater (probably by two orders of magnitude) than what it can do without a network. Thus the modem/DAA is an essential part of Macintosh, and Apple must provide, at the very least, hooks into various information services. [M3, M12]

4. Manufacturing and Service

The electronics and software of Macintosh, as well as the physical packaging, are being designed to allow economical manufacturing techniques and ease of repair. The electronics will be conceptually modular, for example timing will be done by one circuit instead of being distributed as it is on the Apple II. The elimination of user options will streamline all aspects of the design.

5. Learning to use Macintosh

It is time to stop thinking exclusively in terms of writing manuals for computer systems. Manuals are just one means for accomplishing the real goal, which is to teach the user how to operate the software and hardware. For this teaching task, we must use whatever media are most effective within our cost and time constraints: it is essential to the success of Macintosh that it have a level of educational accessories (whether CAI, cassettes or whatever) beyond even what present Apple products do. A major portion of the effort in producing software for this product may go into its self-teaching aspects.

6. Personnel

At present Woz is working part-time on Macintosh. The detailed electronic design and breadboarding is being done by Burrell Smith. A software designer-programmer will be hired. This team, along with myself and a support person, will design and produce prototypes of the electronics, first drafts of the language and hardware manuals and will write the major portion of the software.

It is expected that these 4 full-time people will be able to carry the project through the majority of the hardware and software design and execution stages. The BASIC interpreter or compiler, the industrial designing, the disk and printer hardware, the analog video and power supply design may require the aid of engineering people outside the project personnel.

7. Accomplishments to date

The proportional font has been demonstrated; the majority of the electronics has been designed and breadboarded; preliminary specifications for the editor and a portion of the language have been prepared along with the outline and a few chapters of the user manuals. Much information pertaining to the software and hardware has been gathered, and many design choices [M2] have been made.

8. Immediate tasks

The following are some major tasks that should be completed in the next few months.

A. A breadboarded version of the computer will be completed, this should be done by 20 Feb 80, unless there are excessive delays in obtaining parts (Smith).

B. The editor portion of the software should be written to be demonstrated by 15 March 80. This also entails a preliminary keyboard design. (Raskin)

C. The UCSD Pascal system should be brought up on the breadboarded electronics by 1 April 80 (programmer to be hired).

D. A support person should be assigned to the project. Aside from being the project librarian, this person will do some engineering and research tasks (e.g. investigate current information services for small computer users).

E. A decision is required from engineering as to what kind of disk drives we can expect for this project. We prefer a dual disk drive on a single spindle using something of the power of our present 16 sector technology.

```
{

THE MACINTOSH PROJECT

DOCUMENT 24 VERSION 1

TITLE: MACINTOSH FONT GENERATING PROGRAM

AUTHOR:  JEF RASKIN

DATE: Dec 79

INTRODUCTION

With this program, on an Apple II screen, upper and lower case letters
(an average of 78 characters per line of normal English text) can be displayed.
On the 256 by 256 Macintosh display there will be at least 23 lines of
about 70 characters each, thus giving over 1600 characters on the screen.
}

(*$S+*)
PROGRAM MAKEFONT;

USES TURTLEGRAPHICS;

CONST INTERSPACE=1;
TALL=10;       (* Height of characters. *)
      VERTSPACE=1;  (* Pixels between successive lines. *)
      LEFT = 12;
      RIGHT = 267;
      BOTTOM = 0;
      TOP = 189;

TYPE BYTE=0..255;
     BITMAP=PACKED ARRAY[0.. 9] OF BYTE;
     FONTPOINTER=^FONT;
     FONT=RECORD
            WIDTHS: ARRAY[0..127] OF INTEGER;
            CHARDATA: ARRAY[0..127] OF BITMAP;
          END;

VAR CURRENTFONT: FONTPOINTER;

FUNCTION BINARY(S: STRING):INTEGER;
   VAR I,NUM: INTEGER;
BEGIN
  NUM:=0;
  FOR I:=LENGTH(S) DOWNTO 1 DO
    BEGIN
      NUM:=NUM*2;
      IF S[I]<>' ' THEN NUM:=NUM+1;
    END;
  BINARY:=NUM;
END;
```

```
PROCEDURE MAKECHAR(ASCII,WIDTH: INTEGER;
          S9,S8,S7,S6,S5,S4,S3,S2,S1,S0: STRING);

BEGIN
  WITH CURRENTFONT^ DO
    BEGIN
      WIDTHS[ASCII]:=WIDTH;
      CHARDATA[ASCII, 9]:=BINARY(S9);
      CHARDATA[ASCII, 8]:=BINARY(S8);
      CHARDATA[ASCII, 7]:=BINARY(S7);
      CHARDATA[ASCII, 6]:=BINARY(S6);
      CHARDATA[ASCII, 5]:=BINARY(S5);
      CHARDATA[ASCII, 4]:=BINARY(S4);
      CHARDATA[ASCII, 3]:=BINARY(S3);
      CHARDATA[ASCII, 2]:=BINARY(S2);
      CHARDATA[ASCII, 1]:=BINARY(S1);
      CHARDATA[ASCII, 0]:=BINARY(S0);
    END;
END;


PROCEDURE INIT3A;
BEGIN
  MAKECHAR(  3,0,'',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '');

  MAKECHAR(  8,0,'',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '');

  MAKECHAR( 14,0,'',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
```

```
                ' ',
                ' ',
                ' ',
                ' ');

MAKECHAR( 16,0,' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ');

MAKECHAR( 32,1,' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ');

MAKECHAR( 33,1,'@',
                '@',
                '@',
                '@',
                '@',
                ' ',
                '@',
                '@',
                ' ',
                ' ');

MAKECHAR( 34,3,'@ @',
                '@ @',
                '@ @',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ',
                ' ');

MAKECHAR( 35,5,' @ @',
                ' @ @',
                '@@@@@',
                ' @ @',
                ' @ @',
                '@@@@@',
```

```
                  ' @ @',
                  ' @ @',
                  '',
                  '');
END; (* INIT3A *)

PROCEDURE INIT3B;
BEGIN
  MAKECHAR( 36,5,'  @',
                  ' @@@',
                  '@ @ @',
                  '@ @',
                  ' @@',
                  '  @@',
                  '  @ @',
                  '@ @ @',
                  ' @@@',
                  '  @');

  MAKECHAR( 37,4,'@@ @',
                  '@@ @',
                  '  @',
                  '  @',
                  ' @',
                  ' @',
                  '@ @@',
                  '@ @@',
                  '',
                  '');

  MAKECHAR( 38,5,' @',
                  '@ @',
                  '@ @',
                  ' @',
                  '@ @ @',
                  '@  @',
                  '@ @ @',
                  ' @  @',
                  '',
                  '');

  MAKECHAR( 39,1,'@',
                  '@',
                  '@',
                  '',
                  '',
                  '',
                  '',
                  '',
                  '',
                  '');

  MAKECHAR( 40,3,'  @',
                  ' @',
```

```
                    '@',
                    '@',
                    '@',
                    '@',
                    ' @',
                    '  @',
                    '',
                    '');

  MAKECHAR( 41,3,'@',
                    ' @',
                    '  @',
                    '  @',
                    '  @',
                    '  @',
                    ' @',
                    '@',
                    '',
                    '');
END; (* INIT3B *)

PROCEDURE INIT4A;
BEGIN
  MAKECHAR( 42,5,'  @',
                    '@ @ @',
                    ' @@@',
                    '@ @ @',
                    '  @',
                    '',
                    '',
                    '',
                    '',
                    '');

  MAKECHAR( 43,3,'',
                    '',
                    ' @',
                    ' @',
                    '@@@',
                    ' @',
                    ' @',
                    '',
                    '',
                    '');

  MAKECHAR( 44,2,'',
                    '',
                    '',
                    '',
                    '',
                    '',
                    '',
                    ' @',
                    ' @',
                    '@',
```

```
                  '');

END; (* INIT4A *)


PROCEDURE INIT4B;
BEGIN
  MAKECHAR( 45,3,'',
                 '',
                 '',
                 '',
                 '',
                 '@@@',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '');

  MAKECHAR( 46,1,'',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '@',
                 '@',
                 '',
                 '');

  MAKECHAR( 47,4,'   @',
                 '   @',
                 '  @',
                 '  @',
                 ' @',
                 ' @',
                 '@',
                 '@',
                 '',
                 '');

  MAKECHAR( 48,4,' @@',
                 '@  @',
                 '@  @',
                 '@ @@',
                 '@@ @',
                 '@  @',
                 '@  @',
                 ' @@',
                 '',
                 '');

  MAKECHAR( 49,4,'  @',
                 ' @@',
                 '  @',
```

```
                    '  @',
                    '  @',
                    '  @',
                    '  @',
                    ' @@@',
                    '',
                    '');
END; (* INIT4B *)

PROCEDURE INIT5A;
BEGIN
  MAKECHAR( 50,4,' @@',
                    '@  @',
                    '   @',
                    '  @',
                    ' @',
                    '@',
                    '@',
                    '@@@@',
                    '',
                    '');

  MAKECHAR( 51,4,' @@',
                    '@  @',
                    '   @',
                    ' @@',
                    '   @',
                    '   @',
                    '@  @',
                    ' @@',
                    '',
                    '');

  MAKECHAR( 52,4,'   @',
                    '  @@',
                    ' @ @',
                    '@  @',
                    '@@@@',
                    '   @',
                    '   @',
                    '   @',
                    '',
                    '');

  MAKECHAR( 53,4,'@@@@',
                    '@',
                    '@',
                    '@@@',
                    '   @',
                    '   @',
                    '@  @',
                    ' @@',
                    '',
                    '');
```

```
  MAKECHAR( 54,4,' @@',
                 '@  @',
                 '@',
                 '@',
                 '@@@',
                 '@  @',
                 '@  @',
                 ' @@',
                 '',
                 '');
END; (* INIT5A *)

PROCEDURE INIT5B;
BEGIN
  MAKECHAR( 55,4,'@@@@',
                 '   @',
                 '   @',
                 '  @',
                 '  @',
                 ' @',
                 ' @',
                 ' @',
                 '',
                 '');

  MAKECHAR( 56,4,' @@',
                 '@  @',
                 '@  @',
                 ' @@',
                 '@  @',
                 '@  @',
                 '@  @',
                 ' @@',
                 '',
                 '');

  MAKECHAR( 57,4,' @@',
                 '@  @',
                 '@  @',
                 ' @@@',
                 '   @',
                 '   @',
                 '  @',
                 ' @',
                 '',
                 '');

  MAKECHAR( 58,1,'',
                 '',
                 '',
                 '@',
                 '@',
                 '',
```

```
                    '@',
                    '@',
                    '',
                    '');

  MAKECHAR( 59,2,'',
                    '',
                    '',
                    ' @',
                    ' @',
                    '',
                    ' @',
                    ' @',
                    '@',
                    '');
END; (* INIT5B *)


PROCEDURE INIT6A;
BEGIN
  MAKECHAR( 60,4,'',
                    '   @',
                    '  @',
                    ' @',
                    '@',
                    ' @',
                    '  @',
                    '   @',
                    '',
                    '');

  MAKECHAR( 61,3,'',
                    '',
                    '',
                    '@@@',
                    '',
                    '@@@',
                    '',
                    '',
                    '',
                    '');

  MAKECHAR( 62,4,'',
                    '@',
                    ' @',
                    '  @',
                    '   @',
                    '  @',
                    ' @',
                    '@',
                    '',
                    '');

  MAKECHAR( 63,4,' @@',
```

```
                    '@  @',
                    '   @',
                    '  @',
                    ' @',
                    '',
                    ' @',
                    ' @',
                    '',
                    '');

  MAKECHAR( 64,5,'',
                    ' @@@',
                    '@   @',
                    '@ @ @',
                    '@ @@',
                    '@',
                    '@',
                    ' @@@@',
                    '',
                    '');

  MAKECHAR( 65,4,' @@',
                    '@  @',
                    '@  @',
                    '@@@@',
                    '@  @',
                    '@  @',
                    '@  @',
                    '@  @',
                    '',
                    '');
END; (* INIT6A *)

PROCEDURE INIT6B;
BEGIN
  MAKECHAR( 66,4,'@@@',
                    '@  @',
                    '@  @',
                    '@@@',
                    '@  @',
                    '@  @',
                    '@  @',
                    '@@@',
                    '',
                    '');

  MAKECHAR( 67,4,' @@',
                    '@  @',
                    '@',
                    '@',
                    '@',
                    '@',
                    '@  @',
                    ' @@',
```

```
                    '',
                    '');

  MAKECHAR( 68,4,'@@@',
                    '@  @',
                    '@  @',
                    '@  @',
                    '@  @',
                    '@  @',
                    '@  @',
                    '@@@',
                    '',
                    '');

  MAKECHAR( 69,4,'@@@@',
                    '@',
                    '@',
                    '@@@',
                    '@',
                    '@',
                    '@',
                    '@@@@',
                    '',
                    '');

  MAKECHAR( 70,4,'@@@@',
                    '@',
                    '@',
                    '@@@',
                    '@',
                    '@',
                    '@',
                    '@',
                    '',
                    '');
END; (* INIT6B *)

PROCEDURE INIT7A;
BEGIN
  MAKECHAR( 71,4,' @@',
                    '@  @',
                    '@',
                    '@',
                    '@ @@',
                    '@  @',
                    '@  @',
                    ' @@',
                    '',
                    '');

  MAKECHAR( 72,4,'@  @',
                    '@  @',
                    '@  @',
                    '@@@@',
```

```
                  '@  @',
                  '@  @',
                  '@  @',
                  '@  @',
                  '',
                  '');

  MAKECHAR( 73,3,'@@@',
                  ' @',
                  ' @',
                  ' @',
                  ' @',
                  ' @',
                  ' @',
                  '@@@',
                  '',
                  '');

  MAKECHAR( 74,4,'   @',
                  '   @',
                  '   @',
                  '   @',
                  '   @',
                  '   @',
                  '@  @',
                  ' @@',
                  '',
                  '');

  MAKECHAR( 75,5,'@   @',
                  '@  @',
                  '@ @',
                  '@@',
                  '@ @',
                  '@  @',
                  '@   @',
                  '@   @',
                  '',
                  '');
END; (* INIT7A *)

PROCEDURE INIT7B;
BEGIN
  MAKECHAR( 76,4,'@',
                  '@',
                  '@',
                  '@',
                  '@',
                  '@',
                  '@',
                  '@@@@',
                  '',
                  '');
```

```
MAKECHAR( 77,7,'@     @',
                '@@   @@',
                '@@   @@',
                '@ @ @ @',
                '@ @ @ @',
                '@  @  @',
                '@     @',
                '@     @',
                '',
                '');

MAKECHAR( 78,5,'@   @',
                '@@  @',
                '@@  @',
                '@ @ @',
                '@ @ @',
                '@  @@',
                '@  @@',
                '@   @',
                '',
                '');

MAKECHAR( 79,4,' @@',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                ' @@',
                '',
                '');

MAKECHAR( 80,4,'@@@',
                '@  @',
                '@  @',
                '@@@',
                '@',
                '@',
                '@',
                '@',
                '',
                '');

END; (* INIT7B *)

PROCEDURE INIT8A;
BEGIN
MAKECHAR( 81,5,' @@',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
```

```
                '@  @',
                ' @@@',
                '    @',
                '');

  MAKECHAR( 82,4,'@@@',
                '@  @',
                '@  @',
                '@@@',
                '@@',
                '@ @',
                '@  @',
                '@  @',
                '',
                '');

  MAKECHAR( 83,4,' @@',
                '@  @',
                '@',
                ' @@',
                '   @',
                '   @',
                '@  @',
                ' @@',
                '',
                '');

  MAKECHAR( 84,3,'@@@',
                ' @',
                ' @',
                ' @',
                ' @',
                ' @',
                ' @',
                ' @',
                '',
                '');

  MAKECHAR( 85,4,'@  @',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                '@  @',
                ' @@',
                '',
                '');
END; (* INIT&A *)

PROCEDURE INIT8B;
BEGIN
  MAKECHAR( 86,5,'@   @',
                '@   @',
```

```
                    '@   @',
                    '@   @',
                    ' @ @',
                    ' @ @',
                    '  @',
                    '  @',
                    '',
                    '');

  MAKECHAR( 87,7,'@     @',
                    '@     @',
                    '@  @  @',
                    '@  @  @',
                    '@  @  @',
                    '@  @  @',
                    '@ @ @ @',
                    ' @   @',
                    '',
                    '');

  MAKECHAR( 88,5,'@   @',
                    '@   @',
                    ' @ @',
                    '  @',
                    '  @',
                    ' @ @',
                    '@   @',
                    '@   @',
                    '',
                    '');

  MAKECHAR( 89,5,'@   @',
                    '@   @',
                    '@   @',
                    ' @ @',
                    '  @',
                    '  @',
                    '  @',
                    '  @',
                    '',
                    '');

  MAKECHAR( 90,4,'@@@@',
                    '   @',
                    '  @',
                    '  @',
                    ' @',
                    ' @',
                    '@',
                    '@@@@',
                    '',
                    '');
END; (* INIT8B *)
```

```
PROCEDURE INIT9A;
BEGIN
  MAKECHAR( 91,3,'@@@',
                 '@',
                 '@',
                 '@',
                 '@',
                 '@',
                 '@',
                 '@@@',
                 '',
                 '');

  MAKECHAR( 92,4,'@',
                 '@',
                 ' @',
                 ' @',
                 '  @',
                 '  @',
                 '   @',
                 '   @',
                 '',
                 '');

  MAKECHAR( 93,3,'@@@',
                 '  @',
                 '  @',
                 '  @',
                 '  @',
                 '  @',
                 '  @',
                 '@@@',
                 '',
                 '');

  MAKECHAR( 94,5,'  @',
                 ' @ @',
                 '@   @',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '',
                 '');
END; (* INIT9A *)

PROCEDURE INIT9B;
BEGIN
  MAKECHAR( 95,4,'',
                 '',
                 '',
                 '',
                 '',
                 '',
```

```
                    '',
                    '',
                    '',
                    '',
                    '@@@@',
                    '');

  MAKECHAR( 96,2,'@',
                    '@',
                    ' @',
                    '',
                    '',
                    '',
                    '',
                    '',
                    '',
                    '');

  MAKECHAR( 97,3,'',
                    '',
                    '',
                    '@@@',
                    '  @',
                    '@@@',
                    '@ @',
                    '@@@',
                    '',
                    '');

  MAKECHAR( 98,3,'@',
                    '@',
                    '@',
                    '@@@',
                    '@ @',
                    '@ @',
                    '@ @',
                    '@@@',
                    '',
                    '');

  MAKECHAR( 99,3,'',
                    '',
                    '',
                    '@@@',
                    '@',
                    '@',
                    '@',
                    '@@@',
                    '',
                    '');
END; (* INIT9B *)

PROCEDURE INIT10A;
BEGIN
  MAKECHAR(100,3,'  @',
```

```
                  '  @',
                  '  @',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '',
                  '');

MAKECHAR(101,3,'',
                  '',
                  '',
                  '@@@',
                  '@ @',
                  '@@@',
                  '@',
                  '@@@',
                  '',
                  '');

MAKECHAR(102,3,' @@',
                  '@',
                  '@',
                  '@@',
                  '@',
                  '@',
                  '@',
                  '@',
                  '',
                  '');

MAKECHAR(103,3,'',
                  '',
                  '',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '  @',
                  '@@@');

MAKECHAR(104,3,'@',
                  '@',
                  '@',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@ @',
                  '',
                  '');
END; (* INIT10A *)
```

```
PROCEDURE INIT10B;
BEGIN
MAKECHAR(105,1,'',
                '@',
                '',
                '@',
                '@',
                '@',
                '@',
                '@',
                '',
                '');

MAKECHAR(106,2,'',
                ' @',
                '',
                ' @',
                ' @',
                ' @',
                ' @',
                ' @',
                ' @',
                '@@');

MAKECHAR(107,4,'@',
                '@',
                '@',
                '@  @',
                '@ @',
                '@@',
                '@ @',
                '@  @',
                '',
                '');

MAKECHAR(108,1,'@',
                '@',
                '@',
                '@',
                '@',
                '@',
                '@',
                '@',
                '',
                '');

MAKECHAR(109,5,'',
                '',
                '',
                '',
                '@@@@@',
                '@ @ @',
                '@ @ @',
                '@ @ @',
```

```
                  '@ @ @',
                  '',
                  '');
END; (* INIT10B *)


PROCEDURE INIT11A;
BEGIN
  MAKECHAR(110,3,'',
                  '',
                  '',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@ @',
                  '',
                  '');

  MAKECHAR(111,3,'',
                  '',
                  '',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '',
                  '');

  MAKECHAR(112,3,'',
                  '',
                  '',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '@',
                  '@');

  MAKECHAR(113,3,'',
                  '',
                  '',
                  '@@@',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '  @',
                  '  @');

  MAKECHAR(114,3,'',
                  '',
```

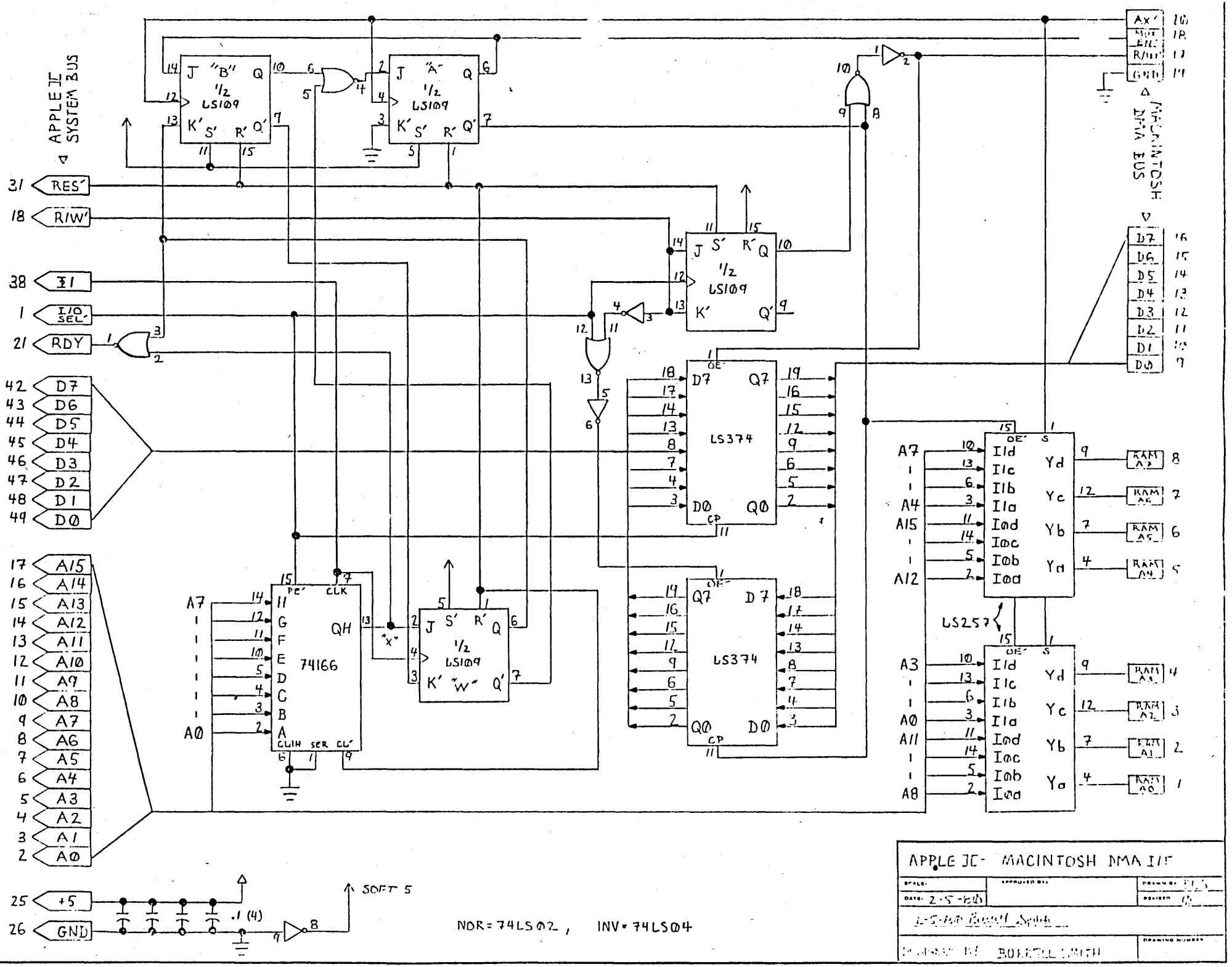
```
                  '',
                  '@ @',
                  '@@',
                  '@',
                  '@',
                  '@',
                  '',
                  '');
END; (* INIT11A *)

PROCEDURE INIT11B;
BEGIN
MAKECHAR(115,3,'',
                  '',
                  '',
                  '@@@',
                  '@',
                  '@@@',
                  '  @',
                  '@@@',
                  '',
                  '');

MAKECHAR(116,2,'@',
                  '@',
                  '@',
                  '@@',
                  '@',
                  '@',
                  '@',
                  ' @',
                  '',
                  '');

MAKECHAR(117,3,'',
                  '',
                  '',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '',
                  '');

MAKECHAR(118,3,'',
                  '',
                  '',
                  '@ @',
                  '@ @',
                  '@ @',
                  ' @',
                  ' @',
                  '',
```

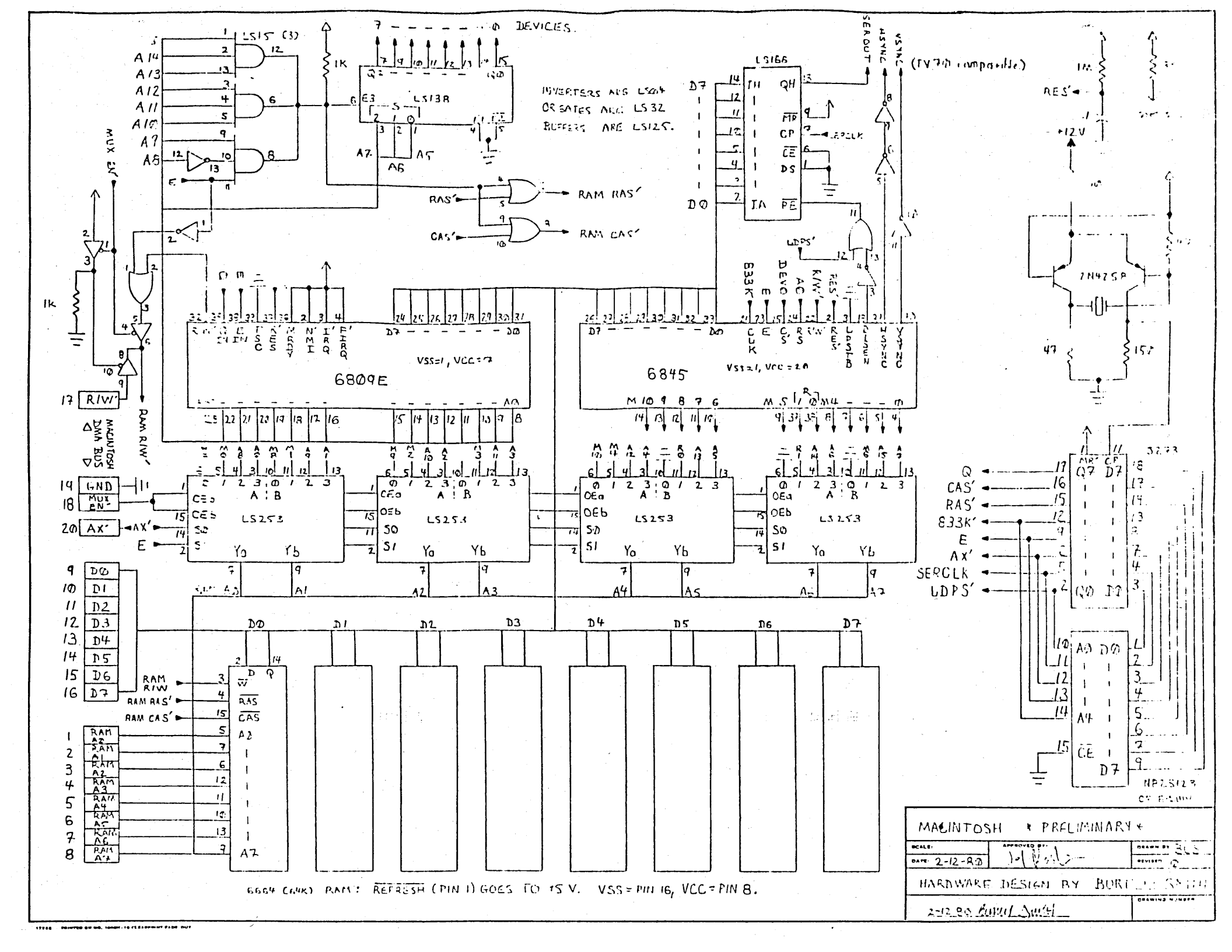
```
                  '');

  MAKECHAR(119,5,'',
                  '',
                  '',
                  '@   @',
                  '@ @ @',
                  '@ @ @',
                  '@ @ @',
                  ' @ @',
                  '',
                  '');
END; (* INIT11B *)


PROCEDURE INIT12A;
BEGIN
  MAKECHAR(120,3,'',
                  '',
                  '',
                  '@ @',
                  '@ @',
                  ' @',
                  '@ @',
                  '@ @',
                  '',
                  '');

  MAKECHAR(121,3,'',
                  '',
                  '',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@ @',
                  '@@@',
                  '  @',
                  '@@@');

  MAKECHAR(122,3,'',
                  '',
                  '',
                  '@@@',
                  '  @',
                  ' @',
                  '@',
                  '@@@',
                  '',
                  '');

  MAKECHAR(123,3,'  @',
                  ' @',
                  ' @',
                  ' @',
```

```
                    '@',
                    ' @',
                    ' @',
                    ' @',
                    '  @',
                    '');

  MAKECHAR(124,1,'@',
                    '@',
                    '@',
                    '@',
                    '@',
                    '@',
                    '@',
                    '@',
                    '@',
                    '');
  END; (* INIT12A *)

  PROCEDURE INIT12B;
  BEGIN
  MAKECHAR(125,3,'@',
                    ' @',
                    ' @',
                    ' @',
                    '  @',
                    ' @',
                    ' @',
                    ' @',
                    '@',
                    '');

  MAKECHAR(126,5,'',
                    ' @',
                    '@ @ @',
                    '   @',
                    '',
                    '',
                    '',
                    '',
                    '',
                    '',
                    '');
END; (* INIT12B *)

PROCEDURE INITCHARS;
BEGIN
  INIT3A; INIT3B;
  INIT4A; INIT4B;
  INIT5A; INIT5B;
  INIT6A; INIT6B;
  INIT7A; INIT7B;
  INIT8A; INIT8B;
  INIT9A; INIT9B;
  INIT10A; INIT10B;
```

```
  INIT11A; INIT11B;
  INIT12A; INIT12B;
END;

PROCEDURE NEWLINE;
BEGIN
  MOVETO(LEFT,TURTLEY-(TALL+VERTSPACE));
END;

PROCEDURE WCHAR2(CH: CHAR);
BEGIN
  DRAWBLOCK(CURRENTFONT^.CHARDATA[ORD(CH)],1,0,0,
    CURRENTFONT^.WIDTHS[ORD(CH)]+INTERSPACE,TALL,TURTLEX,TURTLEY, 5);
  IF TURTLEX < (RIGHT - 8)
    THEN MOVETO(TURTLEX+CURRENTFONT^.WIDTHS[ORD(CH)]+INTERSPACE,TURTLEY)
    ELSE NEWLINE;
END;

PROCEDURE HOME;
BEGIN
  MOVETO(LEFT,TOP-(TALL+VERTSPACE));
END;


PROCEDURE POKE(LOC,VALUE:INTEGER);
TYPE WINDOW = PACKED ARRAY[0..0]OF CHAR;
VAR ADDR:INTEGER;
P: ^WINDOW;
BEGIN
  ADDR := LOC;
  MOVELEFT (ADDR,P,2);
  P^[0] := CHR(VALUE);
END; (*POKE*)

PROCEDURE SEND(CH: CHAR);
BEGIN
  UNITWRITE (6,CH,1,0,12);
END;

PROCEDURE UNIDIRECTIONAL;
CONST MAXBYTE = 255;
      DIRECTION = -12529;
BEGIN
  POKE (DIRECTION,MAXBYTE)
END; (* UNIDIRECTIONAL *)


PROCEDURE PRINTPIC;
CONST CQ = 17;
BEGIN
  UNIDIRECTIONAL;
  SEND (CHR (CQ));(* DISPLAY THE PICTURE *)
END;
```

```
PROCEDURE INTENSITY (DARK:INTEGER);
CONST INTEN = -12528;
BEGIN
  IF DARK < 0 THEN DARK := 0;
  IF DARK > 7 THEN DARK := 7;
  POKE (INTEN,DARK);
END;

PROCEDURE ECHO;
VAR I:INTEGER;
    CH:CHAR;
BEGIN
  INITTURTLE;
  FILLSCREEN (REVERSE);
  HOME;
    REPEAT
      READ (KEYBOARD,CH);
      IF EOLN(KEYBOARD)
        THEN
          BEGIN
            NEWLINE;
            WRITELN;
          END
        ELSE
          BEGIN
            WCHAR2(CH);
            WRITE (OUTPUT,CH);
          END;
      CASE ORD(CH) OF
        8:   MOVETO(TURTLEX-2,TURTLEY); (* graphic backspace *)
        14:  FILLSCREEN (REVERSE);
        16:  HOME;
        102: MOVETO(TURTLEX-1,TURTLEY); (* ligature on lower case "f" *)
      END (* OF CASES *)
    UNTIL ORD(CH)=3;
  intensity (6);
  printpic;
  TEXTMODE;
END;



BEGIN (* MAIN PROGRAM *)
  NEW(CURRENTFONT);
  INITCHARS;
  WRITELN;
  WRITELN ('START TYPING.  CTRL-C TO QUIT, CTRL-N FOR BLACK-ON-WHITE.');
  WRITELN ( CTRL-P HOMES, CTRL-H (LEFT ARROW) BACKSPACES GRAPHICALLY.');
  ECHO;
  WRITELN;
  WRITELN('TEST COMPLETE');
END.
```


APPLE IIc SYSTEM BUS
MACINTOSH DATA BUS
RES'
R/W'
Ø1
I/O SEL'
RDY
D7
D6
D5
D4
D3
D2
D1
D0
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
+5
GND
.1 (4)
SOFT 5
J "B" Q
1/2 LS109
K' S' R' Q'
J "A" Q
J S' R' Q
K'
LS374
74166
QH
"X"
"W"
LS257
NOR = 74LS02 , INV = 74LS04
APPLE IIc - MACINTOSH DMA I/F
2-5-86
BURRELL SMITH


LS15 (3)
A14
A13
A12
A11
A10
A9
A8
1K
LS138
A7
A6
A5
7 ... 0 DEVICES
INVERTERS ARE LS04
OR GATES ARE LS32
BUFFERS ARE LS125.
RAS'
CAS'
RAM RAS'
RAM CAS'
MUX 2X'
R/W'
RAM R/W'
MACINTOSH DMA BUS
6809E
VSS=1, VCC=2
6845
VSS=1, VCC=2.0
LS166
SERDAT
HSYNC
VSYNC
(TV720 compatible)
SERCLK
LDPS'
B33K'
E
DEN'
AC
RW'
RES'
CURSB
DISEN
HSYNC
VSYNC
LS253
GND
MUX EN'
AX'
D0
D1
D2
D3
D4
D5
D6
D7
RAM R/W
RAM RAS'
RAM CAS'
RAM A2
RAM A1
RAM A0
RAM A3
RAM A4
RAM A5
RAM A6
RAM A7
A7
6664 (64K) RAM: REFRESH (PIN 1) GOES TO +5 V. VSS = PIN 16, VCC = PIN 8.
+12.V
RES'
2N4258
47
15K
Q
CAS'
RAS'
833K'
E
AX'
SERCLK
LDPS'
NPC5128
MACINTOSH * PRELIMINARY *
SCALE:
DATE: 2-12-80
APPROVED BY:
DRAWN BY: BLS
REVISED:
HARDWARE DESIGN BY BURT

Tom -
Latest Mac document
Jef

THE MACINTOSH PROJECT

DOCUMENT 25 VERSION 1

TITLE: THE COMPLETELY DISTRIBUTED COMMUNICATIONS NETWORK

AUTHOR: JEF RASKIN

DATE: 28 FEB 1980

0. INTRODUCTION

Previous Macintosh documents have made a case for the establishment of a communications network as being essential to the large- scale sales of personal computers. This realization is not unique to Apple, of course, but is being increasingly recognized by many segments of the computing community. The most important question is: How can Apple establish communications networks for our customers in a timely and economical fashion?

1. WHAT IS A COMMUNICATIONS NETWORK?

A communications network consists of addressors who generate messages and addressees for whom the messages are intended; and a set of potential pathways that join all possible pairs of addressors and addressees. Addressors and addressees are usually people. Each person in the network has both a sender and a receiver which are devices that effect the generation, transmission and delivery of messages.

An addressor generates a message, transmits it over a pathway by means of the sender; an addresee receives the message on the receiver. This formalism describes, for example, the familiar telephone communication network. The usual television and radio "networks" are not communications networks in the sense being discussed here since they are unidirectional.

We must distinguish, in our communications network, between the sending of a message, its transmission, and its reception. A message is sent when the addressor consigns it to the network. The network may have to store the message until a transmission pathway becomes available. Thus the time at which a message is sent may not be the same as when it is transmitted. In any case, the two actions are distinguishable. (We can further distinguish between the time a message is generated , and the time it is sent.)

Likewise, at the receiving end, the receiver or addressee might not be available. Thus the network (or the receiver) may have to store it. The time at which a message is transmitted may not be the same as when it is received. And the time at which it is acquired by the receiver may not be the same as the time when it is (finally) read by the addressee. Again, in any case, the actions of reception and reading differ.

These distinctions are, of course, not necessary with the usual use of the telephone network, which may be described as a "real time" network. One of the benefits of a non-real-time network is immediately apparent to anyone who

has had to work across time zones--or who wishes to speak to someone on a radically different schedule.

In summary, then, the actions are:

Generation --> Sending --> Transmission --> Reception --> Reading

The corresponding actors are:

Addressor.....Sender.....Pathway.....Receiver.....Addressee

1.1 THE ABILITIES OF A NETWORK

The personal computer network must have these abilities:

a. An addressor may, at any time, generate and, at that or any later time send a message.

b. An addressor may inquire as to whether a message has been read by the addressee, and possibly inquire as to the exact state of the message (sent or not, in transmission, received or not, read or not).

c. An addressee may inquire as to the presence of received messages for him or her, and read any or all of them at will.

2. WHY WE MUST HAVE PERSONAL COMPUTER NETWORKS

It is pretty clear from the amount of activity going on in the field that this is a "hot" area. The reason that it is so popular is that there is not much most individuals can do with a single, isolated computer. Yet, with access to a number of data bases and message sending and receiving ability, it is possible to see many uses of personal computers that are neither esoteric nor difficult. A previous Macintosh paper discusses this issue.

3. THE COMPLETELY DISTRIBUTED NETWORK

The ordinary telephone network is not distributed at all (from the point of view of the user). Each telephone has only the ability to passively receive a message, or to initiate the sending of one. What intelligence the system has, and its control, resides elsewhere.

It is useful to define a node as a place on the network where one or more of message generation, sending, reception, and reading may take place. A node may, as a consequence, perform a switching function--i.e. receiving a message and then sending it on one or more pathways for whatever reason.

A terminal node has exactly one bi-directional path leading to it.

A distributed system is one where each terminal node has some control, but where there are non-terminal nodes which can control the flow of information about the network. These non-terminal nodes may also generate, store, monitor or act on messages.

In a completely distributed network, all intelligence and control resides

at terminal nodes.

In the completely distributed network being considered here, the terminal nodes are personal computers, and the pathways are ordinary telephone lines.

4. HOW THIS COMPLETELY DISTRIBUTED NETWORK OPERATES

The terminal node for each person is a computer. When an addressor sends a message his or her computer stores it. The message contains the "address" of the addressee, which in this case is probably the addressee's telephone number. The computer (as soon as it can) attempts to dial the addressee's phone. It can recognize whether or not the phone is answered by the addressee's computer. If it is not properly answered, it tries again after a certain delay (there are reasons for making the delay a constant plus a random time). It will continue making these attempts until the message is received and acknowledged--or until the message is deleted from the list of messages waiting to be sent by the addressor.

Since the computer has a real-time clock, it can be instructed to wait until low night telephone rates are in effect (or until some other condition is met) before it transmits messages.

At the receiving end, the computer, when an incoming call arrives, determines whether it is a computer call or a human call. If it is a human call, it allows the telephone bell to ring, and ignores the call thereafter. If it is a computer call, it stores the message, and sends out the acknowledgement message. If it has been instructed to do so, it will display the message as it arrives, if not, it will store it until the addressee wishes to see it.

4.1 SOME PROBLEMS WITH A COMPLETELY DISTRIBUTED NETWORK

If a message is sent, but the receiving computer is not attached, a person on the receiving end may answer the phone and discover a tone. If the receiver is available, there should be enough time allowed to plug it in, but if not, the call will be wasted. This is a definite problem with a completely distributed network given current phone tariffs and technology.

A broadside (a message to many addressees) can be sent by having a file of many telephone numbers, all of which are sent the same message. But it is in the area of dealing with classes of users that the completely distributed system is weakest compared to a partially distributed system. For example, if a person wanted to correspond with all others interested in, say, butterfly collecting, it would be relatively easy for a central computer to put out such a notice--since it would have the addresses of all users. In a completely distributed system it would take many fruitless calls to try to find these users.

On the other hand, the completely distributed network assures a great degree of privacy to the individuals involved.

A message may be forwarded by merely indicating to the addressee that it is to be forwarded. I suspect that any automatic forwarding scheme will unduly raise the hackles of the phone company.

5. SECURITY

Security is not a responsibility of the network, but of the individual users of the network. All users of the network must assume that transmissions are public. Therefore, a user wishing to make a secure communication must take responsibility for encryption and subsequent decoding by the addressee.

This shifts the burden of privacy away from the network itself. It might be observed that security software is a salable commodity.

6. LEGAL AND REGULATORY PROBLEMS

The use of computers over the telephone network poses a number of problems for the telephone company. As discussed in a earlier Macintosh document, the useage statistics are skewed when computers talk to eachother. In addition, some time-slice methods now in use for compressing many conversations into a single broadband channel may play havoc with schemes for getting greater transmission speeds (in excess of 300 baud) between terminal nodes. There may even be difficulty at 300 baud.

Aside from any genuine difficulties that a computer communications network may cause the telephone system, we must be concerned with the possibility of their attempting to derive added revenues whenever digital communications are used. This, too, was discussed in document M3, to which you are referred.

There are pathways other than the telephone system, but they may not be in place quickly enough for our present needs.

7. DATA BASES

A data base may be considered a "person" who is the addressee of requests for information, and who responds by sending messages consisting of the desired information. No new network protocol is required--but the form of messages received by the data base will have to be carefully contrived.

Companies that provide data bases may also provide message services. The use of data bases is discussed in M3, which includes a list of applications as well as possible vendors.

8. THE REQUIRED INFORMATION EACH MESSAGE CONTAINS

A protocol will have to be established so that messages contain sufficient information to keep the network in operation. For example:

a. Date and time sent.

b. Addressor

c. Addressee (which may be a class)

d. items a, b and c of the message being responded to (if any).

The questions of protocols, both electronic as well as interpersonal have been widely discussed. A planned (but not yet promised) Macintosh document may

cover this in greater detail. In the meantime, a model based on the PCNET (P. O. Box E, Menlo Park, CA 94025) protocols or the Arpanet heading conventions may be kept in mind (See The Network Nation by Hiltz and Turoff).