PcPI ser 974
BOOTSTRAP 9.0

CLIP 13 FROM TOP

Jack Feder
12 Saddletree Dr
Willowdale, ON
M2H 3L3

APPLI-CARD OEM MANUAL

Programmers or OEMs requiring revision information should contact PCPI at the address below.

Personal Computer Products, Inc.
16776 Bernardo Center Drive
San Diego, CA 92128

TABLE OF CONTENTS

INTRODUCTORY NOTE

Throughout this manual, references are made to various files (such as RDWRHST.ASM). These files may be found on the supplied diskette (Chapter 9 contains a list of them). This manual is designed to be used in conjunction with these annotated source files. It is suggested that the reader first list these files and refer to them as necessary when reading this manual. Assembled list files of Apple resident programs (as opposed to printouts of the source files) may be obtained via the supplied 6502 cross assembler (see Chapter 6). The APPLI-CARD resident programs may be assembled with the standard CP/M assembler (ASM or MAC).

CHAPTER 1/ BIOS INFORMATION

CP/M consists of 3 sections:

- Console Command Processor(CCP)
- Basic Disk Operating System (BDOS)
- Basic Input Output System (BIOS)

Of these modules, only the BIOS is hardware dependent. The APPLI-CARD BIOS uses the Apple computer as its I/O processor and sends commands over a communication port to direct the Apple.

The BIOS is hardware dependent code written in assembly language. This code translates calls, by both the BDOS and from application programs, to commands for use by the Apple. When it is necessary that a program communicate with the Apple, subroutines in the RDWREST.ASM file may be used. The Apple and APPLI-CARD communicate over a parallel port. This allows both the APPLI-CARD and the Apple to run at the same time and not intrude upon each others address space. This also allows the APPLI-CARD BIOS to be relatively small, because the direct manipulation of the I/O devices is done in the Apple.

The BIOS has been arranged to support up to 16 mass storage devices as well as 16 character devices. The mass storage devices may be of multiple densities. These are referenced as drives A: through P:. With the use of the INSTALL program, these devices are automatically allocated check vector space, allocation vector space, and host buffers. The user must ensure that the driver is in the proper format. This format is detailed in the BDEXMPL.A65 file. The character devices may be accessed by changing the I/O byte to use the first five devices; the remaining 11 devices must be accessed directly from the application code by using the RDWREST.ASM file. An example character device driver is CDEXMPL.A65. Please refer to it for details on how to write a character device driver.

CHAPTER 2/ APPLE I/O PROCESSOR COMMAND STRUCTURE

2.1 INTRODUCTION

As mentioned previously, the Apple and the APPLI-CARD communicate over a parallel port. The APPLI-CARD sends commands to the Apple and the Apple performs the commands, then waits for the next command. There are 256 possible commands which the APPLI-CARD may send to the Apple. They are divided into 2 Classes:

- 0 through 127 (general commands)
- 128 through 255 (device specific commands)

Each command is followed by parameters with the number of parameters being dependent on the command. The parameters are simply sent following the command byte. For example, the read data command (1) requires the starting address and the number of bytes as parameters.

The following assembly language program segment demonstrates this (the subroutines used are in the RDWRHST.ASM file):

```
LD   C,1          ;C = READ DATA COMMAND
CALL WHBYTE       ; AND SEND IT
LD   DE,1000H     ;DE = STARTING ADDRESS
CALL WHWORD       ; AND SEND IT
LD  DE,123H       ;DE = NUMBER OF BYTES
                    TO RECEIVE
CALL WHWORD       ; AND SEND IT
LD   HL,200H      ;HL = LOCATION TO STORE
LD   DE,123H      ;DE = NUMBER OF BYTES
CALL RHBYTS       ; GO READ THEM
```

Subroutines WHBYTE, WHWORD, and RHBYTS are described fully in the RDWRHST.ASM file. Briefly, WHBYTE rewrites one byte from the APPLI-CARD to the Host processor (the Apple). WHWORD writes two bytes from the host, and RHBYTS reads multiple bytes from the host.

2.2 GENERAL COMMANDS

Currently, 8 of the 128 possible general commands are supported. These are the first 8 commands (0 through 7). The next 8 (9 through 15) are reserved for the system and must not be used. The remaining commands are undefined.

The following is a description of the 8 general commands:

Notes:
- All 2 byte parameters are sent low byte followed by high byte.
- Parameters are listed in the order sent to the Apple.

COMMAND	DESCRIPTION
0	Function: Illegal command (does nothing). Parameters: None. Results: None.
1	Function: Read data from the Apple memory. Comments: This command reads data from the Apple memory and returns it to the APPLI-CARD. Parameters: Starting address (2 bytes). Number of bytes (2 bytes). Results: Returns the requested number of bytes to the APPLI-CARD.
2	Function: Write data to the Apple memory. Comments: This command writes data to the Apple memory. Parameters: Starting address (2 bytes). Number of bytes (2 bytes). Data (The number of bytes is the value passed above) Results: None.
3	Function: Execute a routine. Comments: This command causes the Apple to send a routine at the address passed. The routine must end with a RET. Parameters: Starting address (2 bytes). Results: None.

COMMAND	DESCRIPTION
4	Function: Warm boot operating system. Comments: This routine is used by the warm boot code. It should not be used by other programs. Parameters: None. Results: The operating system is sent to the APPLI-CARD.
5	Function: Connect a device driver. Comments: This is used by DLDRVR and at boot time by DRIVERS. Device drivers are 6502 programs that exist in CP/M files in a relocatable format. They are read into the APPLI-CARD memory via the normal CP/M commands and then are sent to the Apple where they are included in the executing software. General command 5 is used to download these drivers from APPLI-CARD memo to Apple memory. Also refer to the example drivers on the structure of a driver. See DLDRVR.ASM for details. Parameters: Load address (2 bytes, 0 = relocatable). Length of driver (2 bytes). Length of Page 0 (1 byte). Tag field (1 byte, currently 0). Device number (2 bytes (upper byte is 0)). Number of devices (2 bytes (upper byte is 0)). Address of initialize entry point (2 bytes). Address of read entry point (2 bytes). Address of write entry points (2 bytes). Address of other entry point (2 bytes). Address of polling entry point (2 bytes). Driver code (As indicated by length of driver). Relocation bit map (None if load address is 0, else is length of driver/8). Page 0 relocation bit map, (None if length of page 0 is 0 else is length of driver/8).

COMMAND	DESCRIPTION
6	Function: Read 1 byte from the Apple memory. Comments: This routine reads a byte of data using 6502 direct addressing, instead of 6502 indirect addressing used in commands 1 and 2. Parameters: Address to read (2 bytes). Results: Returns one byte of data.
7	Function: Write 1 byte of data to the Apple memory. Comments: This routine writes a byte of data using 6502 direct addressing, instead of 6502 indirect addressing used in commands 1 and 2. Parameters: Address to write (2 bytes). Data to write (1 byte). Results: None.
8 through 15	Reserved for system use.
16 through 127	Undefined.

DEVICE COMMANDS

The 128 device commands allow direct control of 32 devices, which are defined in the Apple. The 32 devices are divided into two groups. Sixteen block devices are used to control mass storage devices (such as floppy disks), and 16 character devices (such as the console and printers). Each device has four commands: INITIALIZE, READ, WRITE, and OTHER. OTHER is used to define 256 more commands. The first 16 of these are reserved; the remaining 240 are available for use.

The following describes the way the device command bytes are encoded:

Bits 0 and 1: Defines which of the four commands to use with the device.

0 = INITIALIZE
1 = READ
2 = WRITE
3 = OTHER

Bits 2 through 5: Define which of the 16 devices to use.

0 = Device 0
1 = Device 1
.
.
15 = Device 15

Bit 6: Defines either Block or Character mode.

0 = Block mode
1 = Character mode

Bit7: Defines General or Device command.

0 = General command
1 = Device command

2.3.1 BLOCK DEVICE COMMANDS

Thissection describes how thefour commands are interpreted for block devices. See BDEXMPL.A65 for an example of a block device driver.

COMMAND	DESCRIPTION
INITIALIZE	Function: Initailize the driver. Comments: This command causes a call to the INITIALIZE entry point in the appropriate driver by the Apple processor. It is automatically called when the driver is connected (General command 5). Parameters: None. Results: Error code (1 bytes; if no errors then error code = 0, else error code <> 0).

COMMAND	DESCRIPTION
READ	Function: Read one sector of data. Comments: This command causes a call to the READ entry point in the appropriate drivers by the Apple processor. Parameters: Sector size (2 bytes). Drive number (1 byte). Track number (2 bytes). Sector number (2 bytes). Results: Sector data (sector size bytes). Error code (1 byte, If no errors then error code = 0, else error code <> 0).
WRITE	Function: Write one sector of data. Comments: This commands causes a call to the WRITE entry point in the appropriate driver by the Apple processor. Parameters: Sector size (2 bytes). Drive number (1 byte). Track number (2 bytes). Sector number (2 bytes). Sector data (sector size bytes). Results: Error code (1 byte, If no errors then error code = 0, else error code <> 0).
OTHER	Defines 256 other commands to a block device. The first 16 are reserved.
0	Function: Send disk parameter command. Comments: This command is used by the BIOS to support multiple density disks. Each time a disk is used for the first time, this command is sent to log in the drive. The driver then determines the type of disk and sends up the appropriate disk parameters. These tables are usually created at assembly time. The data is determined by using the file DISKDATA.ASM file supplied on the disk.

COMMAND	DESCRIPTION
	Parameters: None. Results: Sector size (2 bytes). CP/M records per track (2 bytes). CP/M records per host block (1 byte). CP/M records per allocation block (1 byte). Sector mask (1 byte). Sector shift count (1 byte). CP/M records per track (2 bytes). Block shift factor (1 byte). Block shift mask (1 byte). Disk size - 1 in allocation blocks (2 bytes). Directory size - 1 (2 bytes). Allocation mask (2 bytes). Check size (2 bytes). Offset to first CP/M directory track (2 bytes). Translation table (2 bytes should be 0). Error code (1 byte if no errors then error code = 0, else error code <> 0).
1	Function: Format command. Comments: Formats the entire block device. Parameters: Drive number (1 byte). Results: Error code (1 byte, if no errors then error code = 0, else error code <> 0).
2 through 14	Reserved.
15	Function: Send name of driver. Comments: This command is used to get the name of the driver. Since this command was added after the initial release of the software, not all drivers support this command. If you wish to use the GETNAME subroutine, refer to RDWREST.ASM. Parameters: None.

COMMAND	DESCRIPTION
	Results: Length of name (1 byte max value of 15). Name characters (byte length of name). Error code (1 byte of 0).

.2 CHARACTER DEVICE COMMANDS

This section describes howthe four commands are interpreted for character devices. See CDEXMPL.A65 for an example of a character device driver.

COMMAND	DESCRIPTION
INITIALIZE	Function: Initialize the driver. Comments: This command causes a call to the INITIALIZE entry point of the appropriate driver by the Apple processor. It is automatically called when the driver is connected. Parameters: None. Results: 0 = No errors, else error in initialization.
READ	Function: Read a byte. Comments: This command causes a call to the READ entry point of the appropriate driver by the Apple processor. Parameters: None. Results: Character read (1 byte).
WRITE	Function: Write a byte. Comments: This command causes a call to the WRITE entry point of the appropriate driver by the Apple processor. Parameters: Character to write (1 byte). Results: None.

COMMAND	DESCRIPTION
OTHER	256 other commands to a character device may be used.
0	Function: Output status. Returns the status of the character output routine. Parameters: None. Results: Status code (1 byte, if ready to send then status code <> 0, else status code = 0).
1	Function: Input status. Returns the status of the character input routine. Parameters: None. Results: Status code (1 byte, if character is ready then status code <> 0, else status code = 0).
2	Function: Turns on the video for this device. Comments: This is used for switching from the Apple video to the console video. Parameters: None. Result: Status code (1 byte, ignore).
3	Function: Turn on the Apple video. Comments: This used used for switching from the console video to the Apple video. Parameters: None. Results: Status code (1 byte, ignore).
4	Function: Returnthe width of this device. Comments: This is used to determine the display width of a device. Parameters: None. Results: Return the width of the this device (1

COMMAND	DESCRIPTION
5 through 14	Reserved
15	Function: Send name of driver. Comments: This command is used to get the name of the driver. Since this command was added after the initial release of the software, not all drivers support this command. To use this function, use the GETNAME subroutine in RDWRHST.ASM. Parameters: None. Results: Length of name (1 byte max of 15). Name characters (byte length of name). Error code (1 byte of 0).

.4 COMMAND MAP

hex|decimal COMMAND	DESCRIPTION
00	Illegal does nothing
01	Read data from the Apple memory
02	Write data to the Apple memory
03	Execute a routine
04	Warm boot
05	Connect a driver
06	Read 1 byte from the Apple memory
07	Write 1 byte to the Apple memory
08..0F	Reserved
10..7F	Unused
80..83	Block device 0
84..87	Block device 1
88..8B	Block device 2
8C..8F	Block device 3
90..93	Block device 4
94..97	Block device 5
98..9B	Block device 6
9C..9F	Block device 7
A0..A3	Block device 8
A4..A7	Block device 9
A8..AB	Block device 10
AC..AF	Block device 11
B0..B3	Block device 12
B4..B7	Block device 13
B8..BB	Block device 14
BC..BF	Block device 15
C0..C3	Character device 0
C4..C7	Character device 1
C8..CB	Character device 2
CC..CF	Character device 3
D0..D3	Character device 4
D4..D7	Character device 5
D8..DB	Character device 6
DC..DF	Character device 7
E0..E3	Character device 8
E4..E7	Character device 9
E8..EB	Character device 10
EC..EF	Character device 11
F0..F3	Character device 12
F4..F7	Character device 13
F8..FB	Character device 14
FC..FF	Character device 15

CHAPTER 3/ I/O BYTE

This section describes the use of the I/O byte by the BIOS. The I/O byte is located at address 0003h, and is changed either by the STAT program or directly by an application program. It should be noted that although the I/O byte only supports 5 different devices, the command structure supports 16 devices. The extra devices are available by directly calling the Apple I/O processor.

The following shows the relationship between the logical devices (CON:,RDR:,PUN:,LST:) and the physical device names.

```
bits   7 6    5 4    3 2    1 0
       ----------------------------
       ! LST: ! PUN: ! RDR: ! CON: !
       ----------------------------
```

CONSOLE, CON: = bits 0 and 1.
- 0 = TTY: for input and output (character device 0)
- 1 = CRT: for input and output (character device 3)
- 2 = BAT: causes CRT: to be used for input and output. In addition, all output is echoed to LST:. This is equivalent to a permanent "CTRL P".
- 3 = UC1: for input and output (character device 4)

READER, RDR: = bits 2 and 3.
- 0 = TTY: as input (character device 0)
- 1 = PTR: as input (character device 2)
- 2 = UR1: as input (character device 1)
- 3 = UR2: as input (character device 4)

PUNCH, PUN: = bits 4 and 5.
- 0 = TTY: as output (character device 0)
- 1 = PTP: as output (character device 2)
- 2 = UP1: as output (character device 1)
- 3 = UP2: as output (character device 4)

PRINTER, LST: = bits 6 and 7.
- 0 = TTY: for input and output (character device 0)
- 1 = CRT: for input and output (character device 3)
- 2 = LPT: for input and output (character device 1)
- 3 = UL1: for input and output (character device 4)

CHAPTER 4/ HARDWARE

The following chapter outlines the Apple and APPLI-CARD hardware.

4.1 APPLE HARDWARE

In the hex numbers that follow, the letter "N" (such as 0C0N0) is defined as 8 plus the Apple slot number in which the APPLI-CARD resides. For example, 0C0C0H is the input port of the APPLI-CARD when it is in slot 4.

I/O PORT	DESCRIPTION
0C0N0	8 bit input port. This is the port the Z-80 writes to send a byte of data to the 6502.
0C0N1	8 bit output port. This is the port the Z-80 reads to get a byte of data from the 6502.
0C0N2	1 bit status port. Bit 7 of this port is set to 1 if output data port (0C0N1) is full; i.e., the previous data byte written by the 6502 has not yet been read by the Z-80.
0C0N3	1 bit status port. Bit 7 of this port is set to 1 if the input data port (0C0N0) is full; i.e., there is data for the 6502 to read from the Z-80.
0C0N4	Not used.
0C0N5	Resets the Z-80. This causes the Z-80 to begin execution at location 0. A read or write to this location resets the Z-80.
0C0N6	Interrupt the Z-80. This is connected to channel 3 of the CTC on the APPLI-CARD. If this channel is set up properly, a read or write to this port will interrupt the Z-80. Currently, the CTC is initialized with interrupts disabled.
0C0N7	NMI interrupt for the Z-80. This is connected to the Z-80 NMI pin and causes the Z-80 to take an interrupt and begin execution at 66H. A read or write to this location causes an NMI.

Example of 6502 code to communicate with the Z-80:

```
RDBYTE:
   LDX  SLOTNUM          ;GET Z80 SLOTNUMBER
WAIT1:
   LDA  0C083,X          ;GET INPUT STATUS BIT
   BPL  WAIT1            ;WAIT FOR DATA
   LDA  0C080,X          ;READ THE DATA
   RTS
WRBYTE:
   TAY                   ;SAVE THE DATA
   LDX  SLOTNUM
WAIT2:
   LDA  0C082,X
   BMI   WAIT2           ;WAIT FOR PREVIOUS DATA TO BE
                          ;READ
   TYA                   ;GET BYTE
   STA  0C081,X          ;WRITE THE DATA
   RTS
```

APPLI-CARD HARDWARE

The APPLI-CARD contains the following features:

- 64K of memory.
- 4 MHZ or 6 MHZ Z-80.
- 2K ROM.
- Z-80 CTC.
- Apple parallel port interface circuit.

The following is a description of each of the I/O ports in the APPLI-CARD.

I/O PORT	DESCRIPTION
0	8 bit Output port to 6502. Data written to this port is buffered on the APPLI-CARD for the 6502. The data available flag at DC0N3 is set when this port is written by the Z-80.
20H	8 bit input port from 6502. The 6502 data out is buffered on the APPLI-CARD and is read into the Z-80 by an input to this port. Data in this port is valid when the Z-80 data available at bit 7 of the input port 40H is set.
40H	2 bit status port. Bit 7 is the input port data available flag and is a 1 when valid data is present. Bit 0 is the output port data flag and is a 1 when data has not yet

I/O PORT	DESCRIPTION
60H	1 bit shadow memory control output port. When a 0 is written to bit 0, the ROM is turned off and the lower 32K RAM is active. When a 1 is written to bit 0, the ROM is turned on and the lower 32K of RAM is inactive. The ROM is turned on at power up, RESET, and RESET command via the APPLE interface (0CON5). Note that when the ROM is enabled, it occupies memory from 0 through 07FFFH. Therefore, code which turns the ROM on/off must reside in the upper 32K of memory (8000 through 0FFFF).
80H	CTC CHANNEL 0 PORT
81H	CTC CHANNEL 1 PORT
82H	CTC CHANNEL 2 PORT
83H	CTC CHANNEL 3 PORT

Note: The control codes for the CTC are contained in ZILOG and MOSTEK publications.

Examples of Z-80 code to communicate with the Apple:

```
RDBYTE:
   IN    40H              ;GET STATUS BIT
   RLC                    ;SHIFT STATUS BIT TO CARRY
   JNC   RDBYTE           ;JUMP IF NO DATA
   IN    20H              ;READ THE DATA
   RET

WRBYTE:
   IN    40H              ;GET STATUS BIT
   RRC                    ;SHIFT STATUS BIT TO CARRY
   JC    WRBYTE           ;JUMP IF OLD DATA STILL THERE
   MOV   A,C              ;GET DATA FROM C
   OUT   0H               ;WRITE THE DATA
   RET

SHADOW$OUT$ROM:
   XRA   A                ;A = 0
   OUT   60H              ;TURN OFF ROM
   RET

TURN$ON$ROM:
   MVI   A,1              ;A = 1
   OUT   60H              ;TURN ON ROM
   RET
```

APPLI-CARD FIRMWARE

After power up or a Z-80 reset (0C0N5), the ROM on the APPLI-CARD gains control. At such time, the Z-80 writes an ASCII Z (05AH) to the 6502 input port at 0C0N0. The first thing an Apple program must do is read this data from the 0C0N0 port. After that, the following four commands may be used to control the APPLI-CARD.

Note: all 2 byte parameters are sent low byte followed by high byte.

COMMAND	DESCRIPTION
0	Function: Return the ID string "Z80",ROM VER, SERIAL NUMBER. Parameters: None. Results: "Z80" (3 bytes -- 5AH,38H,30H). ROM VERSION (1 BYTE binary value). SERIAL NUMBER (4 BYTES binary value, low byte first).
1	Function: Read data from the APPLI-CARD memory. Parameters: Starting address (2 bytes). Number of bytes (2 bytes). Results: Data returned (number of bytes).
2	Function: Write data to the APPLI-CARD memory. Parameters: Starting address (2 bytes). Number of bytes (2 bytes). Data bytes (number of bytes).
3	Function: Call a program. Parameters: Starting address (2 bytes). Results: None.

Note: In version 9.0 of the APPLI-CARD, the following commands have been added:

4	Function: Write one byte to APPLI-CARD I/O port. Parameters: I/O port address (1 byte). Byte to output (1 byte). Results:

COMMAND	DESCRIPTION
5	Function: Read one byte from APPLI-CARD I/O port. Parameters: I/O port address (1 byte). Results: Data byte read (1 byte).

While the firmware is running (i.e. the shadow ROM is on), the Z-80 will wait for approximately two minutes for the first command or between subsequent commands from the Apple. If no command is received during that time, the Z-80 will cease looking for commands and start diagnostic testing. At this time, it will write a binary 0 to the Apple data port (output port 0) and increment this value approximately four times a second. The continual incrementing of this value indicates the diagnostics are running properly. The diagnostic program may be interrupted and communications with the Apple restored by resetting the APPLI-CARD (access ti 0C0N5).

CHAPTER 5/ DEBUGGING DRIVERS

INTRODUCTION

Debugging drivers with the APPLI-CARD is complicated by the real time communication between the two computers, and the fact there is one only debugging console and one communications port for the two machines.

As stated in the source code, never attempt to trace through a routine which communicates with the Apple from a Z-80 debugger. Since both the debugger and your code will be trying to talk with the Apple at the same time, the system will invariably hang. The only way to debug communication problems is to do it from the Apple side.

The first thing you must be able to do is set a break point in a driver. To set a break point, you need to get to the Apple monitor. If you have an INTEGER BASIC Apple, then all you have to do is press reset. If you do not have an INTEGER BASIC Apple, you will need to patch the command processor program while the APPLI-CARD is booting (see RESET PATCH below). Any time the APPLI-CARD is waiting for a character to be typed, you may press reset, write a character to 0C0N1 (the 6502 output port), and then reenter the command processor at 0B038H.

The following example assumes that the APPLI-CARD is in slot 5. Be certain to change the I/O addresses if APPLI-CARD is in a different slot.

```
A>              {PRESS APPLE RESET WHEN THE PROMPT IS
                  DISPLAYED}

*               {SET ANY BREAK POINTS}

*C0D1:41        {SUPPLY AN "A" AS THE CHARACTER TYPED}

*B038G          {REENTER THE COMMAND LOOP}

A>A             {WE ARE NOW BACK INTO CP/M WITH THE "A"}
```

It is necessary to know where your code is loaded. The drive code is loaded sequentially beginning at 0B00H. SFTVIDEO.DVR ends at 3600H, and HIRESIO.DVR ends at 7000H. Begin looking for your code after these points.

,.2 RESET PATCH

If you do not have an INTEGER BASIC machine, you may patch the Apple command processor loop at 0B22FH with 65H, and 0B230H with 0FFH. To perform this patch , hit reset after the screen has cleared and before the LOADING DRIVERS message appears. Then patch the above locations and continue the boot by executing at 0B000H.

CHAPTER 6/ ASSEMBLER

INTRODUCTION

The 6502 Cross Assembler A65 was designed to translate 6502 assembly source files to a relocatable format. It runs under CP/M on the APPLI-CARD and uses CP/M files. The output of the assembler is linked with other modules to form an object file. It is not necessary to use this assembler and linker, of course, but they are provided if you wish to use them. The sample files use this assembler.

USER INTERFACE

To execute the assembler, the following files must be present on the default drive: A65.COM, MAIN65.OVL, DMPTABLE.OVL and OPCDE65.TXT,. Type A65 and follow the prompts noted below:

```
A>A65
Input file name: B:CDEXMPL
Output file name: B:CDEXMPL0
List file name (cr or NONE: for none): NONE:
```

The input source file name is a standard CP/M file name. If an extent is not present, then .A65 is assumed.

Examples of input file names:

```
SVAZVX4              {SVAZVX4.A65 on default disk}
B:SVAZVX4            {SVAZVX4.A65 on drive B}
C:SVAZVX4.ABC        {SVAZVX4.ABC on drive C}
```

The output file name is the file to which the relocatable output is written. If a dollar sign ($) is used in place of an output file, then the name of the output file is the same as the input file name, except with an extent of .ERL. If no extent is provided, then .ERL is supplied.

Examples of output file names:

```
$                {Same file name as input but .ERL}
CDEXMPL          {CDEXMPL.ERL on default drive}
B:CDEXMPL.OUT    {CDEXMPL.OUT on drive B}
```

The list file name is the file to which the listing is written. If a dollar sign ($) is used in place of a list file, then the name of the list file is the same as the input file name, except with an extent of .PRN. If you do not want a listing file, then enter a return or NONE:. Other legal names are LST: for the printer, and CON: for the console.

Examples of listing file names:

```
$                {Same file name as input but .PRN}
BDEXMPL          {BDEXMPL.PRN on default drive}
G:BDEXMPL.LST    {BDEXMPL.LST on drive G}
LST:             {listing to printer}
CON:             {listing to console}
NONE:            {No listing}
```

6.3 SOURCE SYNTAX

Source files consist of lines of ASCII characters terminated with a carriage return and line feed. The format of each line is:

label opcode operand ;comment

where any or all of the fields are optional.

6.3.1 LABELS

The label field is a string of characters which begin in column 1, and start with an alphabetic character ('@', 'A' to 'Z' or dollar sign '$'), followed by any number of alphanumeric characters. Only the first 8 characters are significant. If the label begins with a dollar sign, it is a temporary label which is active between two non-temporary labels.

6.3.2 OPCODE

The opcode field may be one of the MOS Technology 6502 opcodes or a pseudo opcode.

6.3.3 OPERAND

The operand field is optional or mandatory depending on the opcode. Expressions in the operand field may be of two types: either numerics or strings. String expressions are multiple character strings imbedded between two double quotes. Examples:

```
.BYTE    "THIS IS A STRING"
.BYTE    "AB"                    ;A SHORT STRING
```

Numeric expressions may consist of numbers, a single character in double quotes, or a label with optional operators and parenthesis. Operators of the same precedence are evaluated left to right. The following are the operators in order of their precedence:

PREC.	OPERATOR	EXPLANATION
1)	()	Parentheses
2)		Unary operators
	+	positive
	-	negative (twos compliment)
	NOT	bit-wise ones compliment
3)		Multiplicative operators
	*	signed multiplication
	/	signed division
	%	signed modulo (remainder)
	U*	unsigned multiplication
	U/	unsigned division
	U%	unsigned modulo
4)		Additive binary operators
	+	add
	-	subtract
	U+	unsigned add
	U-	unsigned subtract
	AND	bit-wise and
	EOR	bit-wise exclusive-or
5)		Comparisons
	=	equal
	<>	not equal
	>	greater than
	<	less than
	>=	greater than or equal
	<=	less than or equal
	U=	unsigned equal
	U<>	unsigned not equal
	U>	unsigned greater than
	U<	unsigned less than
	U>=	unsigned greater than or equal to
	U<=	unsigned less than or equal to

.4 NUMERIC CONSTANTS

Numeric constants in expressions may be one of four radices: binary, octal, decimal or hexadecimal, with the radix as a suffix to the constant. When no radix is specified, the default radix, decimal, is used. The following are examples of each type.

```
256.    ;decimal the default
100H    ;hex 256.
100B    ;binary 4.
100Q    ;octal 64.
```

6.4 PSEUDO OPERATION

Below is a list of pseudo ops for the assembler.
Note: Braced items are optional.

.BLOCK expression

Reserves expression number of bytes in the program.

.BYTE expression{,expression}

Defines bytes and strings in a program.

.DEF name{,name}

Defines a list of labels as entry points into this module. Each name in the list must correspond to a label and not an equate. There may be any number of .DEF statements.

.DSECT

Tells the assembler to assemble all of the generated code into the data segment. Normally this segment is used for data areas. At link time, the address of .DSECT is set with the /D switch, or defaulted to the area following each module if no /D is used.

.EQU expression

Defines a label with a specific value.

.IF expression
{.ELSE}
.ENDC

.IF, .ELSE, and .ENDC are used for conditional assembly. If the expression evaluates to something other than 0, then the code following the .IF is assembled. If the expression evaluates to 0, and if an .ELSE is present, then the code assembled is the code between the .ELSE and the matching .ENDC. IF statements may be nested.

.INCLUDE file name

Causes the source code from the specified diskette file to be included in the main source code (as if it were written inline).

.LINES expression

Defines the number of lines per page. Default is 66.

.LIST

Directs the assembled code to the listing file.

.NOLIST

Discontinues output of the assembled code to the listing file.

.NAME "string"

Defines the module name for the linker (may be up to seven characters long).

.PAGE {"string"}

Advances to the top of the next page and allows a subtitle as an option.

.PSECT

Tells the assembler to assemble the code into the program segment. This is the default if neither .DSECT or .PSECT is invoked. Normally this is the segment used for executable code. At link time the address for .PSECT is set with the /P switch or defaulted to 100E if no /P is used.

.QUERY "string"

Queries the operator for input after displaying the string. This accepts the operator's input and assigns the value in exactly the same manner as .EQU.

.REF name{,name}

Specifies a list of labels which are defined in another module as .DEF's and which are to be accessed by this module. Note: currently no arithmetic is allowed on .REF labels (the assembler does not generate an error).

.TITLE "string"

String is printed at the top of every page.

.WIDTH expression

Defines the width of the output. Default is 79.

.WORD expression{,expression}

Defines words in LOW byte HIGH byte order.

.5 BUGS IN THE ASSEMBLER

At the present time, the assembler has several bugs which are described below.

1) When a label defined in a .REF list is used in an arithmetic expression, an error occurs. Currently, the assembler does not flag this error and the linker does not relocate it properly. If you need to do calculations they must be done at runtime. Typically this condition arises when a 2 byte variable needs to be accessed. You must either define 2 labels or use indexed addressing to access the other byte such as:

```
.REF LOLBL1, HILBL1
LDA  LOLBL1          ;GET LOW BYTE
LDY  HILBL1          ;GET HIGH BYTE
```

or

```
LDX  #1
LDA  LOLBL1,X        ;GET HIGH BYTE
TAY                  ; TO Y
LDA  LOLBL1          ;GET LOW BYTE TO A
```

2) Too many local labels cause the assembler to die. This is the only known bug that causes the assembler to stop while assembling code. If the assembler does stop for no apparent reason, try changing some of the local labels to global labels. This happens only on large files.

3) The assembler also dies while printing out a large symbol table. This also occurs only on large files. The user may ignore it, or break the file into smaller modules.

4) The assembler generates an error when doing an expression, such as:

```
CONST          .EQU 3
               LDA  LABEL+CONST+1
LABEL:         NOP
```

solution:

```
CONST1         .EQU 3+1
               LDA  LABEL+CONST1
LABEL:
```

CHAPTER 7/ LINKER

The linker is used to convert the output of the assembler into an object file. The linker can also be used to combine or link several separately assembled modules into a single file, and resolve the references between them. These references are defined by .DEF's and .REF's in the assembly language source files.

USER INTERFACE

To use the linker, you must have the two files, LINK.COM and LINK.OVL on the default drive. The linker is invoked by typing A>LINK. The input string for the linker is defined as follows:

{output file=}input file1, input file2, ... input fileN {switches}

The output file is optional. If not present, the output file will be the same as the first input file, except with an extent of .COM. Only the first input file needs an extent. If not present, the default extent, .ERL, is used. Switches are used to control actions by the linker. Each of the switches is preceded by a slash, (/), followed by a letter, and (optionally), a parameter.

LINKER	OPTION SWITCHES
/A	/A causes the linker to abort operation without any processing.
/C	/C signals the linker that more input files are found on the next line. This is used when more input files are used than can fit on one line. The last line has no /C.
/D:value	/D sets beginning address of the data segment, .DSECT. The value parameter is a hexadecimal number. If /D is not used, the default is the data follows the code of each module. There may only be one /D switch.
/E	/E causes the linker to output symbols of .REF's and .DEF's which begin with an "@". Symbols which begin with a "@" may be used in system library modules, and are not normally used in a program. This eliminates some possibilities of multiply defined symbols.

/L — /L causes the linker to output the program and data area locations for each module.

/M — /M causes the linker to output the symbol table of the .REF's and .DEF's.

/N — /N causes the linker to refrain from writing the data area to the output file. Space on the disk is saved if the data area does not contain any initialized data.

/F — /F causes the linker to assume that the file name preceding this switch is a file which contains linker input commands. This can be used instead of a submit file. .CMD is assumed if the file does not have an extent.

/P:value — /P sets the address of the program segment .PSECT. The value parameter is a hexadecimal number. If /P is not used, then the default is 100H. There may only be one /P switch.

/S — /S signals the linker to search the file associated with this switch and uses it only if there is a reference to it which is currently undefined. This feature is used mainly with libraries and system modules so unnecessary routines will not be loaded.

/X — /X allows this linker to assemble a 68000 file. It should not be used.

Examples of using the linker:

Link CDEXMPL.ERL at 100H. The output file will be CDEXMPL.COM.

```
A>LINK CDEXMPL
```

Link BDEXMPL.ERL at 0. The the output file will be BDEXMPL0.COM.

```
A>LINK BDEXMPL0=BDEXMPL/P:0
```

Link TEST.REL at 7000H with the data area to 800H, and show the complete symbol table. The output file will be TEST.OBJ on drive A:. Note that TEST.REL will be on drive B:.

```
A>LINK TEST.OBJ=B:TEST.REL,LIBRARY/P:7000/D:800/L/M/N
```

Link TEST1 through 10.

```
A>LINK TEST1,TEST2,TEST3,TEST4/C
*TEST5,TEST6/C
*TEST7,TEST8,TEST9,TEST10
```

Link TEST1 through 10 with a .CMD file. TEST.CMD looks like this:

```
TEST1,TEST2,TEST3,TEST4,TEST5/C
TEST6,TEST7,TEST8,TEST9,TEST10/P:5000
  /D:100/L/M/E
```

```
A>LINK TEST.CMD/F
```

BUGS IN THE LINKER

The only known bug in the linker is that the output file tends to be one sector larger than necessary. This can cause problems when concatenating files for a DRIVER. The solution is to rely on the size of the module defined in the assembler and not by the size of the file output by the linker.

CHAPTER 8/ GENMAP

This file generates a bit map by comparing two input files and outputting the bitmap.

8.1 USER INTERFACE

The user supplies two input files on the command line. A single file name (using the name of the first file) is created with an extent of .MAP.

Example:

```
A>GENBMAP FILE1.COM FILE2.COM
```

The output will be FILE1.MAP

8.2 BUGS IN THE GENMAP

As in the linker, GENMAP tends to output a file one sector too large. The actual number of valid bytes is the length defined in the assembler divided by 8.

CHAPTER 9/ OTHER FILES

The other files supplied on the OEM disks are explained below.

BDEXMPL.A65

This is an example of a block device driver and is to be used as a skeleton block device driver.

CDEXMPL.A65

This is an example of a character device driver and is to be used an a skeleton character device driver.

APLFLPY.A65

This is a block device driver which makes RWTS located at BD00H to interface with the APPLI-CARD. This is an example of a complete driver.

SVAZVX4.A65

This is also an example of a block device driver and shows how multiple density devices are supported.

DRVREQUS.A65

This is an include file used by the above routines. It includes system equates.

RDWRHST.ASM

This is a file which can be used by a Z-80 program to talk directly with the HOST 6502.

RWHEXMPL.ASM

This is an example of a program which uses RDWRHST.ASM. This program searches for a driver of a particular name and then moves the head of the driver off the data area.

M*.SUB

The files which begin with M and have and extent of .SUB are submit files which build a driver.

TECH SUPPORT NOTE

RE: Appli-card driver memory usage.

Note: These memory addresses are in the Apple's memory.

The 6502 control program (CP6502) begins at B000H. This code is approximately 700H bytes long. The drivers which have been installed in the DRIVERS file are loaded sequentially, following CP6502. SFTVIDEO is always the first driver, and it is about 18B0H bytes long.

If using the HIRESIO driver, it will be loaded at 6000H; this code is 1000H long, so it ends at 7000H. This driver uses the Apple hires pages at 2000H and 4000H.

There are device tables begining at 8E80H, so this area must not be affected.

Addresses 9000H to AFFFH contain a copy of the CP/M image, which is used on a reset (warm boot), rather than going to disk to get the image on a warm boot.

* Note: Locations given in the OEM manual on pg. 19 for beginning of drive code and SFTVIDEO location are in error.

APPLI-CARD BOOTSTRAP PROCESS

The bootstrap process for the APPLI-CARD on the Apple II, II+ and IIe is initiated by a 256-byte bootstrap written in 6502 code residing in the first sector of track 0 on the boot disk. This bootstrap is part of the file PCPICPM which is written to the system tracks by the INSTALL utility program. The PCPICPM file is structured as follows:

0000 - 00FF	Apple bootstrap (6502 code)
0100 - 1FFF	CP/M (CCP, BDOS and BIOS - Z80 code)
2000 - 27FF	CP6502 (6502 I/O control program)
2800 - 2FFF	RWTS (6502 floppy disk I/O code)

The Apple bootstrap reads the remainder of the PCPICPM file, containing CP/M, BIOS, CP6502 and RWTS, starting from sector 1 of track 0 into contiguous memory locations in the Apple beginning at address 9100 hexadecimal. The bootstrap then transfers control to the initialization entry point of the CP6502 module at B000 hexadecimal.

During the second phase of bootstrapping, the CP6502 initialization code creates the various tables it requires for later operation, locates the Z80 APPLI-CARD I/O slot, passes the CP/M and BIOS code up to the Z80, and then enters its normal waiting loop, ready to respond to commands from the Z80.

The third phase of bootstrapping is handled by the Z80 CP/M BIOS, which is part of the code uploaded by the CP6502 module in the Apple. The BIOS cold start routines initialize the Z80, I/O ports and interrupts, then call CP6502 to load the DRIVERS file into the Z80 and connect individual drivers to specific devices in the Apple. After device drivers are loaded and connected, the BIOS displays the sign-on message, loads and executes the autostart file (if any), and transfers control to CP/M.

BOOTING FROM OTHER THAN APPLE FLOPPY DISKS

Booting the APPLI-CARD CP/M system from disks other than the default Apple 5.25" floppy disks requires that several sections of executable code and default table information be examined and possibly altered to be compatible with the new bootstrap disk type. The areas which must be examined are:

1. Boot disk parameter block data (part of CP6502)
2. Default disk driver (part of CP6502)
3. Bootstrap loader (000H - 0FFH in PCPICPM)

The Apple I/O control program, CP6502, consists of three components, 1) the main control program, device driver communication and initialization routines, 2) a default disk driver used during the remainder of the bootstrap process, and 3) a default console driver. In some cases, the new boot

disk will be software compatible with standard Apple floppies and different only in density or total capacity. In these instances, the bootstrap sector and default disk driver need not change, and only the disk parameter tables in CP6502 must change. At CP6502+05FH are the following disk parameter tables:

HOST PARAMETER TABLE	(Size)	(Default value)
Bytes per sector	word	256
CP/M records per track	word	32
CP/M records per host block	byte	2
CP/M records per allocation block	byte	8
Sector mask	byte	1
Sector shift count	byte	1

DISK PARAMETER BLOCK		
Sectors per track	word	32
Block shift factor	byte	3
Block mask	byte	7
Extent mask	byte	0
Blocks on disk - 1	word	127
Directory entries - 1	word	47
Alloc0	byte	192
Alloc1	byte	0
Check masks	word	12
Directory track offset	byte	3

If the new boot disk is not software compatible with Apple floppies, both the bootstrap and the default disk driver in CP6502 must be changed (which generally means replacement). Two tables residing in the main CP6502 module contain pointers to the four entry points of each of the default drivers. The block driver table is located at CP6502+04AH and the character driver table is immediately following at CP6502+052H. The four words of the default disk driver table must be changed to point to the new disk driver's INIT, READ, WRITE and OTHER entry points.

Generally the default character driver is loaded at an address lower than the disk driver, so that a new default disk driver may be loaded directly over the existing driver. The new driver may be larger than the original, provided it does not extend into RWTS, located at CP6502+800H. If the new CP/M configuration will never need to use RWTS, that is, never use ANY Apple floppy driver, then the new disk driver may extend into the space reserved for RWTS.

APPLI-DISC RAM Extender
Usage Notes

The PCPI APPLI-CARD can have up to two Appli-Disc RAM extender cards installed to provide up to 256K bytes of RAM memory in addition to the 64K bytes on the Appli-Card. The additional memory on each RAM extender is viewed as either one or two banks of 64K bytes each.

I. SOFTWARE ADDRESSING

RAM banks in an Appli-Card with RAM extender(s) installed are numbered as follows:

BANK #		
0	=	APPLI-CARD main 64K memory
2	=	1st 64K bank (RAM extender 1, bank 1)
4	=	2nd " " (" " 1, " 2)
6	=	3rd " " (" " 2, " 1)
8	=	4th " " (" " 2, " 2)

RAM banks are selected by writing a byte with a Z80 "OUT" instruction to port 0C0H.

Bits:	7	6	5	4	3	2	1	0	
									BANK #
	X	C	X	X	0	0	0	X	0
					0	0	1	X	2
					0	1	0	X	4
					0	1	1	X	6
					1	0	0	X	8

Data byte bits 1, 2, and 3 select the RAM bank. Bits 7, 5, 4, and 0 are ignored.

Bit 6 controls an 8 or 16K byte common area.
- = 1 disables common RAM and enables the entire 64K in the selected bank.
- = 0 enables a common bank of either 8K or 16K on the APPLI-CARD, depending upon jumper position, disabling the corresponding addresses on the RAM extender card. The 8K common area resides at addresses E000H-FFFFH; the 16K bank is at C000H-FFFFH.

Note: I think that instructions on bit 6 are backwards. My investigations imply that a 1 enables common area...

II. HARDWARE JUMPERS

RAM extender boards contain several plug-jumper locations which are used to assign bank addresses and common bank configuration for each board. Near the upper left corner of each RAM extender board the following jumpers and labels can be found: ["o" indicates a pin, "---" is a jumper]

```
          16K      8K
COMM.      o   o---o
SIZE
```

The jumper choice for 8K or 16K common area size is normally on the 8K side. The common area is that area of memory retained from the Appli-card's bank of 64K to each 64K bank of extended memory. The common area is the high 8K (or16K) address locations, which will contain an image of the CP/M operating system.

```
 1ST          2ND
 BD. o---o    o BD.

     o---o    o

     o---o    o
```

The 3 jumper plugs should be installed as shown above (covering left & middle pins) for the first extender board. If a second extender board is present, the 3 jumper plugs for the second board should be installed in the right-hand position (covering right & middle pins).

TECH SUPPORT NOTE

RE: APPLI-CARD IOBYTE alteration notes
DATE: Summer '84, P.B.

The Appli-card's IOBYTE default setting can be reconfigured in version 1.5 and later. Also, the prototype arrays for defining the corresponding character devices for each physical device setting can be altered in all versions.

At times it is desired to put a printer interface card in a slot not currently supported by CP/M for printer output. The details here will aid in modifying the APPLI-CARD CP/M so that the LST: will support printer output to an interface card in a slot other than those supported (slots 1 and 4).

The 4 assignments the LST: (printer) can have are TTY:, CRT:, LPT:, and UL1:. Respectively, these have character device values of 0,3,1,4 (which correspond to slot numbers in the Apple).

The IOBYTE default is for the LST: to = LPT: , therefore the printer interface card would normally be in slot 1. You could, for instance, put a printer card in slot 4 and use the STAT command to reassign the LST: device (i.e. STAT LST:=UL1:).

Since slots 2, 5, or 7 aren't assigned to one of the device names, it would be necessary to patch PCPICPM to enable use of a printer card in one of these slots.

The APPLI-CARD IOBYTE default setting can be reconfigured in version 1.5 and later. Also, the prototype arrays, which define the char. device (slot #) for each physical device, can be altered in all versions.

First load PCPICPM into memory using DDT : A>DDT PCPICPM
The memory addresses given below assume PCPICPM is loaded at address 100H, which is where DDT loads it.

*** VERSION 1.0 ***
The IOBYTE default (at location 1866H) cannot be changed in this version, and will be initialized at 95H.

The prototype arrays (each being 4 bytes long) are located as follows:

CON:	1D63 thru 1D66
LST:	1D71 thru 1D74
RDR:,PUN:	1D7F thru 1D82

*** VERSION 1.5, 1.5A, 1.6, 1.6A ***
The IOBYTE default is located at 184D, and is set to 95H.
The 4 byte prototype arrays follow consecutively in this order:

CON:	184E thru 1851
LST:	1852 thru 1855
RDR:,PUN:	1856 thru 1859

*** VERSION 2.0 ***
The IOBYTE default is located at 185F, and is set to 95H.
The 4 byte prototype arrays follow consecutively in this order:

CON:	1861 thru 1864
LST:	1865 thru 1868
RDR:,PUN:	1869 thru 186c

*** Format of the IOBYTE ***
The IOBYTE itself shows the relationship between the logical devices (CON:,RDR:,PUN:,LST:) and the physical device names. This byte (8 bits) is broken down as follows:

bits	7 6	5 4	3 2	1 0
	LST:	PUN:	RDR:	CON:

CON: = bits 0 and 1
- 0 = TTY: (char. device 0)
- 1 = CRT: (" " 3)
- 2 = BAT: (" " 3,and output to LST:)
- 3 = UC1: (" " 1 or 2 depending on version)

RDR: = bits 2 and 3
- 0 = TTY: (char. device 0)
- 1 = PTR: (" " 2)
- 2 = UR1: (" " 1)
- 3 = UR2: (" " 4)

PUN: = bits 4 and 5
- 0 = TTY: (char. device 0)
- 1 = PTP: (" " 2)
- 2 = UP1: (" " 1)
- 3 = UP2: (" " 4)

LST: = bits 6 and 7
- 0 = TTY: (char. device 0)
- 1 = CRT: (" " 3)
- 2 = LPT: (" " 1)
- 3 = UL1: (" " 4)

The middle 4 bits (bits 2-5) in a prototype byte are the ones which establish what the Character Device Number will be.
Bits 0 and 1 have values of 0. Bits 6 and 7 have values of 1.

For example: the prototype byte for LPT: is set for char. device #1. This byte would look like this: 1 1 0 0 0 1 0 0
which equals a hex value of C4.

To change LPT: to be set to char. device # 7. The byte would look like this: 1 1 0 1 1 1 0 0 which equals a hex value of DC.

For your information, all values would be:

Bits	Hex	Char dev #
1 1 0 0 0 0 0 0	C0	0
1 1 0 0 0 1 0 0	C4	1
1 1 0 0 1 0 0 0	C8	2
1 1 0 0 1 1 0 0	CC	3
1 1 0 1 0 0 0 0	D0	4
1 1 0 1 0 1 0 0	D4	5
1 1 0 1 1 0 0 0	D8	6
1 1 0 1 1 1 0 0	DC	7

Using DDT, patch in this new hex value at the correct location as noted above by version number.

Once you have patched PCPICPM, then you must run the INSTALL program so that this is rewritten on the boot tracks.