Apple II Reference Manual

January 1978

APPLE II
Reference Manual

SIGNAL DESCRIPTION FOR KEYBOARD INTERFACE

B1-B7: 7 bit ASCII data from keyboard, positive logic (high level= "1"), TTL logic levels expected.

GND: System circuit ground. Ø Volt line from power supply.

NC: No connection.

RESET: System reset input. Requires switch closure to ground.

STROBE: Strobe output from keyboard. The APPLE II recognizes the positive going edge of the incoming strobe.

+5V: Positive 5-Volt supply. To avoid burning out the connector pin, current drain MUST be less than 1ØØmA.

-12V: Negative 12-Volt supply. Keyboard should draw less than 50mA.

PERIPHERAL CONNECTORS

The eight Peripheral Connectors mounted near the back edge of the APPLE II board provide a convenient means of connecting expansion hardware and peripheral devices to the APPLE II I/O Bus. These are Winchester #2HW25CØ-111 (or equivalent) 5Ø pin card edge connectors with pins on .1Ø" centers. Location and pin outs are illustrated in Figures 1 and 4.

SIGNAL DESCRIPTION FOR PERIPHERAL I/O

AØ-A15: 16 bit system address bus. Addresses are set up by the 65Ø2 within 3ØØnS after the beginning of $Ø_1$. These lines will drive up to a total of 16 standard TTL loads.

$\overline{\text{DEVICE SELECT}}$: Sixteen addresses are set aside for each peripheral connector. A read or write to such an address will send pin 41 on the selected connector low during $Ø_2$ (5ØØnS). Each will drive 4 standard TTL loads.

DØ-D7: 8 bit system data bus. During a write cycle data is set up by the 65Ø2 less than 3ØØnS after the beginning of $Ø_2$. During a read cycle the 65Ø2 expects data to be ready no less than 1ØØnS before the end of $Ø_2$. These lines will drive up to a total of 8 total low power schottky TTL loads.

$\overline{\text{DMA}}$: Direct Memory Access control output. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

DMA IN: Direct Memory Access daisy chain input from higher priority peripheral devices. Will present no more than 4 standard TTL loads to the driving device.

DMA OUT: Direct Memory Access daisy chain output to lower priority peripheral devices. This line will drive 4 standard TTL loads.

GND: System circuit ground. Ø Volt line from power supply.

$\overline{\text{INH}}$: Inhibit Line. When a device pulls this line low, all ROM's on board are disabled (Hex addressed DØØØ through FFFF). This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

INT IN: Interrupt daisy chain input from higher priority peripheral devices. Will present no more than 4 standard TTL loads to the driving device.

INT OUT: Interrupt daisy chain output to lower priority peripheral devices. This line will drive 4 standard TTL loads.

$\overline{\text{I/O SELECT}}$: 256 addresses are set aside for each peripheral connector (see address map in "MEMORY" section). A read or write of such an address will send pin 1 on the selected connector low during $Ø_2$ (5ØØnS). This line will drive 4 standard TTL loads.

$\overline{\text{I/O STROBE}}$: Pin 20 on all peripheral connectors will go low during $Ø_2$ of a read or write to any address C8ØØ-CFFF. This line will drive a total of 4 standard TTL loads.

$\overline{\text{IRQ}}$: Interrupt request line to the 65Ø2. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output. It is active low.

NC: No connection.

$\overline{\text{NMI}}$: Non Maskable Interrupt request line to the 65Ø2. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output. It is active low.

Q_3: A 1MHz (nonsymmetrical) general purpose timing signal. Will drive up to a total of 16 standard TTL loads.

RDY: "Ready" line to the 65Ø2. This line should change only during $Ø_1$, and when low will halt the microprocessor at the next READ cycle. This line has a 3K Ohm pullup to +5V and should be driven with an open collector output.

$\overline{\text{RES}}$: Reset line from "RESET" key on keyboard. Active low. Will drive 2 MOS loads per Peripheral Connector.

Figure 4 PERIPHERAL CONNECTORS
(EIGHT OF EACH)

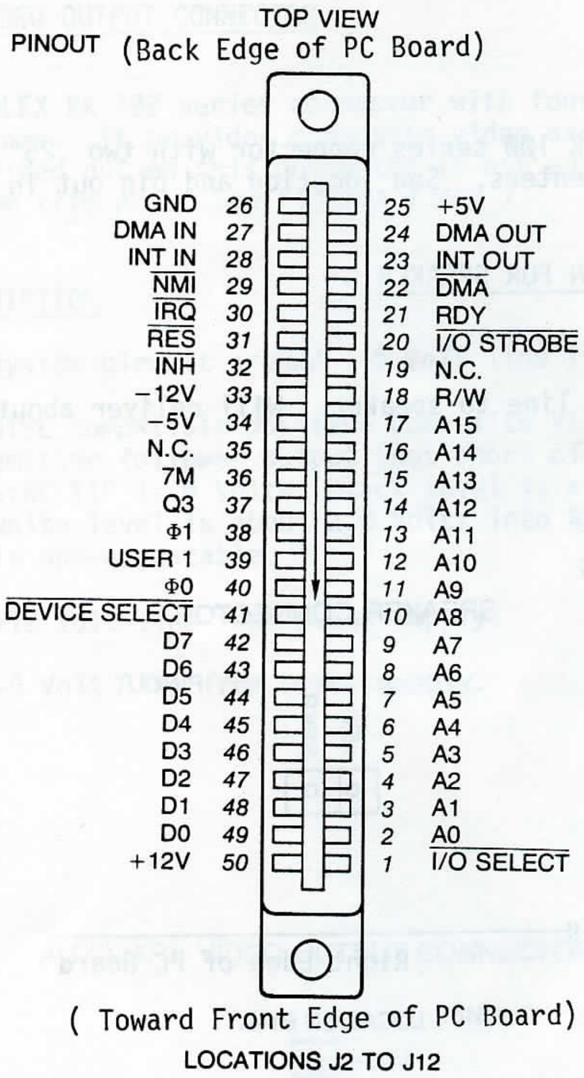
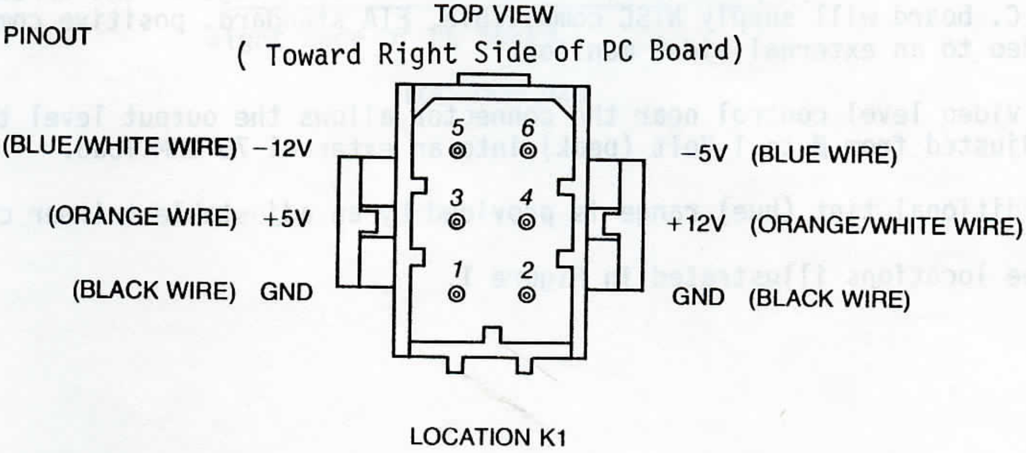


Figure 5 POWER CONNECTOR



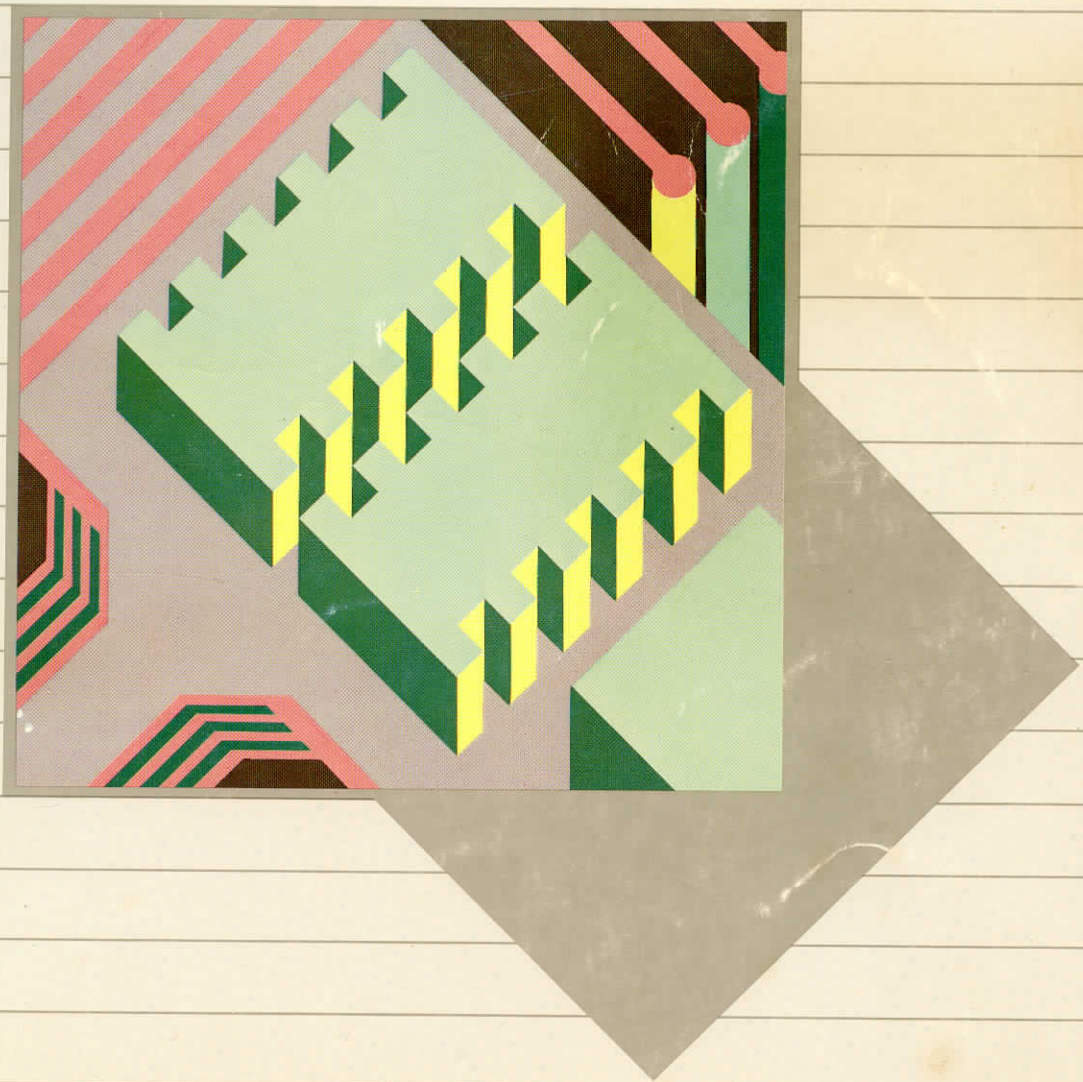
HEX ADDRESS	ASSIGNED FUNCTION	COMMENTS
C060/8	Cassette input	State of "Cassette Data In" appears in bit 7.
C061/9	"SW1"	State of Switch 1 input on Game I/O connector appears in bit 7.
C062/A	"SW2"	State of Switch 2 input on Game I/O connector appears in bit 7.
C063/B	"SW3"	State of Switch 3 input on Game I/O connector appears in bit 7.
C064/C	Paddle 0 timer output	State of timer output for Paddle 0 appears in bit 7.
C065/D	" 1 " "	State of timer output for Paddle 1 appears in bit 7.
C066/E	" 2 " "	State of timer output for Paddle 2 appears in bit 7.
C067/F	" 3 " "	State of timer output for Paddle 3 appears in bit 7.
C07X	"PDL STB"	Triggers paddle timers during $\emptyset_2$.
C08X	$\overline{\text{DEVICE SELECT}}$ 0	Pin 41 on the selected Peripheral Connector goes low during $\emptyset_2$.
C09X	" 1	
C0AX	" 2	
C0BX	" 3	
C0CX	" 4	
C0DX	" 5	
C0EX	" 6	
C0FX	" 7	
C10X	" 8	Expansion connectors.
C11X	" 9	"
C12X	" A	"

HEX ADDRESS	ASSIGNED FUNCTION	COMMENTS
C13X	$\overline{\text{DEVICE SELECT}}$ B	"
C14X	" C	"
C15X	" D	"
C16X	" E	"
C17X	" F	"
C1XX	$\overline{\text{I/O SELECT}}$ 1	Pin 1 on the selected Peripheral Connector goes low during $\emptyset_2$.
C2XX	" 2	
C3XX	" 3	NOTES: 1. Peripheral Connector 0 does not get this signal. 2. $\overline{\text{I/O SELECT}}$ 1 uses the same addresses as $\overline{\text{DEVICE SELECT}}$ 8-F.
C4XX	" 4	
C5XX	" 5	
C6XX	" 6	
C7XX	" 7	
C8XX	" 8, $\overline{\text{I/O STROBE}}$	Expansion connectors.
C9XX	" 9, "	
CAXX	" A, "	
CBXX	" B, "	
CCXX	" C, "	
CDXX	" D, "	
CEXX	" E, "	
CFXX	" F, "	
D000-D7FF	ROM socket D0	Spare.
D800-DFFF	" " D8	Spare.
E000-E7FF	" " E0	BASIC.
E800-EFFF	" " E8	BASIC.
F000-F7FF	" " F0	1K of BASIC, 1K of utility.
F800-FFFF	" " F8	Monitor.

Apple *II*

Reference Manual

For *//e* Only



Programming for Peripheral Cards

The seven expansion slots on the Apple IIe's main circuit board are used for installing circuit cards containing the hardware and firmware needed to interface peripheral devices to the Apple IIe. These slots are not simple I/O ports; peripheral cards can access the Apple IIe's data, address, and control lines via these slots. The expansion slots are numbered from 1 to 7, and certain signals, described below, are used to select a specific slot.

> The older Apple II and Apple II Plus models have an eighth expansion slot: slot number 0. On those models, slot 0 is normally used for a language card or a ROM card; the functions of the Apple II Language Card are built into the main circuit board of the Apple IIe.

Peripheral-card Memory Spaces

Because the Apple IIe's 6502 microprocessor does all of its I/O through memory locations, portions of the Apple IIe's memory space have been allocated for the exclusive use of the cards in the expansion slots. In addition to the memory locations used for actual I/O, there are memory spaces available for programmable memory (RAM) in the main memory and for read-only memory (ROM or PROM) on the peripheral cards themselves.

The memory spaces allocated for the peripheral cards are described below. Those memory spaces are used for small dedicated programs such as I/O drivers. Peripheral cards that contain their own driver routines in firmware like this are called intelligent peripherals. They make it possible for you to add peripheral hardware to your Apple IIe without having to change your programs, provided that your programs follow normal practice for data input and output.

Peripheral-card I/O Space

Each expansion slot has the exclusive use of sixteen memory locations for data input and output in the memory space beginning at location `$C090`. Slot 1 uses locations `$C090` through `$C09F`, slot 2 uses locations `$C0A0` through `$C0AF`, and so on through location `$C0FF`, as shown in Table 6-1.

These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Apple IIe addresses one of the sixteen I/O locations allocated to a particular slot, the signal on pin 41 of that slot, called `DEVICE SELECT'`, switches to the active (low) state. This signal can be used to enable logic on the peripheral card that uses the four low-order address lines to determine which of its sixteen I/O locations is being accessed.

Table 6-1 Peripheral-card I/O Memory Locations

Note: The enabling signal is marked with a prime, to indicate that it is an active-low signal.

Slot	Locations	Enabled by
1	`$C090-$C09F`	`DEVICE SELECT'`
2	`$C0A0-$C0AF`	`DEVICE SELECT'`
3	`$C0B0-$C0BF`	`DEVICE SELECT'`
4	`$C0C0-$C0CF`	`DEVICE SELECT'`
5	`$C0D0-$C0DF`	`DEVICE SELECT'`
6	`$C0E0-$C0EF`	`DEVICE SELECT'`
7	`$C0F0-$C0FF`	`DEVICE SELECT'`

Peripheral-card ROM Space

One 256-byte page of memory space is allocated to each peripheral card. This space is normally used for read-only memory (ROM or PROM) on the card with driver programs that control the operation of the peripheral device connected to the card.

The page of memory allocated to each expansion slot begins at location `$Cn00`, where n is the slot number, as shown in Table 6-2 and Figure 6-3. Whenever the Apple IIe addresses one of the

256 ROM memory locations allocated to a particular slot, the signal on pin 1 of that slot, called `I/O SELECT'`, switches to the active (low) state. This signal enables the ROM or PROM devices on the card, and the eight low-order address lines determine which of the 256 memory locations is being accessed.

Table 6-2 Peripheral-card ROM Memory Locations

Note: The enabling signal is marked with a prime, to indicate that it is an active-low signal.

Slot	Locations	Enabled by
1	$C100-$C1FF	I/O SELECT′
2	$C200-$C2FF	I/O SELECT′
3	$C300-$C3FF	I/O SELECT′
4	$C400-$C4FF	I/O SELECT′
5	$C500-$C5FF	I/O SELECT′
6	$C600-$C6FF	I/O SELECT′
7	$C700-$C7FF	I/O SELECT′

If there is an 80-column text card installed in the auxiliary slot, some of the functions normally associated with slot 3 are performed by the 80-column text card and the built-in 80-column firmware. With a 80-column text card installed, the `I/O SELECT'` signal is not available for slot 3, so firmware in ROM on a card in slot 3 will not run.

Expansion ROM Space

In addition to the small areas of ROM memory allocated to each expansion slot, peripheral cards can use the 2K-byte memory space from `$C800` to `$CFFF` for larger programs in ROM or PROM. This memory space is called expansion ROM space (see the memory map in Figure 6-3). Besides being larger, the expansion ROM memory space is always at the same locations regardless of which slot is occupied by the card, making programs that occupy this memory space easier to write. (See the section "I/O Programming Suggestions", below.)

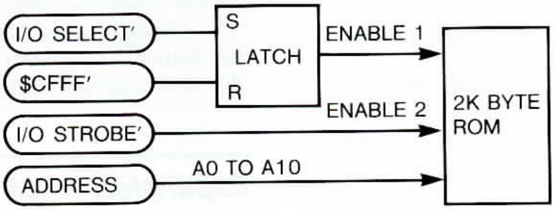
This memory space is available to any peripheral card that needs it. More than one peripheral card can have expansion ROM on it, but only one of them can be active at a time.

Each peripheral card that uses expansion ROM must have a circuit on it to enable the ROM. The circuit does this by a two-stage process: first, it sets a flip-flop when the I/O SELECT' signal, pin 1 on the slot, becomes active (low); second, it enables the expansion ROM devices when the I/O STROBE' signal, pin 20 on the slot, becomes active (low). Figure 6-1 shows a typical ROM-enable circuit.

The I/O SELECT' signal on a particular slot becomes active whenever the Apple IIe's 6502 microprocessor addresses a location in the 256-byte ROM address space allocated to that slot. The I/O STROBE' signal on **all** of the expansion slots becomes active (low) when the 6502 addresses a location in the expansion-ROM memory space, $C800-$CFFF. The I/O STROBE' signal is used to enable the expansion-ROM devices on a peripheral card (see Figure 6-1).

If there is an 80-column text card installed in the auxiliary slot, some of the functions normally associated with slot 3 are performed by the text card and the built-in 80-column firmware. With the text card installed, the I/O STROBE' signal is not available on slot 3, so firmware in expansion ROM on a card in slot 3 will not run.

Figure 6-1 Expansion ROM Enable Circuit

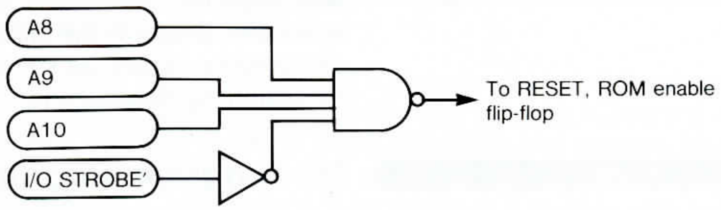


A program on a peripheral card can get exclusive use of the expansion ROM memory space by referring to location $CFFF in its initialization phase. This location is special: all peripheral cards that use expansion ROM must recognize a reference to $CFFF as a signal to reset their ROM-enable flip-flops and disable their expansion ROMs. Of course, doing so also disables the expansion ROM on the card that is about to use it, but the next instruction in the initialization code sets the flip-flop on the expansion-ROM enable circuit on the card. Once this has been done, this card has exclusive use of the expansion memory space and its program can jump directly into the expansion ROM.

CFFF = 53247

As described above, the expansion-ROM disable circuit resets the enable flip-flop whenever the 6502 addresses location $CFFF.

Figure 6-2 ROM Disable Address Decoding



To do this, the peripheral card must detect the presence of `$CFFF` on the address bus. You can use the `I/O STROBE'` signal for part of the address decoding, since it is active for addresses from `$C800` through `$CFFF`. If you can afford to sacrifice some ROM space, your can simplify the address decoding even further and save circuitry on the card. For example, if you give up the last 256 bytes of expansion ROM space, your disable circuit only needs to detect addresses of the form `$CFxx`, and you can use the minimal disable-decoding circuitry shown in Figure 6-2.

Peripheral-card RAM Space

There are 56 bytes of main memory allocated to the peripheral cards, eight bytes per card, as shown in Table 6-3. These 56 locations are actually in the RAM memory reserved for the text and low-resolution graphics displays, but these particular locations are not displayed on the screen and their contents are not changed by the built-in output routine `COUT1`. Programs in ROM on peripheral cards use these locations for temporary data storage.

Table 6-3 Peripheral-card RAM Memory Locations

*Note: The RAM locations normally allocated to slot 3 are taken over by any card installed in the auxiliary slot.

Base			**Slot Number**				
Address	**1**	**2**	**3***	**4**	**5**	**6**	**7**
$0478	$0479	$047A	$047B*	$047C	$047D	$047E	$047F
$04F8	$04F9	$04FA	$04FB*	$04FC	$04FD	$04FE	$04FF
$0578	$0579	$057A	$057B*	$057C	$057D	$057E	$057F
$05F8	$05F9	$05FA	$05FB*	$05FC	$05FD	$05FE	$05FF
$0678	$0679	$067A	$067B*	$067C	$067D	$067E	$067F
$06F8	$06F9	$06FA	$06FB*	$06FC	$06FD	$06FE	$06FF
$0778	$0779	$077A	$077B*	$077C	$077D	$077E	$077F
$07F8	$07F9	$07FA	$07FB*	$07FC	$07FD	$07FE	$07FF

A program on a peripheral card can use the eight base addresses shown in the table to access the eight RAM locations allocated for its use, as shown in the next section, "I/O Programming Suggestions".

I/O Programming Suggestions

A program in ROM on a peripheral card should work no matter which slot the card occupies. If the program includes a jump to an absolute location in one of the 256-byte memory spaces, then the card will only work when it is plugged into the slot that uses that memory space. If you are writing the program for a peripheral card that will be used by many people, you should avoid placing such a restriction on the use of the card.

> To function properly no matter which slot a peripheral card is installed in, the program in the card's 256-byte memory space must not make any absolute references to itself. Instead of using jump instructions, you should force conditions on branch instructions, which use relative addressing.

The first thing a peripheral-card subroutine should do is to save the contents of the 6502's registers. One way to do this is to use the monitor subroutine `IOSAVE`. This subroutine, which starts at location `$FF4A`, stores the registers in zero-page memory locations `$45-$49`. A companion subroutine, `IOREST`, restores the registers from these memory locations. Your program should call `IOREST`, which starts at location `$FF3F`, just before it returns control to the program that called it.

This method of saving the registers is convenient, but it is not always safe. If a second subroutine calls `IOSAVE`, or if an interrupt occurs, the new register contents get saved in the same locations, and the old ones get destroyed. It is safer, though somewhat slower, to save the registers on the stack, and restore them just before returning control to the calling program.

Most single-character I/O is done via the 6502's accumulator. A character being output through your subroutine will be in the accumulator with its high bit set when your subroutine is called. Likewise, if your subroutine is performing character input, it must leave the character in the accumulator with its high bit set when it returns to the calling program.

subroutines, control stack: see Section 3.4

FOR/NEXT loops: see Section 3.3

Warning

Be careful where you execute CLEAR. Since CLEAR resets Applesoft's internal control stack, using it in the midst of a subroutine or in a FOR/NEXT loop can interfere with the orderly flow of program execution. The following program, for example, will fail in line 30 with a NEXT WITHOUT FOR error:

```
10 FOR X = 1 TO 10        —try to loop 10 times

20 PRINT X
30 CLEAR                  —CLEAR resets control stack
                           (among other things)

40 NEXT X                 —program fails here—doesn't
                           know it's in a loop

50 PRINT "HI!"            —program won't get this far
```

1.2.3 The LIST Command

```
LIST
LIST 100
LIST 100,
LIST - 200
LIST ,200
LIST 100, 200
LIST 100 - 200
```

LIST **displays or prints a program**

PR# **statement:** see Section 5.2.1

The LIST command displays on the screen all or part of the program currently in memory, or writes it to the current output device as specified in the last PR# statement. (For example, if there is a printer connected to slot 1, and if the statement PR# 1 has been executed, then the program listing is sent to the printer.)

Listing the entire program

To list the entire program, just type the keyword LIST and press RETURN:

```
LIST
```

Listing a portion of the program

You can list a portion of the program by specifying the first and last lines you want to list, separated by either a comma or a dash:

```
LIST 100, 250             —display lines 100 through
                           250

LIST 100 - 250            —also display lines 100
                           through 250
```

Apple *II*

Applesoft BASIC Programmer's Reference Manual - Volume I

For *//e* Only

5.2 Output

output: the transfer of information from the computer to an external destination

This section describes the *output* facilities available in Applesoft:

PR# statement: see Section 5.2.1

PRINT statement: see Section 5.2.2

number formats: see Section 5.2.3

screen formatting: see Section 5.2.4

miscellaneous output: see Section 5.2.5

- Section 5.2.1 covers the PR# statement, which controls the destination to which output is directed.
- Section 5.2.2 contains a detailed discussion of the PRINT statement, Applesoft's primary output statement.
- Section 5.2.3 gives details on the way numbers are formatted when written with the PRINT statement.
- Section 5.2.4 describes Applesoft's wide variety of facilities for controlling the format in which textual information is displayed on the screen.
- Section 5.2.5 touches briefly on various miscellaneous output facilities not covered elsewhere: the Apple IIe's built-in speaker, annunciator outputs, utility strobe, and cassette tape output.

5.2.1 The PR# Statement

```
PR# 1
PR# X
PR# SLOT - J
```

PR# specifies destination for subsequent output

expansion slot: see *Apple IIe Owner's Manual* and *Apple IIe Reference Manual*

The PR# statement specifies the destination to which the computer will send subsequent output. The expression following the keyword PR# should evaluate to a number between 0 and 7, designating the expansion slot to which output is to be sent.

Slot number 0 specifies output to the screen

When Applesoft is started up, it is set to send output to the display screen. Executing a PR# statement with a slot number from 1 to 7 instructs Applesoft to send output instead to the peripheral output device (such as a printer, terminal, or modem) connected to the designated slot. A slot number of 0 reestablishes the display screen as the current output device. For example, the following program fragment writes a string of characters to the device connected to slot 1, then reestablishes screen output:

```
610 PR# 1              —send output to device in slot 1
620 PRINT Z$           —write contents of string vari-
                         able Z$ to device in slot 1
630 PR# 0              —send future output to screen
```

Notice that the character # is part of the keyword PR# and cannot be omitted.

IN# **statement:** see Section 5.1.1

Restarting the System with PR#: If the slot designated in an IN# or PR# statement contains a disk controller card, Applesoft will attempt to restart (often called "booting") the system from the disk contained in drive 1 connected to that slot. When you do this on purpose, it's the usual way of restarting the system from within Applesoft; when you do it by mistake, it can be a catastrophe.

Be careful!

▲

Warning

If no output device is connected to the slot designated in a PR# statement, the system will hang. To recover, use CONTROL - RESET .

CONTROL - RESET : see Section 1.3.2

A slot number between 8 and 255 will cause unpredictable and possibly aberrant behavior.

▲

Warning

If you are using the Apple IIe 80-Column Text Card, always be sure to deactivate it by typing ESC CONTROL -Q before using PR# to transfer output to another slot. Leaving the Text Card active while using a printer or while restarting the system from a disk may produce amusing but confusing fireworks on the screen.

Although the Text Card is installed in the Apple IIe's special auxiliary slot, it appears to the computer as if it were in slot 3. So to reactivate the Text Card after sending output to another device, type

```
PR# 3
```

You can also return output to the 40-column screen with the Text Card inactive by typing

```
PR# 0
```

However, don't use PR# 0 to redirect output directly from the Text Card to the 40-column screen without first deactivating the Text Card with ESC CONTROL -Q. Under certain circumstances, this may cause text intended for the screen to be written outside the area of memory reserved for it, possibly destroying your Applesoft program or other important information.

A slot number less than 0 or greater than 255 will stop the program with the message

```
?ILLEGAL QUANTITY ERROR
```

POS

Syntax: POS (expr)
Example: POS (0)

Yields the current horizontal position of the cursor on the text display. The argument is ignored, but must be a valid Applesoft expression. [5.2.4]

PR#

Syntax: PR# aexpr
Example: PR# 1

Specifies the destination for subsequent output. The example causes subsequent output to be sent to the device in expansion slot 1. [5.2.1]

PRINT

Syntax: PRINT [{expr [,|;]}]
Example: PRINT
PRINT A$, "X = ";X

Writes a line of output to the current output device. The first example writes a blank line; the second writes the value of variable A$, followed at the next available tab position by the string "X = ", followed immediately by the value of variable X. [5.2.2]

READ

Syntax: READ var [{ ,var}]
Example: READ A, B%, C$

Reads values from DATA statements in the body of the program. The example reads values into variables A, B%, and C$. [5.1.4]

RECALL

Syntax: RECALL name[%]
Example: RECALL MX

Reads values into an array from a tape cassette. The example reads values into array MX. [M]