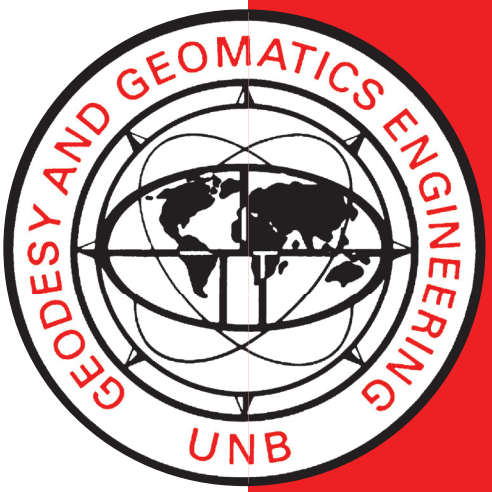
A DIGITAL DATA RECORDER AND TRANSFER DEVICE FOR THE MARCONI 722B SATELLITE NAVIGATION RECEIVER

MARK STEVEN LORD

April 1982



TECHNICAL REPORT
NO. 88

PREFACE

In order to make our extensive series of technical reports more readily available, we have scanned the old master copies and produced electronic versions in Portable Document Format. The quality of the images varies depending on the quality of the originals. The images have not been converted to searchable text.

A DIGITAL DATA RECORDER AND TRANSFER DEVICE FOR THE MARCONI 722B SATELLITE NAVIGATION RECEIVER

by

Mark Steven Lord

for

CS 4993

Fourth Year Undergraduate Project

School of Computer Science

Fredericton, New Brunswick, 1982

Reprinted July 1985

ABSTRACT

Dr. David Wells of the Department of Surveying Engineering uses a MARCONI 722B Satellite Navigation Receiver in his research. At present, data output from this device is recorded using a paper tape punch. The paper tape produced from the punch is then fed into the IBM 3032 at the U.N.B. Computing Center for calculations and analysis. Needless to say, this method has proved to be somewhat awkward and cumbersome.

The aim of this project is to provide a convenient alternative to the current system, using a micro-computer to collect data directly from the satellite receiver, and then transfer this data to the IBM 3032 through the VSPC online terminal system.

ACKNOWLEDGEMENTS

I would like to extend a special "Thank-You" to each of the following people, without whose patient answers to my many questions, this project would have taken even longer to complete:

Dr. B. J. Kurz, Computer Science

Dr. David Wells, Surveying Engineering

Murray Linton, Psychology Technician

CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iii

Chapter Page

1. INTRODUCTION 1

The Transit Satellite System 1
Transit At U.N.B. 2
Data Format and Transfer 2
An Alternative to Paper Tape 6

2. SYSTEM PERFORMANCE SPECIFICATIONS 7

System Hardware Specifications 7
Hardware Requirements 7
The Microprocessor 7
Interfacing Capability 7
Auxiliary Storage 8
Operating System 9
Future Expansion 9
Reliability and Service 10
System Candidates 10
The Commodore CBM 4032 10
The Apple II Plus 11
The TRS-80 Model III 13
System Software Specifications 13
Required Capabilities 13
Data Acquisition and Formatting 14
Error Detection and Recovery 14
Data Storage 15
Operator Review of Data 15
Transfer of Data to IBM 3032 16
Optional Capabilities and/or Provision for Future Expansion 16
Use of a Real Time Clock 16
Computer Feedback to Receiver 17
Data Compression 17
Operator Editing of Pass Files 17
Computing Receiver Positions 18

3. SYSTEM HARDWARE DESCRIPTION 19

The Winning Candidate: Apple II Plus 19
Physical Connections 21
Adapter for HP-1000 Connector on Receiver Cable 21
RS-232 Connector for Modem 23
Cable Clamp for Diskette Drive Ribbon Cables . 24
Modification To PIA Ribbon Cable 24
Chip Replacement On Disk Controller 24
The "Red" Switch on the Language Card 25
Null Modem for Connecting a Printer 26
Parallel Echo-Back Connector 26
Computer Feedback to Receiver 26
Batteries For Calendar/Clock Module 27

4. SYSTEM SOFTWARE DESCRIPTION 28

The RECEIVER Program 28
Running the RECEIVER Program 28
RECEIVER Program Logic Explanations 31
Message Synchronization 31
End of Pass and Timing Words 32
The Screen Display 32
SATLITE Assembly Language Procedures 36
INITPIA 37
IRQHANDL 37
GETWORD 38
RESETIRQ 39
Files Used by the RECEIVER Program 40
The Screen File 40
The Parameter File 40
Format of Pass Files 43
The TALK Program 45
Dumb Terminal Mode 45
The Escape Key 46
The Left Arrow 46
The Right Arrow 46
Hitting Control "C" 46
Passfile Transfer Mode 47
Text File Transfer Mode 47
Other Programs 47
The PEEKPOKE Intrinsic Unit 48
The READTIME Procedure 48
The STARTUP Program 49
The SETTIME Program 50

5. CONCLUSIONS 51

Appendix Page

I. OPERATING INSTRUCTIONS 52

Hardware Connections 53
Connection to the Satellite Receiver 53
Connection of an Acoustic Coupler or Modem . . 53
Connection to Other RS-232 Devices 53
General Considerations 54
System Diskettes 54
The SYSRUN Diskette 54
The SYSTXT Diskette 55
The SYSLIB Diskette 55
The SYSBKP Diskette 55
Booting The System 55
Special PASCAL System Keys 56
Using the RECEIVER Program 58
Functions of RECEIVER 58
Running the RECEIVER Program 58
Files Required at Initialization 58
Files Required After Initialization 58
Creation of Pass Files 59
Operator Control of RECEIVER 60
The "Q" Command 60
The "S" Command 60
The "K" Command 60
The "U" Command 61
Using the TALK Program 61
Functions of TALK 61
Running the TALK Program 61
Files Required at Initialization 62
Files Required After Initialization 62
Main Menu Options 62
Dumb Terminal Mode 63
Pass File Transfer Mode 63
Text File Transfer Mode 64
Quitting the TALK Program 65

II. PROGRAM LISTINGS 66

RECEIVER Program Listing 67
SATLITE Routines Listing 79
TALK Program Listing 84
STARTUP Program Listing 94
PEEKPOKE Unit Listing 96
READTIME Routine Listing 98

BIBLIOGRAPHY 100

LIST OF FIGURES

Figure		Page
1.	Composition of A Data Line From Nine Input Words	4
2.	Non-data Sequence Codes and Interpretations	5
3.	Data Storage Requirements	8
4.	List of Major Purchases	20
5.	Pin Connections on HP-1000 Connector	22
6.	Connections from Receiver Cable to Apple Computer	23
7.	Operator Escape Commands For The RECEIVER Program	30
8.	Special Input Words	34
9.	Significance of STATUS Display	35
10.	Contents of The RCV.SCREEN File	41
11.	Construction of Pass File Names	43
12.	Pass File Produced by RECEIVER Program	44

Chapter 1

INTRODUCTION

1.1 THE TRANSIT SATELLITE SYSTEM

TRANSIT is the name of a system of navigation satellites originally developed for use by the United States Navy to help guide their POLARIS submarines. Released for non-military use in 1967, the system is now widely used throughout the world, for many applications where accurate position determination is necessary. Each of the seven satellites currently in use maintains a circular polar orbit about the Earth. At about 1075 kilometers up, each satellite completes an orbit every 107 minutes. These satellites are constantly transmitting 150 Mhz and 400 Mhz signals as well as timing marks, and a navigation message. This navigation message contains information about the satellite's predicted orbit, known as orbital parameters, which are updated every twelve hours by a ground station in California.

Whenever a satellite rises above the horizon, a receiver station has the opportunity to obtain a position fix based on data collected during the satellite's "pass". By measuring the doppler frequency shift caused by the satellite's motion as a function of time, complex forumlae can then be used to compute the position of the receiver relative to

the satellite. Since each satellite follows a known orbit, one can compute the position of the receiver relative to the Poles, giving latitude and longitude measurements. The accuracy of these measurements is dependent on how many passes are observed.

1.2 TRANSIT AT U.N.B.

Here at U.N.B. we have a MARCONI receiver which is used by the Department of Surveying Engineering for obtaining accurate position fixes when doing surveying. This receiver measures the doppler shift on each of the two carrier frequencies, decodes the satellite message, and also receives the two-minute timing marks. All of this data is passed through an interface section to a paper tape punch, which is used as the primary means of recording data. Optionally, a teletype can be connected to the receiver, allowing human-readable output. The receiver also has a computer interface section, originally designed for an HP-1000 mini-computer. It is with this interface that the remainder of this report is concerned.

1.3 DATA FORMAT AND TRANSFER

The output from the computer interface consists of an eighteen wire cable with sixteen data wires, a control wire, and a ground connection. Data is transmitted in parallel as 16-bit words, where a low voltage indicates a binary one,

and a high voltage indicates a zero. A high to low voltage transient on the control wire indicates that data is ready to be transfered. The data is unbuffered, and no handshaking operations with the receiver are possible.

Data words are generated by the interface approximately every half-second whenever a satellite is being monitored. Each 16-bit word is logically divided into four 4-bit parts. The most significant 4-bits represent a sequence code used for synchronization purposes. The three remaining groups of 4-bits represent data digits in Binary Coded Decimal format. Groups of nine data words make up a "line" of data, and a sequence of twenty-five lines forms a "paragraph". A complete paragraph is sent over a period of exactly two minutes, and up to nine such paragraphs are transmitted for each satellite pass.

Although nine data words make up a line, not all of the BCD digits are used to communicate meaningful data. Figure 1 shows the format of the nine words which make up a line of data.

All other sequence codes which are transmitted can be interpreted as shown in Figure 2 .

Sequence Code	Data In BCD Format	Interpretation
0 1 0 1	D D D D D D D D D D D D	First 3 digits of 400 Mhz count
0 1 1 0	D D D D D D D D D D D D	Next 3 digits of 400 Mhz count
0 1 1 1	X X X X X X X X D D D D	Last digit of 400 Mhz count
1 0 0 1	D D D D D D D D D D D D	First 3 digits of 150 Mhz count
1 0 1 0	D D D D D D D D D D D D	Next 3 digits of 150 Mhz count
1 0 1 1	X X X X X X X X D D D D	Last digit of 150 Mhz count
1 1 0 1	D D D D D D D D D D D D	First 3 digits of satellite msg
1 1 1 0	D D D D D D D D D D D D	Next 3 digits of satellite msg
1 1 1 1	D D D D D D D D D D D D	Last 3 digits of satellite msg

Note that 'X' is used to indicate bits whose values are not used, and 'D' is used to indicate those bits which represent valid data.

Figure 1: Composition of A Data Line From Nine Input Words

Sequence Code	Data In BCD Format	Interpretation
0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	Receiver 2 minute timing word
0 0 0 0	X X X X X X X X X X X X	Test mode 2 minute timing word
0 0 0 1	X X X X X X X X X X X X	Invalid code
0 0 1 0	X X X X X X X X X X X X	Invalid code
0 0 1 1	X X X X X X X X X X X X	Invalid code
0 1 0 0	X X X X X X X X X X X X	Invalid code
1 0 0 0	X X X X X X X X X X X X	Satellite 2 minute timing word
1 1 0 0	X X X X X X X X X X X X	End of pass

Note that 'X' is used to indicate bits whose values are not used.

Figure 2: Non-data Sequence Codes and Interpretations

1.4 AN ALTERNATIVE TO PAPER TAPE

Having once worked at the Bedford Institute in Halifax, where an HP-1000 minicomputer performs the data recording and analysis, Dr. Wells would like to see a computerized alternative available at U.N.B. as well. Thus, it came to pass that the task was offered as a possible undergraduate project for a fourth year computer science student to work on.

Chapter 2

SYSTEM PERFORMANCE SPECIFICATIONS

2.1 SYSTEM HARDWARE SPECIFICATIONS

2.1.1 Hardware Requirements

The following hardware constraints were discussed with Dr. Kurz and Dr. Wells, and using these, three system candidates were examined, and a decision was made as to which system would prove most effective, with regard to both cost and performance.

2.1.1.1 The Microprocessor

The microprocessor itself should be sufficiently capable of handling all of the various I/O devices discussed below. The instruction execution rate must be fast enough to keep up with incoming data so that data words are not lost in the middle of a pass. An ASCII keyboard and good quality video monitor are also necessities. Graphical capabilities, although not needed for this task, would also be of benefit, as would a real time clock.

2.1.1.2 Interfacing Capability

The chosen system must be capable of being physically interfaced to the satellite receiver, as well as to the standard

keyboard, monitor, and diskette drives. An RS-232 compatible port must also be included, for communication with VSPC and possibly other computer installations.

2.1.1.3 Auxiliary Storage

The system should be capable of storing, without operator intervention, at least a single day's data, and preferably twice that amount. At the North Pole, such a system would observe about fifty passes each day. Here at U.N.B., one might observe only about twenty. Based on the North Pole maximum, the amount of storage required for data would be approximately 230,000 bytes for each day of observations.

```
        One data line consists of 23 digits
        One paragraph consists of 25 lines
        A single pass consists of  8 paragraphs
        A day of data consists of 50 passes

  Total data per day = 50 * 8 * 25 * 23 = 230,000 digits.

   If data is stored as ASCII characters, then the total
  storage required, less overhead and program storage,
  would be about 460,000 bytes for two days of data.
  However, with overhead, this figure will probably come
  closer to 500,000 bytes of data storage alone, still
  not taking into account program storage requirements.
```

Figure 3: Data Storage Requirements

Data transfers to auxiliary storage must be done fast enough so that pass data is not lost. In order to allow for easy access to this data, some type of random access storage device should be used, either a small hard disk, or a multiple drive floppy disk system, either of which could handle the above amounts of data.

2.1.1.4 Operating System

The chosen manufacturer must also be able to provide a flexible operating system or monitor, capable of recognizing and communicating with the various devices listed above. It must also support a high level programming language, and should provide interfacing capability to machine language programs written to handle interrupts from the satellite receiver. A completely interrupt driven I/O structure would also be desirable, to help prevent loss of incoming receiver data during disk transfers. Software support of fast disk transfers is also desirable. More on software specifications later.

2.1.1.5 Future Expansion

The system should also be capable of being applied to other tasks in the future, such as controlling other devices when not being used for satellite data acquisition. Capacity for connecting other peripherals such as a printer and plotter should also be considered.

2.1.1.6 Reliability and Service

The unit chosen should have a good reliability reputation. Servicing, if ever needed, should be locally available, as opposed to a central depot in Toronto, for example. The possiblity of a dealer's service contract should also be examined.

2.1.2 System Candidates

Using the above hardware specification, three popular, commercially available microprocessors were examined, and from these, the final system was chosen. The only cost constraint was that the final system should cost no more than about seven or eight thousand dollars, since funding was being provided by a sum of grant money in that range.

2.1.2.1 The Commodore CBM 4032

The first candidate is the CBM 4032 from Commodore Business Machines. It uses a 6502 microprocessor as its brain, and supports up to 32K of user RAM. All I/O is interrupt-driven, and external devices may connect either to the IEEE-488 bus, or directly to the CBM's internal bus. The main processor unit also houses an ASCII keyboard, a small forty-column video display, and a real time clock. The operating system, resident in ROM, includes an assembly language monitor, and a customized version of Dartmouth BASIC as its primary high level language.

The CBM 8050 Floppy Disk Subsystem, which connects to the IEEE-488 bus, has twin double-sided, double-density five inch drives, as well as its own internal CPU and memory to lighten the load on the main computer. With a total storage capacity of over one megabyte per two drives, this system definitely has the best storage capabilities out of all of the systems examined.

An interface for receiving data from the satellite receiver would have to be designed and built at U.N.B.. A serial communications interface, complete with driving software, is available directly from Commodore. Such a system could be puchased and serviced locally, through a local business in downtown Fredericton, for aproximately $4600 for the computer, diskette drives, and serial interface.

2.1.2.2 The Apple II Plus

The second candidate considered was the Apple II Plus, from Apple Computer Inc.. This also uses a 6502 processor, and comes with a built in keyboard. A CRT monitor must be purchased separately, with several black & white or color monitors from which to choose. The Apple's internal hardware generates a forty column color display, with no lowercase capability. The Apple provides hardware and software support for on-screen color graphics, in either HI-RES or LO-RES modes, with up to sixteen different colors. All I/O

on the Apple is done without using interrupts, although use of interrupts is allowed for controlling non-standard I/O under control of user written programs.

The Apple may be purchased with up to 48K of user RAM, 12K of which is taken up by the DOS 3.3 monitor. Applesoft floating point basic comes in ROM, and other languages are available with the addition of a 16K bank-selected Language Card. The best of these languages is Apple PASCAL, a modified version of the UCSD PASCAL language and operating system. With PASCAL, a powerful editor/assembler/compiler combination can be used, and a full 48K of RAM is available for user programs. The Apple has numerous I/O facilities, including a versitile set of game I/O controls, a built-in speaker, outputs for an external speaker, a cassette interface, and eight slots for direct connection to the Apple's internal bus structure. Unfortunately, once these eight slots get full, they cannot be easily expanded. Several interface cards are available for the Apple, including serial communications interfaces, parallel I/O interfaces, and real time clocks.

The Apple DISK-II floppy diskette drives are capable of storing about 140K bytes of data per diskette, but must also hold the disk operating system in this space. For every two drives used, the Apple must have a floppy disk controller card plugged into one of the eight slots inside the machine.

There now exists a local Apple dealer, and a User's Group which meets regularly, both on and off campus. A complete system, consisting of the computer, a nine inch black and white video monitor, a serial RS-232 interface card, a parallel interface card, a real time calendar/clock card, four diskette drives, and two floppy disk controller cards would cost about $5500 and would, at least in part, have to be ordered from catalogs.

2.1.2.3 The TRS-80 Model III

The final "candidate", the TRS-80 Model III, uses a more powerful Z-80 microprocessor, but also uses low capacity diskette drives similar to those for the Apple. This system was not given serious consideration because of the lack of technical information available (none) on the internal hardware configuration. Such information would be necessary to perform interfacing to the satellite receiver.

2.2 SYSTEM SOFTWARE SPECIFICATIONS

This section describes the software requirements of the proposed system.

2.2.1 Required Capabilities

The following capabilities were deemed to be necessary functions of the final software, and are the minimum requirements for a successful system.

2.2.1.1 Data Acquisition and Formatting

First and foremost, the software must acquire the data output from the satellite receiver and unpack the 16-bit words into ASCII digits. These digits must be logically grouped into lines and paragraphs, with one or more paragraphs per pass. The various two minute timing words must also be properly handled, although they should not be stored. The end of a pass will be signalled by either an end of pass control word, or by a receiver two minute mark in the absence of the former.

2.2.1.2 Error Detection and Recovery

There are three major types of errors which can occur in the input data, and each requires different actions to recover.

Data word sequence errors can be detected simply by checking the sequence code (the most significant four bits) of each input word of each line. If a word arrives out of sequence, then the entire paragraph should be rejected or ignored by the software. Sequence errors could be caused by noise, or by restarting the computer program midway through a pass, or by the receiver unlocking from a given pass in the middle of a paragraph, or by other exceptional circumstances.

Due to electromagnetic noise and interference, parts of satellite messages are often garbled and may show up as invalid BCD codes (larger than 1001). This is a common

occurence, and there is no requirement for this project to include error correction facilities for this, since the algorithms tend to be somewhat complex. The software should, however, be prepared to accept this data without complaint and to record it normally.

The first fourteen digits of each line consist of two doppler counts provided by the satellite receiver. These BCD digits should never be garbled as described above, unless due to equipment faults. Should invalid codes for these digits be encountered, the software should reject the entire paragraph in which they occur, in the same manner as described for sequence errors above.

2.2.1.3 Data Storage

The data should be stored on diskette, preferably between passes so as not to lose incoming data from the receiver. The storage format chosen should allow for easy access to the data by other programs.

2.2.1.4 Operator Review of Data

The operator of the receiver and computer equipment should be able to view the data on the computer console, so as to verify correct functioning of all hardware and software, and so that error rates can be viewed and decisions made concerning operation of the receiver.

2.2.1.5 Transfer of Data to IBM 3032

Software must be written to enable the data to be transferred to the IBM-3032 at the U.N.B. Computing Center for later analysis and processing.

The software must be capable of performing the logon procedure to obtain access to VSPC, either under operator supervision, or from logon commands entered by the operator and transmitted directly to VSPC. In view of the constantly changing logon procedures at U.N.B., perhaps the latter might be most practical.

The program should, once signed on, be able to transfer pass files to VSPC workspaces with minimal operator intervention, since at 300 or 1200 baud this can be time consuming. The program should either recognize on its own, or allow the operator to recover from, any exceptional situations which arise, such as unexpected responses from VSPC.

2.2.2 Optional Capabilities and/or Provision for Future Expansion

The following components may either, at the programmer's discretion, be incorporated into this project, or at least allowed for as future enhancements to the system.

2.2.2.1 Use of a Real Time Clock

If the hardware includes a real time clock, then time stamps could be tagged onto each paragraph and saved on diskette with the pass data.

2.2.2.2 Computer Feedback to Receiver

With a special board installed in the receiver, it is possible for commands to be issued to the receiver from the computer. One such command is of interest: requesting that the receiver unlock from the current pass and resume scanning the horizon for other approaching satellites. This would be accomplished by sending a 4-bit (parallel) message from the computer to the receiver. If the computer hardware allows it, this option may be useful in the future.

2.2.2.3 Data Compression

Either in real time, as is preferable, or later under operator supervision, data could be compressed from about 4600 digits per pass to about 1000 digits. This can be accomplished by performing majority voting procedures to correct errors in the satellite message field of each line. Since this information is repeated in each paragraph, much space could be saved by elimination of redundant copies of this information.

2.2.2.4 Operator Editing of Pass Files

If possible, the programmer should provide a facility for non real time operator review and editing of pass files on diskettes. Depending on how the data is stored, either a system provided editor, or a custom written program could allow this to be done.

2.2.2.5 Computing Receiver Positions

As a final option, procedures could be coded to allow the computer to calculate receiver positions from each pass, preferably in real time. To do this would require that the error processing described for data compression be done as well.

Chapter 3

SYSTEM HARDWARE DESCRIPTION

3.1 THE WINNING CANDIDATE: APPLE II PLUS

The hardware which was actually acquired is a system based on the Apple II Plus. This computer was chosen because of its versatile interfacing capabilities which make it capable of a broad range of services. The problems with limited diskette storage capacity have not yet been resolved, although Dr. Wells is currently examining several alternatives to the drives supplied by Apple Computer Inc.. Figure 4 below lists all of the hardware purchased to date for this system.

Bell & Howell Apple II Plus,
with Rear Power and Accessory Unit.
and 48K RAM installed at the factory.

Eastmoor Sales 16K Memory Expansion Board,
(Similar to Apple Language Card).

Apple Disk II Controller Card,
for two 16-sector drives.

Two Apple Disk II 16-Sector Floppy Diskette Drives.

Apple Dos 3.3 System Master Pack,
with reference manuals.

Apple UCSD PASCAL Language System,
with reference manuals.

Package of 10 Unformatted Diskettes,
(now in use with PASCAL).

Model 7424 Calendar/Clock Module,
from California Computer Systems.

Model 7720B Parallel Interface Card,
from California Computer Systems.

Apple 110/300 Baud Communications Interface Card,
with Telpong and Datamover software (not used),
and RS-232 connector cable

Electrohome 9" Black and White Video Monitor,
with (wrong) connector cable.

RF-Modulator for using color television as monitor.

Figure 4: List of Major Purchases

3.2 PHYSICAL CONNECTIONS

The following section describes hardware items modified or constructed by the author. Most of these are minor items, but all were necessary to complete the system.

3.2.1 Adapter for HP-1000 Connector on Receiver Cable

The most important item in this section is the construction of the physical coupling between the Apple and the satellite receiver. The cable from the receiver terminates in a female edge card connector intended for direct connection to an HP-1000 computer. To connect the Apple to this cable required determining which signals from the receiver were needed, and then wiring a connector cable to mate with the edge card connector on one end, and the 25-pin communications jack supplied with the parallel interface card on the other end. Examination of the connections on the edge card connector revealed that the voltage levels and timing were TTL compatible, using negative logic. Figure 5 shows the active pins on the edge card connector.

Pin functions for the PIA jack on the back of the Apple are detailed in the user's manual for that device. Figure 6 below shows the interconnection between these two connectors, accomplished with an adapter cable constructed by the author.

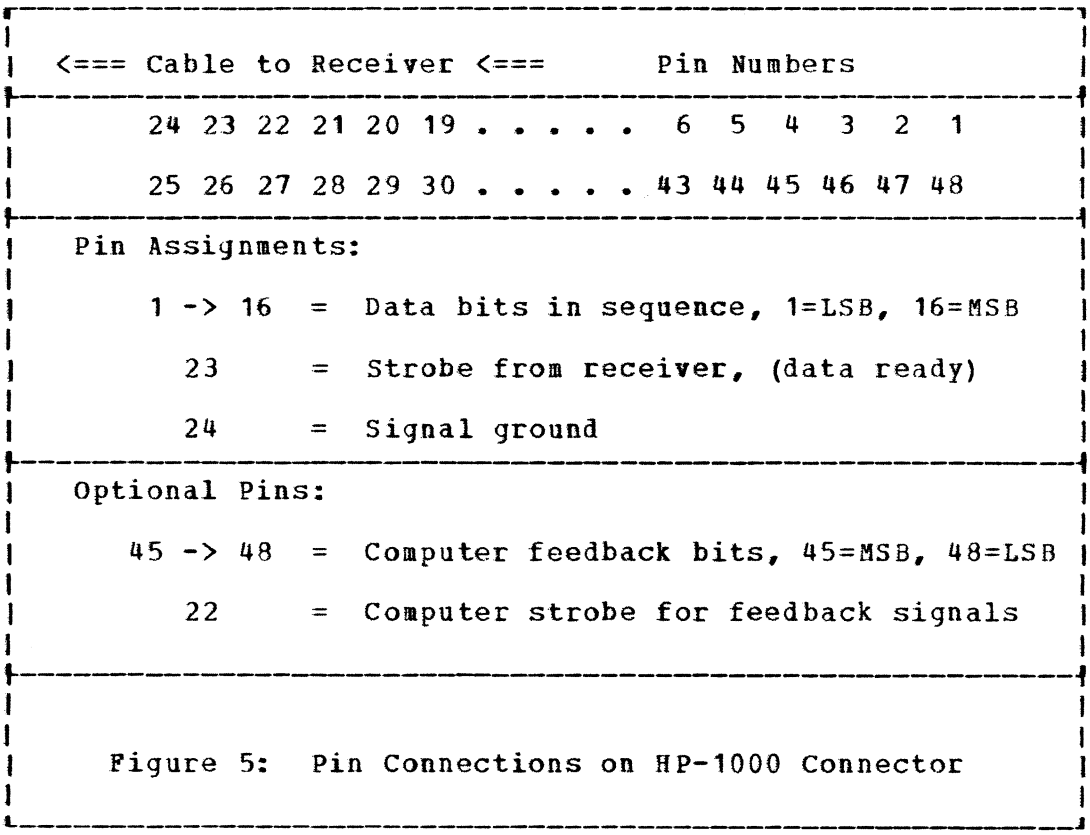


Figure 5: Pin Connections on HP-1000 Connector

PIA Pin Number on DB-25 Jack		Signal & Direction	HP-1000 Connector Pins on Edge Card Connector	
Pin-16	(PB7)	<--data--	Pin-16	(A15)
Pin-03	(PB6)	<--data--	Pin-15	(A14)
Pin-17	(PB5)	<--data--	Pin-14	(A13)
Pin-04	(PB4)	<--data--	Pin-13	(A12)
Pin-18	(PB3)	<--data--	Pin-12	(A11)
Pin-05	(PB2)	<--data--	Pin-11	(A10)
Pin-19	(PB1)	<--data--	Pin-10	(A9)
Pin-06	(PB0)	<--data--	Pin-09	(A8)
Pin-22	(PA7)	<--data--	Pin-08	(A7)
Pin-09	(PA6)	<--data--	Pin-07	(A6)
Pin-23	(PA5)	<--data--	Pin-06	(A5)
Pin-10	(PA4)	<--data--	Pin-05	(A4)
Pin-24	(PA3)	<--data--	Pin-04	(A3)
Pin-11	(PA2)	<--data--	Pin-03	(A2)
Pin-25	(PA1)	<--data--	Pin-02	(A1)
Pin-12	(PA0)	<--data--	Pin-01	(A0)
Pin-08	(CA1)	<-strobe-	Pin-23	(transfer)
Pin-07	(GND)	<-ground-	Pin-24	(signal ground)

Figure 6: Connections from Receiver Cable to Apple Computer

3.2.2 RS-232 Connector for Modem

Although the communications interface card came with an "RS-232" cable for connecting the Apple to a modem (or other device), this cable does not contain sufficient signal wires to be compatible with many modems, so a modified connector had to be placed on the modem end of the cable. This connector has pin 4 (RTS) wired to pin 6 (DSR) to provide the missing signal.

3.2.3 Cable Clamp for Diskette Drive Ribbon Cables

The disk controller card came with a cable clamp fitting for the back of the Apple, as did the communications interface card. However, the parallel interface card did not come with such a clamp. Therefore the two clamps on hand were used for mounting the DB-25 connectors for the two interface cards, and a special clamp was made for the diskette drive cables. Since the calendar/clock module was not an official Apple accessory, its dimensions were slightly too large to allow the original cable clamp to fit snugly beside it, so the new clamp was bent to allow the calendar/clock module more room.

3.2.4 Modification To PIA Ribbon Cable

With the parallel interface card in slot 1 and the communications card in slot 2, there was not quite enough room for the ribbon cable from the parallel interface to reach up between the two, so the cable was removed from its connector and re-inserted pointing upward. This modification does not affect the pin assignments on the DB-25 connector.

3.2.5 Chip Replacement On Disk Controller

Since the 16K memory expansion board was not purchased from Apple Computer, we also did not receive the new versions of the disk controller ROM's. Since the controller card was supposed to already have the new chips on it when we pur-

chased it, this should have made no difference. However, for some reason, our controller card came with the new version of one of these chips, and the old version of the other. The only problem noticed while using the old chip was that PASCAL would not boot properly using a single drive. Murray Linton, who also uses an Apple, stated that his diskette controllers had come with the proper chips, and that he had an extra set which had arrived with his Apple Language Card. Having no other use for them, Murray donated his spare chips to this project and we now have a properly upgraded controller card.

3.2.6 The "Red" Switch on the Language Card

Our non-standard language card also came with another feature, a red switch intended for use with an older version of the Apple II. This switch normally would protrude from the rear of the Apple through one of the cable slots. Unfortunately, the connector for the communications card now occupies that space, so the switch toggle had to be cut to a much shorter length than before, so that it would not be in the way. This switch should always be set in the "UP" position to allow proper booting of PASCAL.

3.2.7 Null Modem for Connecting a Printer

A short null modem cable was constructed to allow use of an RS-232 compatible printer/terminal with the Apple, connected through the communications interface. To use, simply plug the male end of the cable into the DB-25 connector on the back of the Apple, and then plug the printer (or other RS-232 device) into the other end of the cable.

3.2.8 Parallel Echo-Back Connector

On page 5-6 of the Owner's Manual for the parallel interface card is described how to construct a Parallel Echo-Back Connector for use in testing that device. As part of this project, such a connector has been constructed and used for testing purposes. To use, simply plug onto the DB-25 jack on the back of the Apple.

3.2.9 Computer Feedback to Receiver

Provisions have been made in the software to allow computer feedback to the satellite receiver. If this feature is desired, the physical connection is all that is currently lacking. Five wires must be run from the game control outputs on the side of the Apple to the edge card connector on the previously discussed adapter cable. The computer data output is available on annunciators zero to three. The strobe output, also on the game control pins, should be used to strobe data to the receiver. For more information, look

in the small pamphlet entitled "Owners and Operators Guide" for the Bell & Howell Microcomputer System.

3.2.10 Batteries For Calendar/Clock Module

The calendar/clock board is supposed to maintain time-keeping even when the Apple is powered off. To do this requires two small battery cells, type 675, which can be purchased from most camera shops. Two such cells have already been installed, and these should last for about a year. Be careful to note the corrections made by the author to the schematic for the calendar/clock module when inserting the batteries.

Chapter 4

SYSTEM SOFTWARE DESCRIPTION

4.1 THE RECEIVER PROGRAM

4.1.1 Running the RECEIVER Program

The RECEIVER program performs all data acquisition and storage activities. To run the receiver program, simply select the "X)ecute" option from the PASCAL system's main command menu. PASCAL will then prompt for a file name, which in this case is "SYSRUN:RCV.CODE", and the receiver program will be executed. This same procedure applies to all user-written PASCAL programs.

When the program begins execution, it first reads the screen format from the file "#4:RCV.SCREEN.TEXT" and copies it to the Apple screen. The user is then greeted with a formatted activity display and a message informing him how much free core is remaining for program expansion. While the user is reading this message, the RECEIVER program is busy reading its parameter file, "#4:RCV.PARAM.TEXT", to find out how to name pass files, so there was no need to code a delay loop to allow the user time to read this message. If either of these two files can not be opened on the diskette in drive #4, the boot drive, a message is displayed informing the user of this, and the program gracefully terminates itself.

Next, the RECEIVER program continues with its initialization process, interrupts are enabled, and the program waits for input from the satellite receiver. When a complete pass has been collected, the program disables interrupts, attempts to save the pass file on diskette, overwrites the parameter file with the updated value for the next pass number, and then re-enables interrupts and repeats the process of waiting for data again.

This cycle continues until the operator intervenes. The RECEIVER program itself supports four commands which may be entered by pressing the escape key followed by the key corresponding to the desired command. These commands are described in Figure 7, and are also detailed in the appendix entitled "Operating Instructions".

The source code for RECEIVER can be found on either of the "SYSTXT:" or "SYSBKP:" diskettes.

Command	Full-Name	Function
Q	Quit	- Sets a flag in the RECEIVER program which causes the program to terminate upon completion of the current pass. The pass will be saved as usual before the program exits.
S	Stay	- Negates the effect of a previously issued "Quit command". This command was provided to allow the operator to change his mind.
U	Unlock	- This command causes the RECEIVER program to unlock itself from collection of data for the current pass. Completed paragraphs for this pass are saved as usual on diskette, and the program then waits for new pass data from the receiver. If the computer feedback board in the satellite receiver is operational and connected to the Apple computer, then a command will be issued to the receiver requesting that it unlock reception of the current pass and resuming scanning for other satellites.
K	Kill	- This command terminates RECEIVER immediately and returns control to the PASCAL operating system. For a less forceful shutdown, the operator should enter the "Quit" command followed by the "Unlock" command, which will allow saving of the current pass.

Figure 7: Operator Escape Commands For The RECEIVER Program

4.1.2 RECEIVER Program Logic Explanations

The following subsections are intended to supplement the comments present in the RECEIVER.TEXT source file. Due to the size of that file, annotations within the program had to be kept to a minimum to allow the program to be edited and compiled as a single source file.

4.1.2.1 Message Synchronization

When the RECEIVER program begins waiting for a new pass, it simply waits until it receives a data word from the satellite receiver which has a sequence code of '0101', which is the code for the start of a new line. It then begins collecting data words to form lines, checking sequence codes as it goes. Should a timing word be encountered, or a sequence or data error occur, then the lines collected are discarded and the program begins searching for the '0101' code again. If twenty-five lines are collected without any intervening sequence errors, data errors, or timing words (see below), then these are stored in memory as a complete paragraph. The program repeats this collection process until either a receiver two minute timing word is received, an end of pass indicator word is received, or until it has collected eight full paragraphs of data. In the case of the latter, the "Unlock" command (previously discussed) is automatically performed by the program. These data paragraphs are then saved on diskette, and the whole process starts over again.

4.1.2.2 End of Pass and Timing Words

When a receiver two minute mark or an end of pass indicator is encountered, the program simply exits the routine for reading a pass, saves all completed paragraphs of data on diskette, resets its paragraph count to zero, and begins collecting paragraphs all over again. The routine which reads paragraphs adds one to the paragraph count every time it successfully collects twenty-five data lines without interruptions.

Satellite and test two minute timing words simply cause the program to exit the routine which reads paragraphs, thus preventing the paragraph count from being incremented. Therefore, the next invocation of that procedure will cause it to begin a new paragraph in place of the incomplete one.

4.1.2.3 The Screen Display

Most of the above logic can be viewed easily by watching the display created by the program on the Apple monitor while either a satellite pass is being received, or while TEST mode of the satellite receiver is being used to simulate a satellite pass.

The display is broken up into five active sections, plus a section which simply displays program identification.

1. **Program Nesting:** This section can be found in the lower left of the display. Each time a major procedure or function call is made, the routine called

displays its name here in the next empty line on the screen. When that routine completes processing, it erases its name from the screen prior to returning to the calling procedure, whose name is shown on the line above. This display is very useful for debugging purposes (should future program modifications be made), but has other uses as well. If the operator is ever in doubt as to whether the computer has "hung up" in an endless loop, or perhaps just died, he may quickly reassure himself that it is simply waiting for data by watching this display as he presses the spacebar. A quick flicker of "SCANKB" should immediately flash as that routine is called to process the operator's input, which in this case will be ignored. Other uses would be to test the program's response to exceptional conditions in the data: simply watch the program's activity upon receiving the data.

2. **Input Words:** This section is visible as a long column down the righthand side of the display. As data words are received from the receiver (via GETWORD routine) they are displayed in this column in the form of a rolling list. Special codes, such as satellite two minute timing words are not displayed "as is". Instead, the operator will see one of the codes listed in Figure 8 displayed, one extra column to the left to make it more noticeable. Note that

whenever an invalid sequence code is displayed, a "beep" is sounded to alert the operator as the program continues.

Code Which Is Displayed	Interpretation
R2MIN	Receiver two minute timing word
S2MIN	Satellite two minute timing word
T2MIN	TEST mode two minute timing word
ENDPS	End of pass indicator

Figure 8: Special Input Words

3. **Input Lines:** This section occupies the lower middle portion of the screen, and is used for displaying lines of input data as they are formatted by the program. Lines are displayed as a rolling list, and the entire area is cleared at the beginning of each new paragraph.
4. **Program Status and Paragraph/Line Counts:** This area is located in the upper left corner of the display, and is used for two purposes. The first is to show what the program is currently doing. This is called the "STATUS", and will always show one of the values listed in Figure 9 below.

Status	Meaning
SETUP	The program is busy setting up the screen display and initializing variables. This setting is visible only for a brief second after the program begins execution.
DISKIO	Interrupts have been disabled while the program is performing diskette operations such as saving a pass file, or reading or updating the parameter file.
ACTIVE	This is the "normal" mode of operation, meaning that the program is busy processing input data from the receiver.
WAIT	This status value indicates that the program has processed all data in the input queue and is currently waiting for more input words from the receiver. This is almost always displayed when no satellite pass is being monitored, except when two minute timing marks are sent from the receiver.

Figure 9: Significance of STATUS Display

Also visible, underneath the STATUS, are running counts showing which paragraph, and which line within the paragraph, is currently being formatted by the RECEIVER program.

5. **Message Area:** The fifth active display area is the message area, located in the middle of the screen. There is room for three lines to be displayed is this

area, which is used to communicate messages to the operator. When the program first begins execution, a brief message is shown informing the user as to how much free storage remains on the PASCAL system data heap. Then this message is erased and replaced by a list of available user commands. Whenever a new paragraph is started, a timestamp, identical to that written to the pass files, is placed on the third line of this area. Also, the name of the last pass file written to diskette is displayed on the top line of this area. This message will not be displayed if no pass files have been saved during the current execution of the program. When a file handling error occurs, the program will either display an error message in this area, or use the area to prompt the user for a further course of action. Thus, this area is very important to the operator.

4.1.3 SATLITE Assembly Language Procedures

A group of four assembly language routines handle all input from the satellite receiver using a circular queue structure as an input buffer. This queue can hold up to 256 16-bit data words at once before overflow will occur. This allows the PASCAL RECEIVER program to fall up to two minutes behind in retrieving input data before problems occur. The main reason for this particular queue size is to take advantage

of the automatic "wrap-around" which occurs when an 8-bit index register is incremented past "FF", thus allowing the circular queue to be programmed as if it were simply a sequential area of storage. For more details on the operation of this queue, see the program descriptions in the following subsections.

The source code for these programs can be located in the file "SATLITE.TEXT" on either of the "SYSTXT:" or "SYSBKP:" diskettes.

4.1.3.1 INITPIA

This routine initializes the parallel interface adapter (PIA) to cause a maskable interrupt whenever a 16-bit data word is received. The IRQ/BRK vector at memory locations FFFE-FFFF is set to the address of the interrupt handling routine, IRQHANDL, and the previous contents of this vector are saved for later restoration. The queue pointers are initialized to indicate an empty queue, and interrupts are enabled just before this routine returns to the PASCAL calling program.

4.1.3.2 IRQHANDL

This routine is actually part of the GETWORD procedure for assembly and linkage purposes, but the two routines are logically separate. This was done to avoid problems with the linkage editor since IRQHANDL is never invoked directly by any other routines.

IRQHANDL is the routine which handles interrupts from the PIA. Upon an interrupt occuring, this routine first checks to see if it was caused by a "BRK" instruction. If so, then control is passed to the routine at the address specified by the original IRQ/BRK vector. Otherwise, the routine checks to see if the interrupt was caused by the PIA. If not, then control is again passed to the originally specified interrupt handler routine (PASCAL sets this to cause a system reboot by default). Finally, if the interrupt was caused by the PIA, then the 16-bit input is saved in a queue for later retrieval by GETWORD, the queue pointer is incremented, and the routine issues an "RTI" (ReTurn from Interrupt) instruction. All registers modified by this routine are saved on entry, and restored again before the "RTI".

If the queue is full and IRQHANDL has been invoked to add more data to the queue, it will instead replace the last word on the queue with the hexadecimal value "1111". When the RECEIVER program later obtains this value from the queue, it will respond to the invalid first digit by treating it as if a data sequence error had occured, thus effectively causing the current paragraph of input data to be ignored.

4.1.3.3 GETWORD

This routine is called by the RECEIVER program whenever another word of input data is needed. It expects as its

only parameter the address of a PASCAL "STRING" variable (special "Type" in Apple PASCAL) with a declared length of at least four characters. STRING variables are stored with a "length" byte preceeding the data bytes, which is used to store the current length of the STRING. If the input queue is empty, the GETWORD routine will simply set the length of the STRING to zero and return. Otherwise, the length will be set to four, and the next 16-bit word from the input queue will be unpacked into ASCII characters and assigned to the data portion of the STRING.

Unpacking of the 16-bit words is done as follows: The most significant four bits of the word are converted to an ASCII hexadecimal digit in the range 0..9 or A..F. The next three groups of 4-bits are each "OR-ed" with the hexadecimal value "30" to convert them into an ASCII digit in the range 0..9. Should invalid data be present in these fields, they will thus be converted to various punctuation symbols which will look like "garbage".

Interrupts remain enabled throughout execution of GETWORD, and the queue pointer is not updated until the data for that queue entry has been saved elsewhere beforehand. This is done to avoid conflicts with IRQHANDL.

4.1.3.4 RESETIRQ

This routine disables maskable interrupts and resets the PIA to prevent it from causing further interrupts. The original

value for the IRQ/BRK vector is retrieved from its save location and is restored to addresses FFFE-FFFF. The routine then returns to the PASCAL calling program, leaving interrupts disabled.

4.1.4 Files Used by the RECEIVER Program

4.1.4.1 The Screen File

When the RECEIVER program begins execution, it attempts to copy a file from drive #4 named "#4:RCV.SCREEN.TEXT" to the display screen. This file is known as the screen file, and is used to hold the initial data to be displayed on the screen when the program begins execution. The contents of this file may be changed at any time by the user, but moving around display areas will also require modifications to the RECEIVER program to avoid a messy display. If for any reason this file cannot be opened, the receiver program will display a message informing the operator of this, and then gracefully terminate execution.

4.1.4.2 The Parameter File

After initializing the screen, the RECEIVER program attempts to open and read a file from drive #4 named "#4:RCV.PARAM.TEXT". If for any reason this file cannot be opened, a message is displayed and the program terminates gracefully.

```
STATUS:  SETUP  | SATELLITE PASS  |INPUT
PARA/LINE= 00/00| MONITOR PROGRAM |WORDS
                +-----------------+-----
                |
                |
                |
                +-------+---------------+---------+
                |PROGRAM| DOPPLER-COUNT |SATELLITE|
                |NESTING|400-MHZ:150-MHZ| MESSAGE |
                +-------+---------------+---------+
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |
                        |               |$

(* THIS FILE HOLDS THE INITIAL SCREEN
   DISPLAY FOR THE RECEIVER PROGRAM.
   ALL CHARACTERS UP TO THE '$' ARE
   COPIED TO THE CONSOLE SCREEN UPON
   STARTUP WHEN THE PROGRAM EXECUTES *)
```

Figure 10: Contents of The RCV.SCREEN File

This file expected to contain three lines of information, which are to be used by the RECEIVER program in forming distinct, sequentially numbered names for its pass files. The first line should specify the characters to be used in forming the first (root) part of the file names. This should not contain a device specification. The second line should

specify two integer values, separated by one or more spaces, to be used for numbering pass files. The first value represents the next number to be used in creating a pass file name, and the second value is an increment which is added to the first value after each new pass file is created. This file is rewritten to drive #4 each time the pass number is incremented.

The third line of the file should contain characters to be used as the trailing part of the pass file names. Usually this is set to ".TEXT", but if the ability to edit pass files is not required, considerable storage can be saved by specifying a different file type such as ".DATA". The PASCAL overhead for each pass file using ".TEXT" varies depending on file size, but usually about 1500 bytes are wasted in overhead.

Note that since a PASCAL "REWRITE" statement is used to open the file on output, this file need only be present when the program starts execution, since it is "READ" only once by the RECEIVER program. This allows the operator to replace the boot diskette in drive #4 with an empty data diskette after program startup, thus allowing more space for pass files.

Figure 11 shows typical contents of the parameter file, along with some sample pass file names created using these parameters. Note that all three lines should be left justified when entered into the actual file using the PASCAL editor.

Contents of the RCV.PARAM File:

```
PASS
100 10
.TEXT
```

Consective Pass File Names Generated:

```
PASS100.TEXT
PASS110.TEXT
PASS120.TEXT
```

Figure 11: Construction of Pass File Names

4.1.4.3 Format of Pass Files

Pass files are written as a sequence of one or more paragraphs, where a paragraph record consists of a time stamp line followed by twenty-five lines of data. The time stamp is written exactly as generated by the READTIME procedure, in the form YY/MM/DD-d HH:MM:SS, where "YY/MM/DD" is the current date in metric format, "-d" is the day of the week (0=Sunday, 1=Monday, ...), and "HH:MM:SS" is the time in hours, minutes, and seconds. Each of the twenty-five data lines is in the format "1234567 1234567 123456789", where the first group of seven digits is the 400 Mhz doppler count, the second set is the 150 Mhz doppler count, and the third set is the satellite message. Each line has a carriage return character appended to it, to allow editing

using the PASCAL system editor. Editing can only be done if the files are named as ".TEXT" files. Figure 12 shows a partial listing of a pass file generated using the RECEIVER program.

```
                82/04/17-6 18:48:43
                9999999 9999999 060922274
                0122455 0122468 071071955
                0245187 0245215 081181578
                0368201 0368241 091241156
                0491503 0491558 401240702
                0615100 0615171 411190276
                0738999 0739084 521100144
                0863205 0863306 530960506
                0987726 0987840 087688720
                1112569 1112699 841433630
                1237739 1237882 816071120
                1363245 1363403 800204640
                1489093 1489265 800170230
                1615290 1615477 807393060
                1741843 1742042 814548120
                1868760 1868975 800000330
                1996047 1996274 900007900
                2123712 2123951 806461800
                2251762 2252013 800302000
                2380205 2380467 843201070
                2509048 2509319 809999990
                2638298 2638581 800590000
                2767962 2768254 000000000
                2898048 2898350 000000000
                3028564 3028875 000000000
                82/04/17-6 18:50:43
                3301600 3301928 071071955
                0131883 0131891 081181578
              (Remainder of file not shown)
```

Figure 12: Pass File Produced by RECEIVER Program

4.2 THE TALK PROGRAM

Comments in the TALK.TEXT source file are reasonably detailed, so only a basic description is given here. The TALK program was written to handle communications with VSPC (or other systems). To run TALK, simply execute the "SYSRUN:TALK.CODE" file.

Upon startup, the TALK program displays a "menu" of command options from which the operator can specify what type of TALKing is to be done. These options are described in more detail below. Note that the file transfer options both require that VSPC TAPE mode be entered beforehand by the user, using dumb terminal mode. Both of these routines will of course attempt to issue the command on their own, but if problems arise, the user will have to hit control "C" to terminate the module and then enter TAPE mode himself. TAPE mode causes the last line of output sent by VSPC to be terminated with a special "DC1" control character, which is used by this program to determine when VSPC is ready for more input. The dumb terminal option does not depend on this feature, but the transfer options do.

4.2.1 Dumb Terminal Mode

Selection of this mode of operation effectively turns the Apple keyboard and display into a "dumb" ASCII terminal, operating in half-duplex at 300 baud. This allows the user to communicate with a large variety of remote devices,

including other computer systems. Four keys on the Apple keyboard supply helpful functions to the operator.

4.2.1.1 The Escape Key

When pressed, this key generates a "beep" and transmits a "BREAK" or "ATTN" signal to the remote device.

4.2.1.2 The Left Arrow

This key functions as a VSPC character delete key. When pressed, it causes a backspace-linefeed combination to be transmitted to the remote device, thus acting as a RUBOUT key. The ASCII "DEL" character was originally transmitted here, but VSPC does not seem to recognize it, so the backspace-linefeed combination was used instead.

4.2.1.3 The Right Arrow

This key causes an ASCII horizontal tab character to be transmitted. On the Apple screen, it appears as a space character.

4.2.1.4 Hitting Control "C"

If a "C" is typed while holding down the "CTRL" key, the resulting character code is not transmitted. Instead, the TALK program will return the user to its original command menu.

4.2.2 Passfile Transfer Mode

This option allows the user to specify a range of pass files to be transferred to VSPC workspaces for later use. Once the program has completed prompting the user for the pass file range, it proceeds to transfer the pass files without need for further intervention. Note that if any keys are pressed on the Apple keyboard during the transfer, the characters typed will also be sent to VSPC. All of the special keys described for the dumb terminal mode can also be used here. Therefore, to stop data transfer, hit control "C".

4.2.3 Text File Transfer Mode

This option allows the user to specify a single file of type ".TEXT" to be transferred to a VSPC workspace. Once again, any characters entered at the keyboard during the transfer will also be transmitted to VSPC, except for control "C", which will cause a return to the main TALK program menu.

4.3 OTHER PROGRAMS

This section contains brief descriptions of programs which were written as an indirect part of this project. Some of these are general subroutines which are used by the RECEIVER and TALK programs, while others, such as SETTIME, are utility programs.

4.3.1 The PEEKPOKE Intrinsic Unit

The PASCAL language, on its own, does not support absolute addressing of memory locations, so two routines, PEEK and POKE, have been written to allow this capability. The PEEK function and the POKE procedure work the same as their BASIC counterparts. These routines are grouped into an INTRINSIC UNIT, called PEEKPOKE, which has been installed in the SYSTEM.LIBRARY file for general use. The TALK program uses these routines to communicate directly with the 6850 ACIA on the communications interface card.

These procedures are modified versions of routines published as part of an aricle in BYTE Magazine (see bibliography).

4.3.2 The READTIME Procedure

This program is an assembly language procedure which can be called from PASCAL. It expects as a parameter, the address of a nineteen byte area of storage in which to place a packed character string representing the current date and time. READTIME reads this information directly from the calendar/clock module, and formats the time stamp into ASCII characters with punctuation included. The returned result will be in the form YY/MM/DD-d HH:MM:SS, where "YY/MM/DD" is the current date in metric format, "-d" is the day of the week (0=Sunday, 1=Monday, ...), and "HH:MM:SS" is the current time of day in standard format. This program has been

assembled and placed into the SYSTEM.LIBRARY file for general use.

4.3.3 The STARTUP Program

Apple PASCAL allows for creation of a "turn-key" system, where a program is automatically run when the system is powered on. To do this, the programmer merely has to name his program code file as "SYSTEM.STARTUP" and copy it onto the boot diskette. Such a program has been written for this system, although it does not run either of the RECEIVER or TALK programs. Instead, it is used to sound a greeting tune and display the current date and time for the operator. There are two sources for a "date" under PASCAL: the boot diskette, and the calendar/clock module. Normally, the user is expected to update the date stored on the boot diskette every day, using the D(ate command of the system F(iler program. This is the date which is recorded in diskette directories whenever files are created. Unfortunately, PASCAL relies totally upon the user to update this daily.

The SYSTEM.STARTUP program written for this project is automatically run whenever the system is initialized. The first function it performs upon executing is to compare the date from the boot diskette with the date from the calendar/clock module. If they match, then the greeting tune is played and the date and time displayed. If they do not match, then the date on the boot disk is updated by this

program, and the user is requested to issue an I(nitialize command to PASCAL so that the operating system will read the new date from the disk by doing a warm start. During this re-initialization, the SYSTEM.STARTUP program is again executed, causing the greeting tune to be played and the new date displayed.

4.3.4 The SETTIME Program

This program is written in Apple BASIC, and was copied directly from the Owner's Manual for the calendar/clock module. It is intended to be used to set the date and time on the calendar/clock module from scratch. This program can be found on the work diskette in the DOS 3.3 System Master Pack.

Chapter 5

CONCLUSIONS

The system proposed and implemented in this report satisfies all of the minimal requirements for a reasonable alternative to the present paper tape system for recording and transfering data from the satellite receiver. In addition, many of the optional requirements, such as use of a real time clock, and editing of pass files, have also been implemented. The Apple computer with PASCAL is a very powerful programming tool, and is well suited to such applications; the lack of a good mass storage diskette system being its only drawback.

Many options for future expansion still remain, such as data compression and more complex error correcting routines, perhaps enough for another CS 4993 project?

As can be seen in the final progress and status report, many more manhours than previously estimated have gone into the preparation of this system. Hopefully, it will soon be put to the test and prove itself worthwhile.

Appendix I

OPERATING INSTRUCTIONS

This appendix is intended as a User's Guide for people who are familiar with TRANSIT and who have had previous experience with modern micro-computers.

I.1 HARDWARE CONNECTIONS

I.1.1 Connection to the Satellite Receiver

To prepare the Apple for use with the satellite receiver, simply connect the power cables to 120VAC and plug the receiver adaptor cable into the PIA connector on the back of the Apple. Connect the other end of this cable to the HP-1000 connector on the receiver cable. The system should now be ready for booting.

I.1.2 Connection of an Acoustic Coupler or Modem

To prepare the Apple for use with a coupler or modem, simply connect the power cables to 120VAC and plug the end of the black RS-232 cable labelled "APPLE" into the RS-232 jack (also labelled) at the back of the Apple computer. The other end of this cable, labelled "MODEM" should be connected to the jack on the coupler or modem. For VSPC, the coupler or modem should be set to FULL-DUPLEX, even though the Apple TALK program behaves as a half-duplex terminal with local-echo.

I.1.3 Connection to Other RS-232 Devices

To use the Apple with other RS-232 compatible DTE equipment, the Apple's power cables should be connected to 120VAC, and the "NULL MODEM" cable should be plugged into the RS-232 jack at the rear of the Apple computer. The external device should then be connected to the female jack on this cable,

using the cable supplied with the external device. If the device is equipped with a jack instead of a cable, the Apple's black modem cable can be used to connect the device, with the end which is labelled "APPLE" connected to the null modem, and the other end connected to the jack on the external device.

I.2 GENERAL CONSIDERATIONS

I.2.1 System Diskettes

Apple PASCAL uses four diskettes to hold its distributed operating system components. These are labelled APPLE1, APPLE2, APPLE3, and APPLE0. For a two drive system, APPLE1 is normally placed in drive #4, and APPLE2 in drive #5. More information on these diskettes can be obtained by consulting the PASCAL reference manuals.

The system described in this report also uses four diskettes, although the entire system could easily fit on a single diskette. Two of these diskettes are merely copies of APPLE1 and APPLE2, with a few extra program files present. The other two are used primarily for backup purposes.

I.2.1.1 The SYSRUN Diskette

This diskette is a copy of the APPLE1 diskette, along with the extra files required to run the RECEIVER and TALK programs, as well as a copy of the SYSTEM.STARUP program for setting and displaying the system date. This diskette is

referred to as the "boot" diskette, and is normally placed in drive #4.

I.2.1.2 The SYSTXT Diskette

This diskette is actually a copy of APPLE2, with the source files for the RECEIVER and TALK programs present. This diskette contains all PASCAL system files required for compiling and assembling these two programs and their subroutines. This diskette is normally placed in drive #5.

I.2.1.3 The SYSLIB Diskette

This diskette contains the source files for the READTIME, PEEKPOKE, and STARTUP programs, as well as an up to date backup copy of the SYSTEM.LIBRARY file from SYSRUN.

I.2.1.4 The SYSBKP Diskette

This diskette is used for backup purposes, and contains copies of the source files for the RECEIVER, SATLITE, TALK, PEEKPOKE, READTIME, and STARTUP programs. This disk should only be used when making or retreiving backups of these programs with the PASCAL F(iler command.

I.2.2 Booting The System

To boot the PASCAL operating system for use with the programs described in this project, place the SYSRUN diskette in drive #4, place the SYSTXT or a formatted SCRATCH disk-

ette (for data files) in drive #5, and then power on the Apple. One of two events should normally occur:

1. You will eventually be greeted with a musical tune and a message display, showing the current date and time according to the Apple, or
2. You will be greeted with a short "beep beep beep" and be requested to type "I" to re-initialize the system date for the day. After typing this command, you should then be greeted as described for event 1. above.

I.2.3 Special PASCAL System Keys

The PASCAL reference manuals describe several special keys which may be pressed to allow limited control over the execution of most programs. The keys which may be of interest with respect to this program are listed below.

1. **Control "S"** - Pauses execution of a program the next time it attempts I/O. The program remains halted until this key combination is pressed again. This is useful for halting the display of the RECEIVER program while monitoring pass data. No incoming data will be lost as long as the program is allowed to continue within two minutes from when it was paused.
2. **Control "@"** - Abnormally terminates execution of a program the next time it attempts I/O. The program is halted, an obscure system error message is

printed, and the system waits for the user to press reset.

3. **Control "RESET"** - Causes a hardware reset cycle. If pressed once, the PASCAL system will reboot itself from the diskette in drive #4. If pressed twice quickly in succession, the computer will be placed under control of the Applesoft Floating Point BASIC monitor. NEVER NEVER NEVER press RESET while either of the red lights on the front of the diskette drives are "on". Doing so will probably cause destruction of data on the diskette in that drive, although no physical damage to the computer will occur.
4. **Control "A"** - Allows the user to toggle the "80 column" display from showing the left-most 40 columns to showing the rightmost 40 columns, or vica-versa.
5. **Control "Z"** - Causes the display to scroll left or right with cursor movements when inputing text or other data.
6. **Control "F"** - This command should never be used with the RECEIVER and TALK programs.

For more information, consult the appropriate PASCAL reference manuals.

I.3 USING THE RECEIVER PROGRAM

I.3.1 Functions of RECEIVER

This program is intended for performing data acquisition from the satellite receiver. Each "pass" of satellite data received is stored in a pass file as the program executes. Monitoring of output from the receiver is possible with the dynamic screen display maintained by this program.

I.3.2 Running the RECEIVER Program

To run this program, the SYSRUN diskette should be present in drive #4, and the user should be in PASCAL command mode. After entering the "X" command to execute a user program, the system will prompt for a file name. To execute the RECEIVER program, the user should reply with "#4:RCV", and the program will begin execution.

I.3.2.1 Files Required at Initialization

Running the RECEIVER program requires that the SYSTEM.LIBRARY, RCV.CODE, RCV.SCREEN.TEXT, and RCV.PARAM.TEXT files all be present on the boot diskette in drive #4. These files are supplied on the SYSRUN diskette.

I.3.2.2 Files Required After Initialization

Once the file has completed initialization, none of the above files need be present. This allows the boot diskette to be removed from drive #4, and a data diskette to be

inserted in its place to allow more room for output pass files from the RECEIVER program. Be careful never to remove a diskette from a diskette drive while the red light is on, or else you may destroy the contents of one or more files on that diskette, or maybe all of them.

I.3.2.3 Creation of Pass Files

Data from the satellite receiver is formatted into pass files, which are saved on diskette for later editing and transfer. These files are named according to information in the RCV.PARAM.TEXT file, which is described elsewhere in this report. Normally, these pass files are saved on the diskette in drive #5. If there is no room on drive #5, or if there is no diskette in that drive, or if there is indeed no drive #5, then the program will search for another drive on which to save its pass files. The drives searched, in order of priority, are drives 5,11,12,9,10, and lastly, drive 4. Once a drive has been found which has room for more data, all subsequent pass data will be saved on that diskette until one of the previously mentioned problems occurs. The exception to this rule is drive #4. The receiver will always search for another drive before saving files on the diskette in this drive.

I.3.3 Operator Control of RECEIVER

The RECEIVER program allows the user to enter a limited set of four commands. To use one of these commands, the "ESCape" key must first be pressed, followed by the key designated for a particular command.

I.3.3.1 The "Q" Command

This command requests that the program terminate itself at the end of the current pass, after first saving the pass data, if any. Should this command be entered when no pass is being monitored, the program will wait until the next receiver two minute timing word is received before it quits.

I.3.3.2 The "S" Command

This command, which stands for "Stay", requests that the program ignore a previously issued "Q" (Quit) command.

I.3.3.3 The "K" Command

This command requests that the program terminate immediately, without saving data for the current pass, if any. This is intended as a quick way to exit the program. If a fast exit is desired without loss of data, the "Q" command should be issued followed by the "U" command instead of using this command.

I.3.3.4 The "U" Command

This command requests the program to "unlock" itself from collecting data for the current pass. If any complete paragraphs have been collected prior to the issuing this command, they will be saved as a normal pass file on diskette. The program will then return to collecting data again, unless a "Q" command was also issued prior to the "U" command, in which case the program will terminate itself.

I.4 USING THE TALK PROGRAM

I.4.1 Functions of TALK

The TALK program is intended to be used for communications with VSPC for the purpose of transferring pass data to the IBM 3032 at the U.N.B. Computing Center. The program has three modes of operation, which are discussed later in this section.

I.4.2 Running the TALK Program

To run this program, the SYSRUN diskette should be present in drive #4, and the user should be in PASCAL command mode. After entering the "X" command to execute a user program, the system will prompt for a file name. To execute the TALK program, the user should reply with "#4:TALK", and the program will begin execution.

I.4.2.1 Files Required at Initialization

Running the TALK program requires that the SYSTEM.LIBRARY, and TALK.CODE files both be present on the boot diskette in drive #4. These files are supplied on the SYSRUN diskette.

I.4.2.2 Files Required After Initialization

There only files which are required after starting execution of this program are those files, if any, which are to be transferred to VSPC using the transfer options of this program.

I.4.3 Main Menu Options

Upon starting execution, the TALK program will greet the user with a short command menu and a "beep" prompting the user to select an option from the menu. The options currently supported are described later, and all of these options allow the following keys to be used for non-standard purposes:

1. **Escape Key** - This key serves as a "BREAK" or "ATTN" key.
2. **Left Arrow** - This key generates a VSPC "RUBOUT" sequence of a backspace followed by a linefeed. This effectively will "delete" the last character typed on the current line.
3. **Right Arrow** - This key generates tab characters, the same as a "TAB" key on most standard terminals would.

The tabs will show up as a single space on the Apple display screen.

4. **Control "C" -** This code is obtained by typing a "C" while holding down the "CTRL" key. It causes an immediate return to the TALK program's main menu, and can be used to terminate file transfers prematurely.

I.4.3.1 Dumb Terminal Mode

This option allows the Apple keyboard and screen to be used as a half-duplex asynchronous ASCII terminal. This allows the user full control, desired or not, over the VSPC logon procedure, since it is up to the user to enter the commands to achieve signon. To exit from this mode, type a Control "C" character as described above.

I.4.3.2 Pass File Transfer Mode

This mode allows the user to transfer pass files created by the RECEIVER program to VSPC workspaces. Before this command is entered, the user should first sign on to VSPC using dumb terminal mode, and then issue the VSPC command "TAPE".

To transfer pass data to VSPC, the user must first specify which files are to be sent. The program prompts for a rootname, which is the full name of the diskette and file(s) to be transfered, less the file number embedded within the name. The program next prompts for the starting value for a range of these numbers, and then prompts for an ending

value. Finally, it requests that the user enter an increment to be added to the first value each time through the transfer process, to generate the next pass file name in sequence. If a given pass file cannot be opened for any reason, a message is displayed and the program proceeds on to the next file in sequence, until all have been attempted, or until the user hits Control "C" to return to the main menu. The following is an example of a rootname for the file PASS123.TEXT on the diskette in drive #5:

#5:PASS.TEXT

I.4.3.3 Text File Transfer Mode

This mode is similar to that described above for the transfer of pass files, except that it will work for files which may contain any type of text information. This mode must be used for transferring files which have been edited using the PASCAL system editor, including pass files.

The program first prompts the user for a file name, which should be entered in full, including diskette name, and then asks for a VSPC workspace name under which to save the text. Once the user has entered these two items, the program proceeds on its merry way, transferring text until all has been sent and saved, or until the user hits Control "C" to return to the main menu. The following is an example of a text file name: ' SYSTXT:SATLITE.TEXT '. The .TEXT may be omitted if desired.

I.4.3.4 Quitting the TALK Program

This option allows the user to exit from the TALK program.

Appendix II

PROGRAM LISTINGS

This appendix contains listings of programs written as part of this project. The only program discussed in the text which is not included is the SETTIME program, which was not written by the author. A listing of that program can be found in the Owner's manual for the calendar/clock module, from where it was originally obtained.

II.1 RECEIVER PROGRAM LISTING

```
(*$S+*) (* TURN LEVEL-1 COMPILER SWAPPING ON FOR LARGE PROGRAM *)
(****************************************
 * DUE TO THE PHYSICAL SIZE OF THIS     *
 * PROGRAM, COMMENTS MAY SEEM TO BE A   *
 * BIT SCARCE IN SOME SECTIONS.  THIS   *
 * WAS DELIBERATELY DONE TO ALLOW THE   *
 * PASCAL EDITOR TO RETAIN ITS LIMITED  *
 * CAPACITY TO EDIT THIS PROGRAM.       *
 * OTHERWISE, THIS PROGRAM WOULD BE TOO*
 * LARGE FOR THE EDITOR TO HANDLE IN A  *
 * SINGLE TEXT FILE.  PLEASE CONSULT    *
 * THE WRITTEN DOCUMENTATION TO CLARIFY*
 * ANY MAJOR UNCLEAR DETAILS.           *
 ****************************************)
PROGRAM RECEIVER;
USES  APPLESTUFF, PEEKPOKE;

CONST MAXPARA =  8; (* THIS LINE SPECIFIES # OF PARAGRAPHS/PASS *)
      MAXLINE = 25; (* THIS LINE SPECIFIES # OF LINES/PARAGRAPH *)

      (* THE FOLLOWING CONSTANTS ARE USED FOR *)
      (* POSITIONING ITEMS ON THE SCREEN, AND *)
      (* MOST CAN BE SAFELY ALTERED TO MODIFY *)
      (* THE SCREEN FORMAT.                   *)
      XMODE=7;  YMODE=0;
      XPNUM=11; YPNUM=1;
      XLNUM=14; YLNUM=1;
      XMSG =1;  YMSG =3;
      XLINE=9 ; YLINEMIN=10;YLINEMAX=23;
      XPROC=0;  YPROCMIN=10;YPROCMAX=23;
      XWORD=35; YWORDMIN=3; YWORDMAX=23;

TYPE FTNTYPE    = (SHOW,ERASE,CLEAR);
     DATAWORD   = STRING[5];
     DATALINE   = PACKED ARRAY[1..26] OF CHAR;
     TIMESTAMP  = PACKED ARRAY[1..20] OF CHAR;
     PARARECORD = RECORD
        PASSTIME:TIMESTAMP;
        PASSLINE:ARRAY[1..MAXLINE] OF DATALINE
        END;

VAR PROCNAMES:ARRAY[YPROCMIN..YPROCMAX] OF STRING[8];
    PASSPARA :ARRAY[1..MAXPARA] OF PARARECORD;
    SEQCODES :PACKED ARRAY[1..9]   OF CHAR;
    BUFFER   :PACKED ARRAY[0..511] OF CHAR;
    INPUTWORD       :DATAWORD;
    INPUTLINE       :DATALINE;
    PARACOMPLETED   :BOOLEAN;
```

```
    MEMUNUSED       :STRING[5];
    SCRPROC,SCRLINE,SCRWORD,PARACNT     :INTEGER;
    CLEARLINES,QUITREQUESTED,ESCPRESSED :BOOLEAN;

    PARAMFILE               :TEXT;
    PASSFILE                :FILE OF PARARECORD;

    PFNUMBER,PFINCREMENT    :INTEGER;
    PFDEVICE                :STRING[7];
    PFROOTNAME,PFEXTENSION  :STRING[14];
    PFNAME                  :STRING[26];


PROCEDURE INITPIA;                  EXTERNAL;
PROCEDURE GETWORD (VAR STRING4);    EXTERNAL;
PROCEDURE RESETIRQ;                 EXTERNAL;
PROCEDURE READTIME(VAR PKCHAR19);   EXTERNAL;


PROCEDURE SHOWMODE(MODE:STRING);
(* DISPLAYS STATUS: 'ACTIVE','WAIT','DISKIO' *)
BEGIN (* SHOWMODE *)
   GOTOXY(XMODE,YMODE);
   WRITE(MODE:7)
END; (* SHOWMODE *)


PROCEDURE SHOWMSG(MSGNUM:INTEGER;MESSAGE:STRING);
(* USED TO DISPLAY MOST MESSAGES IN 3-LINE MESSAGE AREA *)
(* MESSAGES ARE CENTERED IN THE 33-CHAR DISPLAY AREAS.  *)
VAR FILLER:INTEGER;
BEGIN (* SHOWMSG *)
   GOTOXY(XMSG,YMSG+MSGNUM);
   FILLER:=(33-LENGTH(MESSAGE)) DIV 2;
   IF FILLER<0 THEN FILLER:=0;
   WRITE(MESSAGE:(33-FILLER),'':FILLER);
END; (* SHOWMSG *)


PROCEDURE SHOWLINE(VALUE:DATALINE;FTN:FTNTYPE);
(* USED TO DISPLAY FORMATTED SATELLITE DATA LINES ON SCREEN *)
BEGIN (* SHOWLINE *)
   IF FTN=SHOW THEN
      BEGIN
         GOTOXY(XLINE,SCRLINE);
         WRITE(VALUE:25);
         IF SCRLINE=YLINEMAX THEN
            SCRLINE:=YLINEMIN
         ELSE
            BEGIN
               SCRLINE:=SCRLINE+1;
               GOTOXY(XLINE,SCRLINE);
               WRITE(' ':25)
            END;
```

```
         CLEARLINES:=FALSE
      END
   ELSE IF (FTN=CLEAR) AND (NOT CLEARLINES) THEN
      BEGIN
         FOR SCRLINE:=YLINEMAX DOWNTO YLINEMIN DO
            BEGIN
               GOTOXY(XLINE,SCRLINE);
               WRITE(' ':25)
            END;
         SCRLINE:=YLINEMIN;
         CLEARLINES:=TRUE
      END
END; (* SHOWLINE *)


PROCEDURE SHOWWORD(VALUE:DATAWORD);
(* USED TO DISPLAY INCOMING DATA FROM RECEIVER AS-IS *)
BEGIN (* SHOWWORD *)
   GOTOXY(XWORD,SCRWORD);
   WRITE(VALUE:5);
   IF SCRWORD=YWORDMAX THEN
      SCRWORD:=YWORDMIN
   ELSE
      BEGIN
         SCRWORD:=SCRWORD+1;
         GOTOXY(XWORD,SCRWORD);
         WRITE(' ':5)
      END
END; (* SHOWWORD *)


PROCEDURE SHOWPROC(NAME:STRING;FTN:FTNTYPE);
(* USED TO DISPLAY CURRENTLY EXECUTING PROCEDURES FOR DEBUGGING *)
(* IF THIS ROUTINE IS CALLED WITH FTN=SHOW, THEN THE PROCNAME   *)
(* IS DISPLAYED ON THE SCREEN, UNDERNEATH ALL PREVIOUS NAMES.   *)
(* A SUBSEQUENT CALL WITH FTN=ERASE WILL CAUSE ALL PROCNAMES UP *)
(* TO THE NAME SPECIFIED TO BE DELETED FROM THE SCREEN. IN THIS *)
(* WAY, A SUBROUTINE CAN "EXIT" FROM ITS CALLER AND REMOVE BOTH *)
(* NAMES FROM THE SCREEN AT ONCE.                               *)
VAR FOUND:BOOLEAN;
BEGIN (* SHOWPROC *)
   IF FTN=SHOW THEN
      BEGIN
         GOTOXY(XPROC,SCRPROC);
         PROCNAMES[SCRPROC]:=NAME;
         WRITE(NAME:8);
         SCRPROC:=SCRPROC+1
      END
   ELSE IF FTN=ERASE THEN
      BEGIN
         FOUND:=FALSE;
         REPEAT
            GOTOXY(XPROC,SCRPROC);
            WRITE(' ':8);
```

```
            FOUND:=PROCNAMES[SCRPROC]=NAME;
            PROCNAME[SCRPROC]:='';
            SCRPROC:=SCRPROC-1
         UNTIL FOUND;
         SCRPROC:=SCRPROC+1;
      END;
END; (* SHOWPROC *)


PROCEDURE FORMATSCREEN;
(* ROUTINE TO READ SCREEN FILE AND INITIALIZE SCREEN DISPLAY *)
CONST ENDOFSCREEN='$'; (* CHAR IN FILE TO INDICATE END OF DISPLAY *)
VAR SCRNFILE:FILE;
    BYTECNT,BLOCKCNT:INTEGER;
BEGIN (* FORMATSCREEN *)
   PAGE(OUTPUT);
   (*$I-*) RESET(SCRNFILE,'#4:RCV.SCREEN.TEXT'); (*$I+*)
   IF IORESULT<>0 THEN
      BEGIN
         GOTOXY(0,7);
         WRITELN('UNABLE TO OPEN #4:RCV.SCREEN.TEXT');
         NOTE(35,50);
         EXIT(PROGRAM)
      END;
   BYTECNT:=512;
   BLOCKCNT:=BLOCKREAD(SCRNFILE,BUFFER,1,1);
   WHILE (IORESULT=0) AND (BYTECNT=512) DO
      BEGIN
         BYTECNT:=SCAN(512,=ENDOFSCREEN,BUFFER);
         UNITWRITE(1,BUFFER,BYTECNT,0,2);
         BLOCKCNT:=BLOCKREAD(SCRNFILE,BUFFER,1)
      END;
   CLOSE(SCRNFILE);
   SCRPROC   :=YPROCMIN;
   SCRWORD   :=YWORDMIN;
   SCRLINE   :=YLINEMIN;
   CLEARLINES:=FALSE;
END; (* FORMATSCREEN *)


PROCEDURE UNLOCKPASS;
(* ROUTINE TO SET ANNUNCIATOR OUTPUTS AND STROBE TO COMMAND *)
(* RECEIVER TO UNLOCK FROM THE CURRENT PASS.  THESE OUTPUTS *)
(* ARE PART OF THE APPLE GAME I/O CONTROLLERS.              *)
VAR STROBE:INTEGER;
BEGIN (* UNLOCKPASS *)
   TTLOUT(0,TRUE);
   TTLOUT(1,FALSE);
   TTLOUT(2,FALSE);
   TTLOUT(3,TRUE);
   STROBE:=PEEK(-16320);
END; (* UNLOCKPASS *)
```

```
PROCEDURE READPARA;  FORWARD;
PROCEDURE READPASS;  FORWARD;


PROCEDURE READWORD;
(* PROCEDURE TO GET NEXT 4-DIGIT INPUT WORD FROM RECEIVER *)
(* USING ASSEMBLER INPUT QUEUE HANDLER, "GETWORD".        *)
VAR ENDOFPARA,ENDOFPASS:BOOLEAN;

   PROCEDURE SCANKB;
   (* SUB-PROCEDURE TO SCAN KEYBOARD FOR USER INPUT.  TO ISSUE *)
   (* A COMMAND, USER MUST FIRST HIT <ESC> KEY, AND THEN THE   *)
   (* APPROPRIATE KEY FOR HIS COMMAND.                         *)
      CONST QUIT   = 'Q';   (* EXIT PROGRAM AFTER END OF PASS  *)
            STAY   = 'S';   (* CANCELS EFFECT OF QUIT COMMAND  *)
            UNLOCK = 'U';   (* ISSUE UNLOCK-PASS COMMAND       *)
            KILL   = 'K';   (* TERMINATE PROGRAM IMMEDIATELY|  *)
      VAR KBCHR:CHAR;
      BEGIN
         SHOWPROC('SCANKB',SHOW);
         READ(KEYBOARD,KBCHR);
         IF NOT ESCPRESSED THEN
            ESCPRESSED:= KBCHR=CHR(27)
         ELSE
            BEGIN
               ESCPRESSED:=FALSE;
               UNITCLEAR(2);
               IF KBCHR IN [QUIT,STAY,UNLOCK,KILL] THEN
                  CASE KBCHR OF
                     QUIT:QUITREQUESTED:=TRUE;
                     STAY:QUITREQUESTED:=FALSE;
                     UNLOCK:BEGIN
                            UNLOCKPASS;
                            ENDOFPASS:=TRUE
                        END;
                     KILL:BEGIN
                            PARACNT:=0;
                            QUITREQUESTED:=TRUE;
                            ENDOFPASS:=TRUE
                        END;
                  END (* CASE *)
            END;
         SHOWPROC('SCANKB',ERASE);
      END; (* SCANKB *)

BEGIN (* READWORD *)
   SHOWPROC('READWORD',SHOW);
   ENDOFPARA:=FALSE;
   ENDOFPASS:=FALSE;

   IF KEYPRESS THEN SCANKB;
   GETWORD(INPUTWORD);

   IF LENGTH(INPUTWORD)=0 THEN
```

```
      BEGIN
         SHOWMODE('WAIT');
         REPEAT
            IF KEYPRESS THEN SCANKB;
            GETWORD(INPUTWORD)
         UNTIL (LENGTH(INPUTWORD)<>0) OR ENDOFPARA OR ENDOFPASS;
         SHOWMODE('ACTIVE')
      END;

   IF LENGTH(INPUTWORD)=4 THEN
      CASE INPUTWORD[1] OF
         '0': IF INPUTWORD<>'0000' THEN
                 BEGIN
                    ENDOFPARA:=TRUE;
                    SHOWWORD('T2MIN')
                 END
              ELSE
                 BEGIN
                    ENDOFPASS:=TRUE;
                    SHOWWORD('R2MIN')
                 END;
         '1','2','3','4':BEGIN
              ENDOFPARA:=TRUE;
              SHOWWORD(INPUTWORD);
              WRITE(CHR(7))
           END;
         '8':BEGIN
              ENDOFPARA:=TRUE;
              SHOWWORD('S2MIN')
           END;
         'C':BEGIN
              ENDOFPASS:=TRUE;
              SHOWWORD('ENDPS')
           END;
         '5','6','7','9','A','B','D','E','F':
              SHOWWORD(INPUTWORD);
      END; (* CASE *)

   IF ENDOFPASS THEN
      BEGIN
         SHOWPROC('READPASS',ERASE);
         EXIT(READPASS)
      END
   ELSE IF ENDOFPARA THEN
      BEGIN
         SHOWPROC('READPARA',ERASE);
         EXIT(READPARA)
      END;

   SHOWPROC('READWORD',ERASE);
END; (* READWORD *)


PROCEDURE READLINE;
```

```
(* ROUTINE TO FORMAT NEXT LINE OF RECEIVER INPUT (9 WORDS) *)
(* INTO VARIABLE "INPUTLINE".  SEQUENCE CODES OF THE INPUT *)
(* WORDS ARE CHECKED FOR PROPER SEQUENCE, AND THE DOPPLER  *)
(* COUNTS ARE TESTED TO ENSURE THAT THEY CONTAIN ONLY BCD  *)
(* DIGITS.  THIS TESTING IS NOT DESIRED FOR THE SATELLITE  *)
(* MESSAGE (LAST 3 WORDS).                                 *)
VAR WORDNUM,DIGIT:INTEGER;
    DATAERROR    :BOOLEAN;
BEGIN (* READLINE *)
   SHOWPROC('READLINE',SHOW);
   DATAERROR:=FALSE;
   DIGIT:=1;
   FOR WORDNUM:=1 TO 9 DO
      BEGIN
         READWORD;
         IF INPUTWORD[1]<>SEQCODES[WORDNUM] THEN
            DATAERROR:=TRUE
         ELSE
            CASE WORDNUM OF
               1,2,4,5:
                  IF NOT ((INPUTWORD[2] IN ['0'..'9']) AND
                   (INPUTWORD[3] IN ['0'..'9']) AND
                   (INPUTWORD[4] IN ['0'..'9'])) THEN
                     DATAERROR:=TRUE
                  ELSE
                     BEGIN
                        INPUTLINE[DIGIT]  :=INPUTWORD[2];
                        INPUTLINE[DIGIT+1]:=INPUTWORD[3];
                        INPUTLINE[DIGIT+2]:=INPUTWORD[4];
                        DIGIT:=DIGIT+3
                     END;
               3,6:
                  IF NOT (INPUTWORD[4] IN ['0'..'9']) THEN
                     DATAERROR:=TRUE
                  ELSE
                     BEGIN
                        INPUTLINE[DIGIT]:=INPUTWORD[4];
                        DIGIT:=DIGIT+2
                     END;
               7,8,9:
                  BEGIN
                     INPUTLINE[DIGIT]  :=INPUTWORD[2];
                     INPUTLINE[DIGIT+1]:=INPUTWORD[3];
                     INPUTLINE[DIGIT+2]:=INPUTWORD[4];
                     DIGIT:=DIGIT+3
                  END;
            END; (* CASE *)
         IF DATAERROR THEN
            BEGIN
               WRITE(CHR(7));
               SHOWPROC('READPARA',ERASE);
               EXIT(READPARA)
            END;
      END;
```

```
   SHOWPROC('READLINE',ERASE);
END; (* READLINE *)


PROCEDURE READPARA;
(* ROUTINE TO SET TIMESTAMP FOR NEXT PARAGRAPH OF INPUT *)
(* AND THEN TO CALL READLINE ENOUGH TIMES TO OBTAIN A   *)
(* COMPLETE PARAGRAPH.  IF ANY ERRORS OCCUR IN READLINE,*)
(* OR IF READWORD ENCOUNTERS 2-MINUTE MARKS, THEN THIS  *)
(* ROUTINE WILL NEVER COMPLETE AND THUS THE PARACNT     *)
(* POINTER WILL NOT BE ADVANCED, THUS CAUSING THE INPUT *)
(* PARAGRAPH TO BE IGNORED.  NOTE THAT 2-MINUTE MARKS   *)
(* BETWEEN PARAGRAPHS WILL CAUSE THE TIMESTAMP TO BE    *)
(* UPDATED, BUT WILL HAVE NO ILL EFFECTS OTHERWISE.     *)
VAR PARANUM,LINECNT:INTEGER;
    CURRENTTIME:TIMESTAMP;
    DISPLAYSTRING:STRING;
BEGIN (* READPARA *)
   SHOWPROC('READPARA',SHOW);
   PARANUM:=PARACNT+1;

   GOTOXY(XPNUM,YPNUM); WRITE(PARANUM:2);

   READTIME(CURRENTTIME);
   CURRENTTIME[SIZEOF(CURRENTTIME)]:=CHR(13);
   DISPLAYSTRING:='                    '; (* 19 SPACES *)
   MOVELEFT(CURRENTTIME[1],DISPLAYSTRING[1],19);
   SHOWMSG(2,CONCAT('TIMESTAMP = ',DISPLAYSTRING));

   SHOWLINE(INPUTLINE,CLEAR);
   WITH PASSPARA[PARANUM] DO
      BEGIN
         PASSTIME:=CURRENTTIME;
         FOR LINECNT:=1 TO MAXLINE DO
            BEGIN
               GOTOXY(XLNUM,YLNUM); WRITE(LINECNT:2);
               READLINE;
               PASSLINE[LINECNT]:=INPUTLINE;
               SHOWLINE(INPUTLINE,SHOW);
            END
      END;
   PARACNT:=PARANUM;
   SHOWPROC('READPARA',ERASE);
END; (* READPARA *)


PROCEDURE READPASS;
(* THIS PROCEDURE COLLECTS PASS DATA UNTIL EITHER THE *)
(* END OF PASS IS REACHED (READWORD WILL CAUSE EXIT), *)
(* OR UNTIL IT HAS COLLECTED THE MAXIMUM ALLOWABLE    *)
(* NUMBER OF DATA PARAGRAPHS - WHICHEVER OCCURS FIRST.*)
BEGIN (* READPASS *)
   SHOWPROC('READPASS',SHOW);
   PARACNT:=0;
```

```
   REPEAT
      READPARA
   UNTIL (PARACNT=MAXPARA);
   UNLOCKPASS;
   SHOWPROC('READPASS',ERASE);
END; (* READPASS *)


PROCEDURE WRITEPASS; FORWARD; (* REFERENCE NEEDED IN OPENPASSFILE *)


PROCEDURE OPENPASSFILE;
(*THIS PROCEDURE ATTEMPTS TO OPEN A NEW PASS FILE FOR SAVING CURRENT
 PASS DATA IN.  FILE SIZE IS COMPUTED, AND RCV.PARAM IS USED TO MAKE
 A NEW FILE NAME UP.    ATTEMPTS ARE THEN MADE TO PRE-EXTEND THIS
 FILE TO ITS FULL SIZE ON AN OUTPUT DISK, GIVING LAST PREFERENCE TO
 THE (USUALLY) BOOT DISKETTE IN DRIVE #4.  IF ALL ATTEMPTS FAIL, THE
 USER IS PROMPTED BY A HIGH-PITCHED BEEP-BEEP NOISE TO SPECIFY  A
 FURTHER COURSE OF ACTION FOR THE PROGRAM: EITHER TERMINATE, OR TRY
 AGAIN TO FIND SPACE (IE. IF THE USER FIRST INSERTS A NEW DISKETTE).*
VAR PFBLOCKCNT,PREFERENCE,DUMMY:INTEGER;
    REPLY:CHAR;
    PFSIZE,PFDIGITS:STRING[5];
    PFPARTIALNAME  :STRING[19];
BEGIN (* OPENPASSFILE *)
   SHOWPROC('OPENFILE',SHOW);
   STR(PFNUMBER,PFDIGITS);

   (* THE FOLLOWING LINES DETERMINE THE REQUIRED FILE SIZE *)
   (* IN BLOCKS OF THE OUTPUT PASS FILE.  ".TEXT" FILES ARE*)
   (* A SPECIAL CASE BECAUSE THEY REQUIRE A 2-BLOCK HEADER *)
   (* RECORD (WRITTEN BY OPERATING SYSTEM) AND THEY MUST   *)
   (* BE WRITTEN (CREATED) IN EVEN INCREMENTS OF 2-BLOCKS. *)

   IF PFEXTENSION='.TEXT' THEN
      PFBLOCKCNT:=2*(1+(PARACNT*SIZEOF(PARARECORD) DIV 1024))+2
   ELSE
      PFBLOCKCNT:=1+(PARACNT*SIZEOF(PARARECORD) DIV 512);
   STR(PFBLOCKCNT,PFSIZE);
   PFPARTIALNAME:=CONCAT(PFROOTNAME,PFDIGITS,PFEXTENSION,
                                              '[',PFSIZE,']');
   SHOWMSG(1,'');

   (* WE CAN USE THE SAME DISK AS LAST TIME ONLY IF IT WAS *)
   (* NOT THE BOOT DRIVE (#4:). OTHERWISE, WE HAVE TO GO   *)
   (* SEARCHING FOR SPACE ELSEWHERE FIRST.                 *)

   IF PFDEVICE<>'#4:' THEN
      BEGIN
         PFNAME:=CONCAT(PFDEVICE,PFPARTIALNAME);
         SHOWMSG(0,CONCAT('NEW FILE= ',PFNAME));
         SHOWMSG(2,'');
         (*$I-*) REWRITE(PASSFILE,PFNAME); (*$I+*)
         IF IORESULT=0 THEN
```

```
            BEGIN
               SHOWPROC('OPENFILE',ERASE);
               EXIT(OPENPASSFILE)
            END;
      END;

   (* THE FOLLOWING LOGIC SEARCHES FOR AN OUTPUT DISK, IN   *)
   (* THE ORDER OF PRIORITY SPECIFIED WITHIN THE CASE BELOW*)

   REPEAT
      SHOWMSG(2,'[SEARCHING FOR NEW OUTPUT DISK]');
      FOR PREFERENCE:=1 TO 6 DO
         BEGIN
            CASE PREFERENCE OF (* THESE ARE PASCAL DISKETTE UNITS *)
               1: PFDEVICE:='#5:'; (* FIRST CHOICE *)
               2: PFDEVICE:='#11:';(* SECOND CHOICE*)
               3: PFDEVICE:='#12:';(* THIRD CHOICE *)
               4: PFDEVICE:='#9:'; (* FOURTH CHOICE*)
               5: PFDEVICE:='#10:';(* FIFTH CHOICE *)
               6: PFDEVICE:='#4:'  (* LAST RESORT ONLY! *)
            END; (* CASE *)
            PFNAME:=CONCAT(PFDEVICE,PFPARTIALNAME);
            SHOWMSG(0,CONCAT('NEW FILE= ',PFNAME));
            (*$I-*) REWRITE(PASSFILE,PFNAME); (*$I+*)
            IF IORESULT=0 THEN
               BEGIN
                  SHOWPROC('OPENFILE',ERASE);
                  EXIT(OPENPASSFILE)
               END
         END;
      SHOWMSG(1,'NO SPACE FOR OUTPUT FILE');
      SHOWMSG(2,'<ESC>=KILL; <RETURN>=RETRY');
      UNITCLEAR(2);
      WHILE NOT KEYPRESS DO
         BEGIN
            NOTE(45,25);              (* BEEP AND *)
            FOR DUMMY:=1 TO 2000 DO (*  DELAY!  *)
         END;
      READ(KEYBOARD,REPLY);
      SHOWMSG(1,'');
   UNTIL REPLY=CHR(27); (* ESCAPE CHARACTER *)
   QUITREQUESTED:=TRUE;
   SHOWPROC('WRITEPAS',ERASE);
   EXIT(WRITEPASS)
END; (* OPENPASSFILE *)


PROCEDURE CLOSEPASSFILE;
(* THIS ROUTINE CLOSES THE CURRENT PASSFILE AND UPDATES *)
(* RCV.PARAM.TEXT TO REFLECT THE NEXT PASS NUMBER TO BE *)
(* USED IN CREATING PASS FILES.                         *)
BEGIN (* CLOSEPASSFILE *)
   SHOWPROC('CLOSEFIL',SHOW);
   CLOSE(PASSFILE,LOCK);
```

```
   SHOWMSG(1,'PASS FILE SUCCESSFULLY WRITTEN');
   SHOWMSG(2,'[UPDATING RCV.PARAM.TEXT]');
   PFNUMBER:=PFNUMBER+PFINCREMENT;
   REWRITE(PARAMFILE,'#4:RCV.PARAM.TEXT[4]');
   WRITELN(PARAMFILE,PFROOTNAME);
   WRITELN(PARAMFILE,PFNUMBER,' ',PFINCREMENT);
   WRITELN(PARAMFILE,PFEXTENSION);
   CLOSE(PARAMFILE,LOCK);
   SHOWMSG(0,CONCAT('LAST PASS= ',PFNAME));
   SHOWMSG(1,'');
   SHOWMSG(2,'')
END; (* CLOSEPASSFILE *)


PROCEDURE WRITEPASS;
(* THIS ROUTINE HANDLES THE (VERY) FAST TRANSFER OF A GROUP *)
(* OF DATA PARAGRAPHS (IE. THE CURRENT PASS) TO A PASS FILE *)
(* ON DISKETTE.  THE TWO PROCEDURES ABOVE AID IN THIS QUEST.*)
VAR  PARANUM:INTEGER;
BEGIN (* WRITEPASS *)
   SHOWPROC('WRITEPAS',SHOW);
   RESETIRQ; (* DISABLE INTERRUPTS WHILE USING DISKETTE DRIVES *)
   SHOWMODE('DISKIO');

   OPENPASSFILE;
   FOR PARANUM:=1 TO PARACNT DO
      BEGIN
         PASSFILE↑:=PASSPARA[PARANUM];
         PUT(PASSFILE)
      END;
   CLOSEPASSFILE;

   INITPIA; (* ENABLE INTERRUPTS AGAIN *)
   SHOWMODE('ACTIVE');
   SHOWPROC('WRITEPAS',ERASE);
END; (* WRITEPASS *)


PROCEDURE SETPARAMETERS;
(* THIS ROUTINE ATTEMPTS TO READ THE PASS FILE NAMING *)
(* PARAMETERS FROM #4:RCV.PARAM.TEXT.  IF THE FILE    *)
(* CANNOT BE OPENED, AN ERROR MESSAGE IS DISPLAYED    *)
(* AND THE PROGRAM TERMINATES.                        *)
(* THE PARAMETERS EXPECTED ARE: (ON SEPARATE LINES)   *)
(*      1.   DEVICENAME:ROOTSUFFIX                    *)
(*      2.   NEXTPASSNUMBER  PASSNUMBERINCREMENT      *)
(*      3.   .EXTENSION                               *)
(* TYPICAL VALUES FOR THESE WOULD BE:                 *)
(*           #5:PASS                                  *)
(*           100  10                                  *)
(*           .TEXT                                    *)
BEGIN
   SHOWPROC('SETPARAM',SHOW);
   SHOWMODE('DISKIO');
```

```
   PFDEVICE:='#5:';
   (*$I-*) RESET(PARAMFILE,'#4:RCV.PARAM.TEXT'); (*$I+*)
   IF IORESULT<>0 THEN
      BEGIN
         SHOWMSG(1,'UNABLE TO OPEN #4:RCV.PARAM.TEXT');
         NOTE(35,50);
         EXIT(PROGRAM)
      END;
   READLN(PARAMFILE,PFROOTNAME);
   READLN(PARAMFILE,PFNUMBER,PFINCREMENT);
   READLN(PARAMFILE,PFEXTENSION);
   CLOSE(PARAMFILE,NORMAL);
   SHOWPROC('SETPARAM',ERASE);
END;


BEGIN (* RECEIVER *)
   FORMATSCREEN;
   SHOWPROC('RECEIVER',SHOW);

   STR((2*MEMAVAIL),MEMUNUSED);
   SHOWMSG(1,CONCAT('MEMAVAIL AT STARTUP = ',MEMUNUSED,' BYTES'));

   SETPARAMETERS;

   INITPIA; (* ENABLE INTERRUPTS *)
   SHOWMODE('ACTIVE');

   INPUTLINE[8] :=' ';
   INPUTLINE[16]:=' ';
   INPUTLINE[26]:=CHR(13);

   SEQCODES     :='5679ABDEF';
   ESCPRESSED   :=FALSE;
   QUITREQUESTED:=FALSE;

   UNITCLEAR(2); (* CLEAR KEYBOARD TYPE-AHEAD BUFFER *)

   REPEAT
      SHOWMSG(1,'USER <ESC> COMMANDS: Q,S,U,K');
      READPASS;
      IF PARACNT>0 THEN
         WRITEPASS
   UNTIL QUITREQUESTED;

   SHOWMODE('QUIT');
   RESETIRQ;
   SHOWPROC('RECEIVER',ERASE);
   PAGE(OUTPUT)
END. (* RECEIVER *)
```

II.2 SATLITE ROUTINES LISTING

```
          .TITLE   "SATLITE - SATELLITE INTERFACE ROUTINES"
          .NOMACROLIST
          .NOPATCHLIST

;  MACRO TO POP 16-BIT RETURN ADDRESS:

          .MACRO POP
          PLA
          STA     %1
          PLA
          STA     %1+1
          .ENDM

;  MACRO TO PUSH 16-BIT RETURN ADDRESS:

          .MACRO PUSH
          LDA     %1+1
          PHA
          LDA     %1
          PHA
          .ENDM

;  MEMORY MAP FOR 6821 PERIPHERAL INTERFACE ADAPTER:

PIASLOT   .EQU  1          ;APPLE SLOT NUMBER OF PARALLEL INTERFACE
PIABASE   .EQU  <PIASLOT*10>+0C080
PIADRA    .EQU  PIABASE+0  ;SIDE "A" DATA DIRECTION REGISTER
PIAPRA    .EQU  PIABASE+0  ;SIDE "A" PERIPHERAL INTERFACE REGISTER
PIASRA    .EQU  PIABASE+1  ;SIDE "A" STATUS REGISTER
PIACRA    .EQU  PIABASE+1  ;SIDE "A" COMMAND REGISTER
PIADRB    .EQU  PIABASE+2  ;SIDE "B" DATA DIRECTION REGISTER
PIAPRB    .EQU  PIABASE+2  ;SIDE "B" PERIPHERAL INTERFACE REGISTER
PIASRB    .EQU  PIABASE+3  ;SIDE "B" STATUS REGISTER
PIACRB    .EQU  PIABASE+3  ;SIDE "B" COMMAND REGISTER

;  SPECIAL SYSTEM MONITOR LOCATIONS:

IRQVECTR .EQU  0FFFE      ;BASE ADDRESS OF IRQ/BRK INTERRUPT VECTOR
LANGCARD .EQU  0C080      ;BASE ADDRESS FOR SLOT#0 = LANGUAGE-CARD

;  PASCAL-SUPPLIED ZERO-PAGE TEMPORARY WORK AREAS:

RTADDR    .EQU  00        ;SAVE AREA FOR PASCAL RETURN ADDRESS
STRING    .EQU  02        ;USED FOR INDIRECT PARAM ADDRESS IN READPIA

;  ROUTINE TO INITIALIZE PIA AND BUFFER QUEUE:

          .PROC INITPIA   ;ROUTINE TO INITIALIZE PIA HANDLING
```

```
          .DEF  OLDIRQ
          .REF  QFWDPTR,QBKWPTR,QBYTE1,QBYTE2,IRQHANDL

START     SEI             ;DISABLE INTERRUPTS UNTIL DONE
          POP   RTADDR    ;POP RETURN ADDRESS FROM STACK

          LDA   #00       ;CLEAR ACCUMULATOR
          STA   PIACRA    ;REQUEST ACCESS TO DDRA
          STA   PIADRA    ;SET ALL BITS FOR INPUT
          STA   PIACRB    ;REQUEST ACCESS TO DDRB
          STA   PIADRB    ;SET ALL BITS FOR INPUT
          LDA   #05       ;LOAD IN COMMAND BITS
          STA   PIACRA    ;SET UP COMMAND REGISTER A
          LDA   #04       ;LOAD IN COMMAND BITS
          STA   PIACRB    ;SET UP COMMAND REGISTER B

          LDA   #00       ;LOAD INITIAL VALUE FOR BACKWARD POINTER
          STA   QBKWPTR   ;SAVE BACKWARD POINTER
          LDA   #01       ;SET QFWDPRT TO ONE GREATER THAN QBKWPTR
          STA   QFWDPTR   ;SAVE FORWARD POINTER

          LDA   LANGCARD+0B ;REMOVE LANGUAGE-CARD WRITE-PROTECTION
          LDA   LANGCARD+0B ;THIS INSTRUCTION HAS TO BE DONE TWICE

          LDA   IRQVECTR  ;GET LSB OF CURRENT IRQ VECTOR
          STA   OLDIRQ    ;SAVE FOR INTERRUPT HANDLER
          LDA   IRQVECTR+1;GET MSB OF CURENT IRQ VECTOR
          STA   OLDIRQ+1  ;SAVE FOR INTERRUPT HANDLER

          LDA   IRQADR    ;GET MSB OF IRQ ROUTINE ADDRESS
          STA   IRQVECTR  ;STORE IN MSB OF IRQ VECTOR
          LDA   IRQADR+1  ;GET LSB OF IRQ ROUTINE ADDRESS
          STA   IRQVECTR+1;STORE IN LSB OF IRQ VECTOR

          LDA   LANGCARD+8;WRITE PROTECT THE LANGUAGE-CARD AGAIN

          CLI             ;ENABLE INTERRUPTS AGAIN

          PUSH  RTADDR    ;PUSH RETURN ADDRESS BACK ONTO STACK
          RTS             ;RETURN TO CALLING PROGRAM
OLDIRQ    .WORD  0000     ;SAVE AREA FOR ORIGINAL MONITOR IRQ VECTOR
IRQADR    .WORD  IRQHANDL ;ADDRESS OF INTERRUPT ROUTINE, 16-BITS

;  PROC TO DISABLE INTERRUPTS AND RESTORE ORIGINAL IRQ/BRK VECTOR

          .PROC RESETIRQ  ;CLEANUP ROUTINE FOR END-OF-PROCESSING
          .REF  OLDIRQ

START     SEI             ;DISABLE INTERRUPTS

          LDA   #00       ;CMD WORD FOR PIA = NO INTERRUPTS ALLOWED
          STA   PIACRA    ;STORE IN A-SIDE COMMAND REGISTER
          STA   PIACRB    ;STORE IN B-SIDE COMMAND REGISTER
```

```
         LDA   LANGCARD+0B ;REMOVE LANGUAGE-CARD WRITE-PROTECTION
         LDA   LANGCARD+0B ;THIS INSTRUCTION HAS TO BE DONE TWICE
         LDA   OLDIRQ      ;GET LSB OF ORIGINAL IRQ ADDRESS
         STA   IRQVECTR    ;STORE IN IRQ VECTOR
         LDA   OLDIRQ+1    ;GET MSB OF ORIGINAL IRQ ADDRESS
         STA   IRQVECTR+1  ;STORE IN IRQ VECTOR
         LDA   LANGCARD+8  ;WRITE PROTECT THE LANGUAGE-CARD AGAIN

         RTS               ;RETURN TO CALLING PROGRAM

;  PROCEDURE TO RETURN THE NEXT "WORD" FROM THE QUEUE:

         .PROC GETWORD,1 ;PROCEDURE TO RETRIEVE INPUT WORDS
         .DEF  IRQHANDL,QBYTE1,QBYTE2,QBKWPTR,QFWDPTR
         .REF  OLDIRQ

EMPTYCHR .EQU  20          ;EMPTY QUEUE INDICATOR CHARACTER = SPACE

START    POP   RTADDR      ;SAVE PASCAL RETURN ADDRESS
         POP   STRING      ;SAVE ADDRESS OF STRING PARAMETER

         LDY   #00         ;USE Y AS STRING INDEX - SET TO "LENGTH" BYT
         LDX   QBKWPTR     ;GET BACKWARD POINTER FOR BUFFER QUEUE
         INX               ;POINT TO NEXT WORD IN BUFFER
         CPX   QFWDPTR     ;CHECK FOR EMPTY QUEUE
         BNE   GETBYTE1    ;BRANCH IF NOT EMPTY

UNDFFLOW LDA   #00         ;SET LENGTH OF STRING TO ZERO
         STA   @STRING,Y   ;STORE A SPACE CHARACTER
         BEQ   EXITGET     ;ALWAYS BRANCH (TO EXIT)

GETBYTE1 LDA   #04         ;SET LENGTH OF STRING TO 4 BYTES
         STA   @STRING,Y   ;SAVE IN "LENGTH" BYTE
         LDA   QBYTE1,X    ;GET FIRST HALF OF 16-BIT INPUT WORD
         LSR   A           ;SHIFT UPPER NIBBLE TO LEFT SIDE OF ACCUM
         LSR   A
         LSR   A
         LSR   A
         ORA   #30         ;CONVERT TO ASCII
         CMP   #3A         ;CHECK FOR NON-NUMERIC DIGIT
         BMI   ST1         ;BRANCH IF DIGIT IN RANGE 0->9
         CLC               ;CLEAR CARRY FOR ADD
         ADC   #07         ;CONVERT DIGIT TO HEX CHAR A->F
ST1      INY               ;POINT AT FIRST BYTE OF STRING
         STA   @STRING,Y   ;SAVE AS FIRST CHARACTER IN STRING
         LDA   QBYTE1,X    ;GET ORIGINAL VALUE AGAIN
         AND   #0F         ;ISOLATE LOWER NIBBLE
         ORA   #30         ;CONVERT TO ASCII
         INY               ;POINT AT SECOND BYTE OF STRING
         STA   @STRING,Y   ;SAVE AS SECOND CHARACTER IN STRING

GETBYTE2 LDA   QBYTE2,X    ;GET SECOND HALF OF 16-BIT INPUT WORD
         LSR   A           ;SHIFT UPPER NIBBLE TO LEFT SIDE OF ACCUM
         LSR   A
```

```
          LSR   A
          LSR   A
          ORA   #30         ;CONVERT TO ASCII
          INY               ;POINT AT THIRD BYTE OF STRING
          STA   @STRING,Y   ;SAVE AS THIRD CHARACTER IN STRING
          LDA   QBYTE2,X    ;GET ORIGINAL VALUE AGAIN
          STX   QBKWPTR     ;SAVE NEW QUEUE POINTER NOW THAT DATA IS SAF
          AND   #0F         ;ISOLATE LOWER NIBBLE
          ORA   #30         ;CONVERT TO ASCII
          INY               ;POINT AT FOURTH BYTE OF STRING
          STA   @STRING,Y   ;SAVE AS FOURTH CHARACTER IN STRING

EXITGET   PUSH  RTADDR      ;PUSH PASCAL RETURN ADDRESS ON STACK
          RTS               ;RETURN TO CALLING PROGRAM

QBYTE1    .BLOCK 256        ;QUEUE AREA FOR FIRST 8 BITS (15-8) OF INPUT
QBYTE2    .BLOCK 256        ;QUEUE AREA FOR SECOND 8 BITS (7-0) OF INPUT
QFWDPTR   .BYTE  00         ;POINTER TO FRONT OF QUEUE
QBKWPTR   .BYTE  00         ;POINTER TO REAR OF QUEUE

;  INTERRUPT-DRIVEN ROUTINE TO BUFFER DATA FROM THE PIA.
;  TO MINIMIZE THE TIME REQUIRED TO SERVICE INTERRUPTS,
;  THIS ROUTINE HAS NOT BEEN CODED FOR RE-ENTRANCY.
;  AS A RESULT, INTERRUPTS REMAIN DISABLED WHILE THIS
;  ROUTINE EXECUTES, AND ARE RE-ENABLED BY THE RTI INSTRUCTION.

OVFLCHAR .EQU  11           ;"UNUSED" SEQ CODE - USE AS OVERFLOW FLAG

IRQHANDL STA   SAVEACC      ;SAVE ACCUMULATOR
          PLA               ;GET STATUS REG FROM STACK
          PHA               ;RESTORE ONTO STACK
          AND   #10         ;TEST "B" BIT
          BEQ   NOTBRK      ;SKIP NEXT SECTION IF TRUE INTERRUPT

NOTPIA    LDA   SAVEACC     ;RESTORE ACCUMULATOR CONTENTS
          JMP   @OLDIRQ     ;BRANCH TO MONITOR'S IRQ/BRK ROUTINE

NOTBRK    LDA   PIASRA      ;WAS IRQ CAUSED BY PIA?
          BPL   NOTPIA      ;IF NOT, BRANCH TO ORIGNIAL IRQ ROUTINE

          TXA               ;SAVE INDEX-X ON STACK
          PHA

          LDX   QFWDPTR     ;SET UP QUEUE POINTER IN INDEX-X
          CPX   QBKWPTR     ;CHECK FOR FULL QUEUE
          BNE   SAVEDATA    ;BRANCH IF QUEUE IS OK

OVERRUN   LDA   #OVFLCHAR   ;LOAD QUEUE OVERFLOW CHARACTER
          DEX               ;POINT AT PREVIOUS QUEUE ELEMENTS
          STA   QBYTE1,X    ;SAVE IN PLACE OF LAST 16-BITS IN QUEUE
          STA   QBYTE2,X
          BNE   EXITIRQ     ;ALWAYS BRANCH

SAVEDATA  LDA   PIAPRB      ;GET BITS 15-8 OF INPUT FROM PIA-B
```

```
          EOR    #0FF        ;INVERT ALL BITS
          STA    QBYTE1,X    ;SAVE THEM AS QBYTE1
          LDA    PIAPRA      ;GET BITS 7-0 OF INPUT FROM PIA-A
          EOR    #0FF        ;INVERT ALL BITS
          STA    QBYTE2,X    ;SAVE THEM AS QBYTE2
          INX                ;ADVANCE QUEUE POINTER TO NEXT POSITION
          STX    QFWDPTR     ;SAVE NEW FORWARD POINTER FOR QUEUE

EXITIRQ   PLA                ;RESTORE INDEX-X FROM STACK
          TAX
          LDA    SAVEACC     ;RESTORE ACCUMULATOR
          RTI                ;RETURN TO INTERRUPTED ROUTINE
SAVEACC   .BYTE  00          ;ACCUM SAVE AREA FOR INTERRUPT ROUTINE

          .END
```

II.3 TALK PROGRAM LISTING

```
(*************************************
 * PROGRAM: TALK                     *
 * WRITTEN: 19-APR-82 BY MARK S LORD *
 *-----------------------------------*
 * THIS PROGRAM ALLOWS COMMUNICATIONS *
 * BETWEEN THE APPLE COMPUTER AND AN  *
 * OUTSIDE SOURCE, VIA THE SERIAL I/O *
 * INTERFACE CARD IN APPLE SLOT #2.   *
 *                                    *
 * THREE DIFFERENT MODES OF OPERATION *
 * CAN BE USED AS SELECTED FROM THE   *
 * PROGRAM'S MAIN MENU:               *
 *                                    *
 * D)UMB TERMINAL MODE:               *
 *    THIS OPTION CAUSES THE APPLE    *
 *    TO BEHAVE AS IF IT WERE A "DUMB"*
 *    ASCII TERMINAL, OPERATING IN    *
 *    HALF-DUPLEX MODE.  CHARACTERS   *
 *    TYPED AT THE KEYBOARD ARE ECHOED*
 *    LOCALLY AND ALSO SENT OUT VIA   *
 *    THE SERIAL INTERFACE; CHARACTERS*
 *    RECEIVED FROM THE INPUT SIDE OF *
 *    THE ACIA ARE DISPLAYED ON THE   *
 *    APPLE MONITOR  AS  THEY  ARE    *
 *    RECEIVED.  SINCE LOCAL I/O IS   *
 *    DONE USING THE PASCAL READ/WRITE*
 *    PROCEDURES, NOT ALL CHARACTERS  *
 *    TYPED AT THE KEYBOARD WILL BE   *
 *    PROCESSED BY THIS PROGRAM.  FOR *
 *    EXAMPLE, THE CTRL¬A, CTRL¬S... .*
 *    IN ADDITION, THIS PROGRAM ALSO  *
 *    USES ITS OWN SPECIALLY DEFINED  *
 *    KEYS:  <CTRL¬C>  RETURNS USER TO*
 *                MAIN PROGRAM MENU.  *
 *           <ESC>  FUNCTIONS THE SAME*
 *                AS A "BREAK" KEY.  *
 *           <RIGHTARROW>  ACTS AS A  *
 *                VSPC CHARACTER <DEL>*
 *                KEY.                *
 *           <LEFTARROW>  SENDS A TAB *
 *                CHARACTER TO VSPC.  *
 *                                    *
 * T)RANSFER TEXT MODE:               *
 *    THIS OPTION IS SPECIFICALLY FOR *
 *    USE WITH VSPC, AND THUS ASSUMES *
 *    THAT THE USER HAS PREVIOUSLY    *
 *    SIGNED ONTO VSPC, AND THAT THE  *
 *    VSPC "TAPE" COMMAND HAS BEEN    *
```

```
 *    PREVIOUSLY ISSUED (EVEN THOUGH IT*
 *    IS ALSO SENT AGAIN BY THIS       *
 *    PROGRAM SECTION). THE NAME OF AN *
 *    APPLE ".TEXT" FILE IS REQUESTED  *
 *    AND THEN THE NAME FOR A CORRES-  *
 *    PONDING VSPC WORKSPACE TO WHICH  *
 *    THE CONTENTS OF THAT FILE ARE TO *
 *    BE COPIED.  THE PROGRAM THEN     *
 *    PROCEEDS WITH THE VSPC COMMANDS  *
 *    NECESSARY TO TRANSFER THE FILE   *
 *    LINE-BY-LINE TO THE VSPC WS.     *
 *    NOTE THAT "DUMB TERMINAL" MODE   *
 *    AND ALL OF ITS OPTIONS ARE ALSO 0*
 *    IN EFFECT THROUGHO+T THE TRANSFER*
 *    SO THAT THE <ESC> AND <CTRL¬C>   *
 *    KEYS MAY BE USED TO PREMATURELY  *
 *    TERMINATE THE TRANSFER.          *
 *                                     *
 * P) ASSFILE TRANSFERING:             *
 *    THIS OPTION IS VERY SIMILAR TO   *
 *    THE OPTION ABOVE IN ALL RESPECTS *
 *    EXCEPT THAT IT IS INTENDED TO    *
 *    STREAMLINE THE TRANSFER OF THE   *
 *    SPECIALLY FORMATTED "PASS FILES" *
 *    PRODUCED BY THE RECEIVER PROGRAM.*
 *    THE USER IS PROMPTED FOR A       *
 *    ROOTNAME, WHICH CONSISTS OF THE  *
 *    NON-NUMERIC PORTIONS OF THE FULL *
 *    PASSFILE NAME.  FOR EXAMPLE, IF  *
 *    THE PASSFILES WERE NAMED USING   *
 *    THE STANDARD CHARACTER SEQUENCE: *
 *         #5:PASS340.TEXT             *
 *         #5:PASS350.TEXT             *
 *         #5:PASS370.TEXT             *
 *    THEN THE APPROPRIATE ROOTNAME    *
 *    WOULD BE:  #5:PASS.TEXT          *
 *    THE USER IS THEN PROMPTED FOR    *
 *    THE SEQUENCE NUMBER RANGE FOR    *
 *    THESE FILES, THUS ALLOWING       *
 *    SEVERAL PASSES TO BE TRANSFERED, *
 *    AND FINALLY, THE USER IS ASKED   *
 *    FOR A PASS NUMBER INCREMENT. IF  *
 *    THE FILES WERE NAMED AS ABOVE,   *
 *    THEN THESE VALUES WOULD BE       *
 *         340, 370, 10                *
 *    NOTE THAT ALTHOUGH PASS380.TEXT  *
 *    DOES NOT EXIST, THIS WILL CAUSE  *
 *    NO PROBLEMS SINCE THE PROGRAM    *
 *    WILL SIMPLY PRINT A MESSAGE TO   *
 *    THIS EFFECT AND THEN CONTINUE    *
 *    ONWARD.                          *
 ************************************)
PROGRAM TALK;
USES APPLESTUFF,PEEKPOKE;
```

```
CONST ACIASTATUS =-16210; (* ADDRESS OF ACIA STATUS REG. *)
      ACIADATA   =-16209; (* ADDRESS OF ACIA DATA REG.   *)
      ACIABREAK  =    96; (* ACIA COMMAND FOR "BREAK"    *)
      ACIARESET  =     3; (* ACIA COMMAND FOR CHIP RESET *)
      ACIASPEED  =    17; (* ACIA SPEED SELECT = 300 BAUD*)
      ESCAPE     =    27; (* ASCII CODE FOR <ESC> CHAR.  *)
      LINEFEED   =    10; (* ASCII CODE FOR <LF> CHAR.   *)
      LEFTARROW  =     8; (* CODE FOR SPECIAL APPLE KEY  *)
      RIGHTARROW =    21; (* CODE FOR SPECIAL APPLE KEY  *)
      CTRLC      =     3; (* ASCII CODE FOR CONTROL-C    *)
      DC1        =    17; (* ASCII CODE FOR DC1 CHAR.    *)


TYPE  LONGSTRING = STRING[255];
TYPE  DATALINE   = PACKED ARRAY[1..26] OF CHAR;
      TIMESTAMP  = PACKED ARRAY[1..20] OF CHAR;
      PARARECORD = RECORD
          PASSTIME:TIMESTAMP;
          PASSLINE:ARRAY[1..25] OF DATALINE
          END;

VAR KBCHR           :CHAR;
    KBVAL,REPLYVAL  :INTEGER;
    QUITREQUESTED   :BOOLEAN;
    ROOTNAME,PASSNAME,NUMSTRING,VSPCNAME,OUTSTRING:STRING;
    PASSPARA        :PARARECORD;

    PASSFILE        :FILE OF PARARECORD;


PROCEDURE SCANACIA;
(*****************************************
 * THIS ROUTINE SCANS THE ACIA FOR       *
 * INCOMING DATA.  IF DATA IS PRESENT,   *
 * IT IS DISPLAYED ON THE APPLE MONITOR  *
 * AND THE ASCII NUMERIC VALUE IS        *
 * PLACED IN "REPLYCHR".  OTHERWISE,     *
 * "REPLYCHR" IS SET TO ZERO.            *
 *****************************************)
BEGIN (* SCANACIA *)
   IF ODD(PEEK(ACIASTATUS)) THEN
      BEGIN
         REPLYVAL:=PEEK(ACIADATA);
         WRITE(CHR(REPLYVAL))
      END
   ELSE
      REPLYVAL:=0;
END; (* SCANACIA *)


PROCEDURE SENDACIA(OUTVALUE:INTEGER);
(*****************************************
 * THIS ROUTINE WILL TRANSMIT A BYTE     *
 * OUT THROUGH THE ACIA.  IT WAITS       *
```

```
 * UNTIL THE "READY" FLAG OF THE ACIA  *
 * IS SET, AND THEN TRANSFERS THE DATA *
 * BYTE SPECIFIED BY ITS ASCII NUMERIC *
 * VALUE IN "OUTVALUE".                *
 ***************************************)
VAR STATUS:INTEGER;
BEGIN (* SENDACIA *)
   REPEAT
      STATUS:=PEEK(ACIASTATUS) DIV 2
   UNTIL ODD(STATUS);
   POKE(ACIADATA,OUTVALUE)
END; (* SENDACIA *)


PROCEDURE PROCESSCOMMAND; FORWARD;


PROCEDURE SCANKEYBOARD;
(***************************************
 * THIS ROUTINE CHECKS TO SEE IF ANY   *
 * MORE KEYBOARD INPUT HAS BEEN ENTERED*
 * BY THE USER.  IF SO, IT IS PROCESSED*
 * AS DESCRIBED AT THE TOP OF THIS     *
 * PROGRAM IN THE D)DUMB TERMINAL CMD. *
 ***************************************)
BEGIN (* SCANKEYBOARD *)
   IF KEYPRESS THEN
      BEGIN
         READ(KEYBOARD,KBCHR);
         IF EOLN(KEYBOARD) THEN
            KBCHR:=CHR(13);
         KBVAL:=ORD(KBCHR);
         IF KBVAL IN [ESCAPE,CTRLC,LEFTARROW,RIGHTARROW] THEN
            CASE KBVAL OF
               ESCAPE:
                  BEGIN
                     POKE(ACIASTATUS,ACIABREAK);
                     NOTE(40,25);
                     POKE(ACIASTATUS,ACIASPEED)
                  END;
               CTRLC:
                  BEGIN
                     WRITELN(CHR(7),'<CTRL¬C>');
                     EXIT(PROCESSCOMMAND)
                  END;
               LEFTARROW:
                  BEGIN
                     WRITE(KBCHR,' ',KBCHR);
                     SENDACIA(KBVAL);
                     SENDACIA(LINEFEED)
                  END;
               RIGHTARROW:
                  BEGIN
                     KBCHR:=CHR(9);
```

```
                    WRITE(KBCHR);
                    SENDACIA(9)
                 END;
            END (* CASE *)
         ELSE
            BEGIN
               WRITE(KBCHR);
               SENDACIA(KBVAL)
            END
      END;
END; (* SCANKEYBOARD *)


PROCEDURE DUMBTERMINAL;
(*************************************
 * THIS ROUTINE ALLOWS DIRECT USER   *
 * COMMUNICATIONS WITH A REMOTE DEVICE *
 * BY CAUSING THE APPLE TO BEHAVE AS A *
 * NON-INTELLIGENT ASYNC ASCII TERMINAL*
 *************************************)
BEGIN (* DUMBTERMINAL *)
   WRITELN('== DUMB TERMINAL MODE ==');
   WRITELN;
   WRITELN('== HIT <CTRL-C> TO QUIT ==');
   WRITELN(CHR(7));
   KBVAL:=0;
   REPEAT
      SCANACIA;
      SCANKEYBOARD
   UNTIL KBVAL=CTRLC;
END; (* DUMBTERMINAL *)


PROCEDURE XMITVSPC(MESSAGE:LONGSTRING);
(*************************************
 * THIS ROUTINE USES "SENDACIA" TO   *
 * TRANSMIT A LINE OF CHARACTERS TO  *
 * VSPC.  A CARRIAGE-RETURN IS SENT AT *
 * THE END OF THE LINE, AND ALL CHARS  *
 * SENT ARE ALSO ECHOED ON THE APPLE'S *
 * MONITOR AS THEY ARE TRANSMITTED.  *
 *************************************)
VAR REPLY:CHAR;
    I:INTEGER;
BEGIN (* XMITVSPC *)
   MESSAGE:=CONCAT(MESSAGE,' ');
   MESSAGE[LENGTH(MESSAGE)]:=CHR(13);
   I:=0;
   REPEAT
      I:=I+1;
      SCANKEYBOARD;
      SCANACIA;
      WRITE(MESSAGE[I]);
      SENDACIA(ORD(MESSAGE[I]))
```

```
   UNTIL MESSAGE[I]=CHR(13);
   REPEAT
      SCANKEYBOARD;
      SCANACIA
   UNTIL REPLYVAL=DC1;
END; (* XMITVSPC *)


PROCEDURE SENDPASS;
(*************************************
 * THIS ROUTINE HANDLES THE ACTUAL     *
 * TRANSFER OF A PRE-OPENED PASS FILE  *
 * TO VSPC.  A VSPC WORKSPACE IS NAMED *
 * AND SAVED FOR THE PASS, THE NAME    *
 * USED BEING THE SAME AS THAT OF THE  *
 * PASS FILE, LESS DEVICE NAME AND     *
 * EXTENSION OF COURSE.                *
 *************************************)
VAR DOTPOS,LINENUM:INTEGER;
BEGIN (* SENDPASS *)
   XMITVSPC('CLEAR');
   VSPCNAME:=PASSNAME;
   DELETE(VSPCNAME,1,POS(':',VSPCNAME));
   DOTPOS:=POS('.',VSPCNAME);
   DELETE(VSPCNAME,DOTPOS,(1+LENGTH(VSPCNAME)-DOTPOS));
   XMITVSPC(CONCAT('NAME ',VSPCNAME));
   XMITVSPC('INPUT 1 1');
   REPEAT
      WITH PASSPARA DO
         BEGIN
            PASSPARA:=PASSFILE¬;
            OUTSTRING:='                         ';
            MOVELEFT(PASSTIME[1],OUTSTRING[1],19);
            XMITVSPC(OUTSTRING);
            OUTSTRING:='                              ';
            FOR LINENUM:=1 TO 25 DO
               BEGIN
                  MOVELEFT(PASSLINE[LINENUM],OUTSTRING[1],25);
                  XMITVSPC(OUTSTRING)
               END;
            GET(PASSFILE)
         END
   UNTIL EOF(PASSFILE);
   XMITVSPC('');
   XMITVSPC(CONCAT('SAVE ',VSPCNAME));
   CLOSE(PASSFILE);
END; (* SENDPASS *)


PROCEDURE PASSTRANSFER;
(*************************************
 * THIS ROUTINE SERVES AS THE DRIVER   *
 * FOR THE SENDPASS ROUTINE. IT PROMPTS*
 * THE USER FOR THE PASS FILE RANGES   *
```

```
 * AND THEN LOOPS, CALLING SENDPASS TO *
 * TRANSFER INDIVIDUAL PASS FILES. OPEN*
 * ERRORS ARE LOGGED ON THE SCREEN FOR *
 * THE USER TO OBSERVE AS THE PROGRAM  *
 * CONTINUES WITH THE NEXT FILE IN SEQ.*
 ***************************************)
VAR DOTPOS,IOERR,PASSNUM,LASTNUM,INCREMENT:INTEGER;
BEGIN (* PASSTRANSFER *)
   WRITELN('== PASS FILE TRANSFER PROCEDURE ==');
   WRITELN;
   WRITELN('== ENTER ROOT-NAME (DEV:SUFFIX.EXT) ==');
   (*$I-*)
   REPEAT
      REPEAT
         WRITE  ('== ENTER ==> ');
         READLN(ROOTNAME);
         IF LENGTH(ROOTNAME)=0 THEN
            EXIT(PASSTRANSFER);
         DOTPOS:=POS('.',ROOTNAME)
      UNTIL NOT(DOTPOS IN [0,1,LENGTH(ROOTNAME)]);
      RESET(PASSFILE,ROOTNAME);
      IOERR:=IORESULT;
      CLOSE(PASSFILE)
   UNTIL (IOERR<>7); (* WAIT FOR VALID FILE SPEC *)
   WRITELN;
   WRITELN('== ENTER PASS NUMBER RANGE ==');
   PASSNUM:=0;
   REPEAT
      WRITE('== ENTER FIRST NUMBER ==> ');
      READLN(PASSNUM)
   UNTIL (IORESULT=0) AND (PASSNUM>=0);
   LASTNUM:=0;
   REPEAT
      WRITE('== ENTER LAST NUMBER ===> ');
      READLN(LASTNUM)
   UNTIL (IORESULT=0) AND (LASTNUM>=0);
   INCREMENT:=0;
   REPEAT
      WRITE('== ENTER INCREMENT =====> ');
      READLN(INCREMENT)
   UNTIL (IORESULT=0) AND (INCREMENT>0);
   (*$I+*)
   XMITVSPC('TAPE');
   XMITVSPC('');
   REPEAT
      SCANACIA;
      SCANKEYBOARD;
      PASSNAME:=ROOTNAME;
      STR(PASSNUM,NUMSTRING);
      INSERT(NUMSTRING,PASSNAME,DOTPOS);
      (*$I-*) RESET(PASSFILE,PASSNAME); (*$I+*)
      IOERR:=IORESULT;
      WRITE('== ',PASSNAME,' - ');
      IF IOERR=0 THEN
```

```
         BEGIN
            WRITELN('NOW BEING SENT ==');
            SENDPASS;
            CLOSE(PASSFILE)
         END
      ELSE
         IF IOERR=10 THEN
            WRITELN('NOT FOUND ==')
         ELSE
            WRITELN(CHR(7),'OPEN ERR#',IOERR,' ==');
      PASSNUM:=PASSNUM+INCREMENT
   UNTIL PASSNUM>LASTNUM;
END; (* PASSTRANSFER *)


PROCEDURE TEXTTRANSFER;
(****************************************
 * THIS ROUTINE HANDLES TRANSFERING OF *
 * NORMAL TEXT FILES TO VSPC. THE USER *
 * IS PROMPTED FOR A FILE SPECIFICATION*
 * IN WHICH THE ".TEXT" IS OPTIONAL,   *
 * AND THEN PROCEEDS TO TRANSFER THE   *
 * FILE TO A USER-SPECIFIED VSPC WS.   *
 ****************************************)
VAR TEXTNAME,VSPCNAME:STRING;
    TEXTLINE:LONGSTRING;
    IOERR:INTEGER;
    TEXTFILE:TEXT;
BEGIN (* TEXTTRANSFER *)
   WRITELN('== PROCEDURE TO TRANSFER TEXT FILES ==');
   WRITELN;
   WRITELN('== ENTER NAME OF FILE ==');
   REPEAT
      WRITE('== FILE NAME ==> ');
      READLN(TEXTNAME);
      IF LENGTH(TEXTNAME)=0 THEN
         EXIT(TEXTTRANSFER);
      (*$I-*)
      RESET(TEXTFILE,TEXTNAME);
      IOERR:=IORESULT;
      IF IOERR=10 THEN
         BEGIN
            INSERT('.TEXT',TEXTNAME,(1+LENGTH(TEXTNAME)));
            RESET(TEXTFILE,TEXTNAME);
            IOERR:=IORESULT
         END;
      (*$I+*)
      IF IOERR<>0 THEN
         BEGIN
            IF IOERR=10 THEN
               WRITELN('FILE NOT FOUND - RE-ENTER')
            ELSE
               WRITELN('OPEN ERROR #',IOERR,' - RE-ENTER')
         END
```

```
   UNTIL IOERR=0;
   WRITELN;
   WRITELN('== ENTER NAME FOR VSPC WORKSPACE ==');
   WRITE('== WORKSPACE NAME ==> ');
   READLN(VSPCNAME);
   IF 0=LENGTH(VSPCNAME) THEN
      BEGIN
         CLOSE(TEXTFILE);
         EXIT(TEXTTRANSFER)
      END;
   XMITVSPC('TAPE');
   XMITVSPC('');
   XMITVSPC('CLEAR');
   XMITVSPC(CONCAT('NAME ',VSPCNAME));
   XMITVSPC('INPUT 1 1');
   IF NOT EOF(TEXTFILE) THEN
      REPEAT
         READLN(TEXTFILE,TEXTLINE);
         IF LENGTH(TEXTLINE)=0 THEN
            TEXTLINE:=' ';
         XMITVSPC(TEXTLINE)
      UNTIL EOF(TEXTFILE);
   XMITVSPC('');
   XMITVSPC(CONCAT('SAVE ',VSPCNAME));
   CLOSE(TEXTFILE);
END; (* TEXTTRANSFER *)


PROCEDURE PROCESSCOMMAND;
(****************************************
 * THIS ROUTINE SERVES AS A COMMON      *
 * INTERFACE BETWEEN THE MAIN PROGRAM   *
 * AND THE COMMAND-PROCESSING PROC'S.   *
 * IT'S PRESENCE IS REQUIRED IN ORDER   *
 * TO ALLOW SCANKEYBOARD TO HAVE A      *
 * COMMON EXIT POINT FOR HANDLING A     *
 * USER <CTRL¬C> COMMAND.               *
 ****************************************)
BEGIN (* PROCESSCOMMAND *)
   CASE KBCHR OF
      'Q':QUITREQUESTED:=TRUE;
      'D':DUMBTERMINAL;
      'P':PASSTRANSFER;
      'T':TEXTTRANSFER;
   END; (* CASE *)
END; (* PROCESSCOMMAND *)


(****************************************
 * THE MAIN ROUTINE (BELOW) HANDLES     *
 * GENERAL INTIALIZATION AND THE        *
 * PROMPTING FOR, AND INPUT OF, USER    *
 * COMMAND OPTIONS FROM ITS MAIN MENU.  *
 ****************************************)
```

```
BEGIN (* TALK *)
   QUITREQUESTED:=FALSE;
   PAGE(OUTPUT);
   POKE(ACIASTATUS,ACIARESET);
   POKE(ACIASTATUS,ACIASPEED);
   REPEAT
      WRITELN('== SERIAL COMMUNICATIONS PROGRAM ==');
      WRITELN;
      WRITELN('== COMMAND MODE ==');
      WRITELN;
      WRITELN('OPTIONS ARE:');
      WRITELN('    D = DUMB TERMINAL MODE');
      WRITELN('    P = TRANSFER SATELLITE PASS FILES');
      WRITELN('    T = TRANSFER ANY TEXT FILE       ');
      WRITELN('    Q = QUIT');
      WRITELN;
      WRITE  ('== ENTER COMMAND ==> ');
      REPEAT
         WRITE(CHR(7));
         READ(KEYBOARD,KBCHR)
      UNTIL KBCHR IN ['D','P','T','Q'];
      PAGE(OUTPUT);
      PROCESSCOMMAND;
      WRITELN;
      WRITELN;
   UNTIL QUITREQUESTED;
   PAGE(OUTPUT);
END. (* TALK *)
```

II.4 STARTUP PROGRAM LISTING

```
PROGRAM STARTUP;
USES APPLESTUFF;
TYPE SYSTEMDATE = PACKED RECORD
        MONTH: 1..12;    (* 4 BITS *)
        DAY:   1..31;    (* 5 BITS *)
        YEAR:  0..99     (* 7 BITS *)
     END;              (* =16 BITS TOTAL *)
VAR BLOCK:ARRAY[0..255] OF SYSTEMDATE;
    OLDBLOCK:SYSTEMDATE;
    DATEANDTIME:PACKED ARRAY[1..19] OF CHAR;
PROCEDURE READTIME(VAR PC19); EXTERNAL;
BEGIN
   (* NOW UPDATE THE "SYSTEM DATE" - STORED IN BLOCK 2 OF BOOT DISK *
   (*$R-*)
   UNITREAD(4,BLOCK,512,2);
   OLDBLOCK:=BLOCK[10];
   WITH BLOCK[10] DO
      BEGIN
         READTIME(DATEANDTIME);
         YEAR :=(ORD(DATEANDTIME[1])-48)*10+ORD(DATEANDTIME[2])-48;
         MONTH:=(ORD(DATEANDTIME[4])-48)*10+ORD(DATEANDTIME[5])-48;
         DAY  :=(ORD(DATEANDTIME[7])-48)*10+ORD(DATEANDTIME[8])-48;
      END;
   IF BLOCK[10]<>OLDBLOCK THEN
      BEGIN
         UNITWRITE(4,BLOCK,512,2);
         GOTOXY(0,10);
         WRITELN('PLEASE TYPE I(NITIALIZE TO RESET DATE');
         WRITE(CHR(7),CHR(7),CHR(7));
      END
      (*$R+*)
   ELSE
      BEGIN
         GOTOXY(0,8);
         WRITE(' * THE CURRENT ');
         NOTE(24,90);
         WRITE('DATE, ');
         NOTE(26,90);
         WRITE('DAY ');
         NOTE(22,90);
         WRITELN('AND TIME IS: *');
         NOTE(10,90);
         WRITELN;
         WRITELN;
         WRITELN('     * * * YY/MM/DD-D HH:MM:SS * * *');
         WRITELN;
         WRITELN;
         READTIME(DATEANDTIME);
```

```
            WRITELN('       * * ',DATEANDTIME,' * *');
            NOTE(17,150);
        END;
END.
```

II.5 PEEKPOKE UNIT LISTING

```
(*$S+*)

UNIT PEEKPOKE; INTRINSIC CODE 23;

INTERFACE
   FUNCTION  PEEK(ADDR:INTEGER):INTEGER;
   PROCEDURE POKE(ADDR,DATA:INTEGER);

   (************************************
    * NOTE THAT PARAMETERS "ADDR" AND *
    * "DATA" SHOULD BE INTEGER VALUES.*
    * IN THE FOLLOWING RANGES:        *
    *                                 *
    * -32767 <= ADDR <= +32767        *
    *      0 <= DATA <=   +255        *
    *                                 *
    * THE POKE FUNCTION WILL ALWAYS   *
    * CONVERT "DATA" TO  MOD 256      *
    * BEFORE IT IS POKED.             *
    ************************************)

iMplementation

   type byte  = packed array [0..1] of 0..255;
        dirty = record
                  case boolean of
                    true :(int:integer);
                    false:(ptr:¬BYTE);
                  END;


   FUNCTION PEEK;
   VAR  TRICK:DIRTY;
   BEGIN
      TRICK.INT:=ADDR;
      PEEK:=TRICK.PTR¬[0]
   END;


   PROCEDURE POKE;
   VAR TRICK:DIRTY;
   BEGIN
      DATA:=ABS(DATA MOD 256);
      TRICK.INT:=ADDR;
      TRICK.PTR¬[0]:=DATA
   END;
```

```
BEGIN
  (* DUMMY STARTUP INITIALIZATION FOR UNIT *)
END.
```

II.6 READTIME ROUTINE LISTING

```
            .TITLE "READTIME - PROCEDURE TO RETURN DATE & TIME"
            .NOMACROLIST
            .NOPATCHLIST

;   MACRO TO POP 16-BIT RETURN ADDRESS:

            .MACRO POP
            PLA
            STA     %1
            PLA
            STA     %1+1
            .ENDM

;   MACRO TO PUSH 16-BIT RETURN ADDRESS:

            .MACRO PUSH
            LDA     %1+1
            PHA
            LDA     %1
            PHA
            .ENDM


SLOTNUM     .EQU  4             ;SLOT NUMBER FOR CLOCK BOARD
CLOCK       .EQU  <10*SLOTNUM>+0C080  ;BASE ADDRESS FOR CLOCK CARD
BUFFER      .EQU  00            ;PAGE ZERO WORD FOR PARAMETER ADDRESS
RTADDR      .EQU  02            ;TEMPORARY STORAGE FOR RETURN ADDRESS

;   PROCEDURE TO READ DATE AND TIME FROM CLOCK IN SLOT #SLOTNUM:
;   PARAMETER MUST BE ADDRESS OF 19-BYTE AREA TO RECEIVE THE DATE AND
;   TIME.  OUTPUT IS FORMATTED AS FOLLOWS:  YY/MM/DD-W HH:MM:SS
;   YY=YEAR,MM=MONTH,DD=DAY,W=WEEKDAY,HH=HOUR,MM=MINUTE,SS=SECOND

            .PROC READTIME,1  ;PROC TO RETURN DATE & TIME

            POP   RTADDR      ;SAVE PASCAL RETURN ADDRESS
            POP   BUFFER      ;STORE PARAMETER ADDRESS IN PAGE ZERO

            LDA   #30         ;APPLY TIMER-HOLD AND CHIP SELECT
            STA   CLOCK+1
            LDX   #40         ;SET UP, THEN EXECUTE A 200 MSEC DELAY
DELAY       DEX
            BPL   DELAY

            LDX   #0C         ;SET INDEX TO POINT AT YEAR
            LDY   #00         ;SET BUFFER INDEX TO FIRST CHAR
NEXTDIGT    LDA   FILLCHAR,X  ;GET NEXT INSERTION CHARACTER
            BEQ   GETDIGIT    ;CHECK FOR NULL CHARACTER
```

```
          STA   @BUFFER,Y    ;SAVE IN OUTPUT BUFFER
          INY               ;INCREMENT BUFFER POINTER
GETDIGIT  TXA               ;COPY DIGIT INDEX INTO ACCUMULATOR
          ORA   #30         ;INSERT A "3" IN UPPER 4 BITS
          STA   CLOCK+1     ;REQUEST DIGIT
          LDA   CLOCK       ;GET THE DIGIT
          LDA   CLOCK       ;(INSURE PROPER TIMING BY
          LDA   CLOCK       ;  DOING THREE CHIP READS)
          AND   BCDMASK,X   ;GET RID OF EXCESS BITS
          ORA   #30         ;CONVERT TO ASCII BCD DIGIT
          STA   @BUFFER,Y   ;SAVE IN BUFFER
          INY               ;INCREMENT BUFFER POINTER
          DEX               ;DECREMENT DIGIT INDEX
          BPL   NEXTDIGT    ;LOOP BACK FOR NEXT DIGIT

          LDA   #2F         ;GET CODE TO RELEASE HOLD
          STA   CLOCK+1     ;ISSUE NOHOLD REQUEST

          PUSH  RTADDR      ;GET PASCAL RETURN ADDRESS
          RTS               ;RETURN TO CALLING PROGRAM

;  THE FOLLOWING BIT PATTERNS ARE "ANDED" WITH EACH BCD DIGIT
BCDMASK   .BYTE 0F,0F,0F,07,0F,03,07,0F,03,0F,01,0F,0F
;  THE FOLLOWING (NON-ZERO ONLY) BYTES ARE INSERTED BEFORE DIGITS
FILLCHAR  .BYTE 00,3A,00,3A,00,20,2D,00,2F,00,2F,00,00

          .END
```

BIBLIOGRAPHY

R6500 Microcomputer System Programming Manual. Anaheim, California: Rockwell International, 1979.

Apple II Communications Interface Card: Installation and Operating Manual. Cupertino, California: Apple Computer Inc., 1978.

Owner's Manual for Model 7720 Parallel Interface. Sunnyvale, California: California Computer Systems, 1980.

Owner's Manual for Model 7424 Calendar/Clock Module. Sunnyvale, California: California Computer Systems, 1980.

Owner's and Operators Guide Chicago, IL: Bell & Howell Audio Visual Products Division, 1981.

Apple Pascal Language Reference Manual. Cupertino, California: Apple Computer Inc., 1980.

Apple Pascal Operating System Reference Manual. Cupertino, California: Apple Computer Inc., 1980.

Apple II Reference Manual. Cupertino, California: Apple Computer Inc., 1981.

Stansell, Thomas A. The TRANSIT Navigation Satellite System. Torrance, California: Magnavox Advanced Products Division, 1978.

Welsh, Jim, & Elder, John; Hoare, C. A. R. (Editor). Introduction to PASCAL. London: Prentice-Hall International Inc., 1979.

Tonkins, Ross M. A Guided Tour of Apple PASCAL UNIT's and Libraries. California: BYTE Publications Inc., Feb/1982.