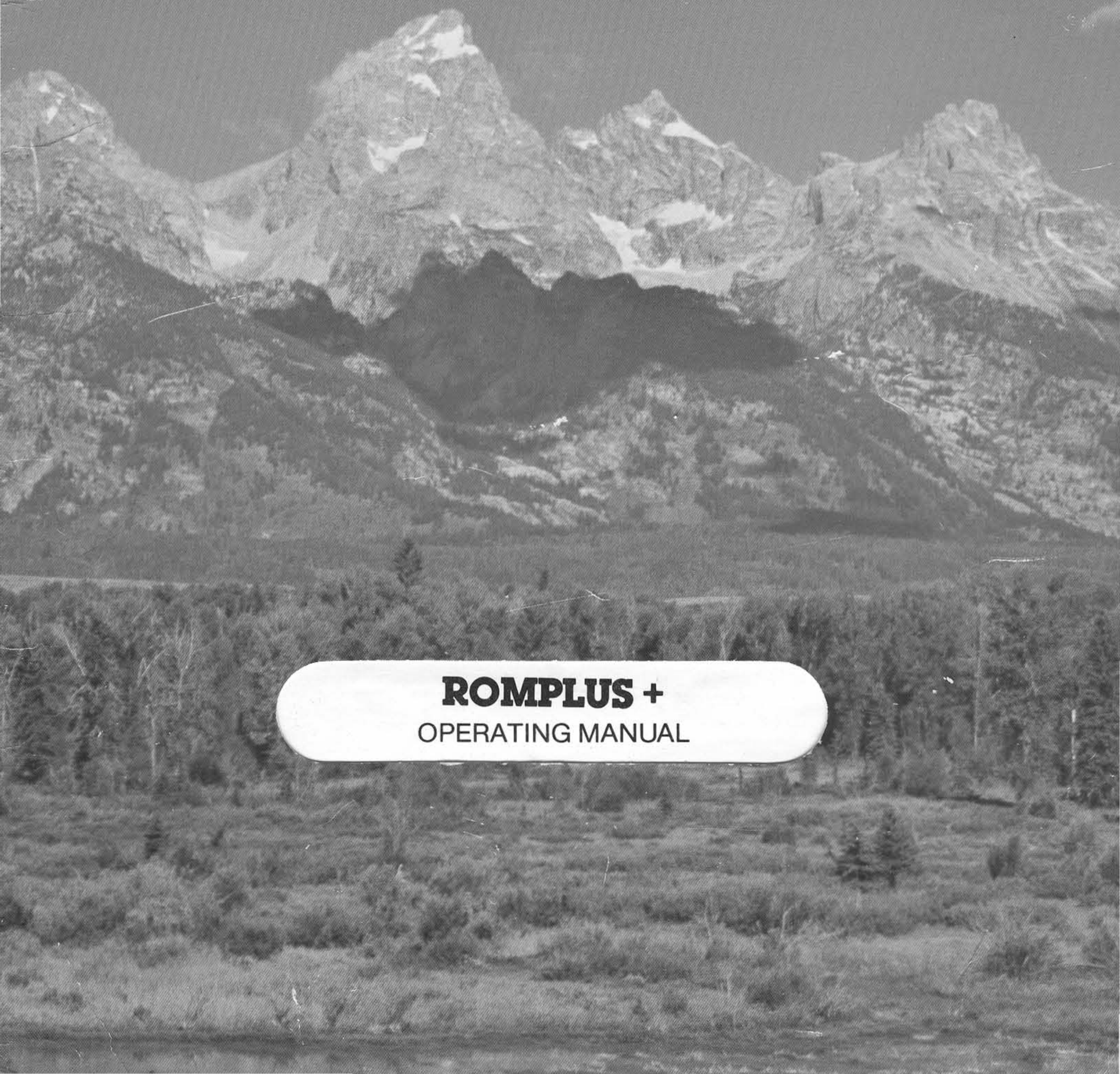
Mountain Computer
INCORPORATED



ROMPLUS +
OPERATING MANUAL

ROMPLUS +

OPERATING MANUAL

Manual Part Number 11-00233-01D

Table of Contents

INTRODUCTION

Mountain Hardware's ROMPLUS+ is a powerful addition to your Apple II* computer. ROMPLUS+ has room for six of the 2316 type ROM's, or the 2716 EPROM. With each 2316 chip holding 2K bytes of memory, ROMPLUS+ has the capacity of 12K bytes of read only memory.

Whether your applications of the Apple II are for business, education, research, or just fun, eventually you will discover a set of programs that you use constantly. Examples are special peripheral drivers, utility routines, and data collection programs. You may access these programs on the ROMPLUS+ board as soon as you turn your Apple II on.

Additionally, ROMPLUS+ provides 255 bytes of RAM which may be activated or deactivated under program control. The on-board control ROM simplifies your program selection. You need only type a few keystrokes to run any program on ROMPLUS+. The control ROM relieves the burden of remembering many different addresses. ROMPLUS+ also has two TTL level inputs, and these are available for any user application. For example, an option on Mountain Hardware's Keyboard Filter ROM uses one of these inputs to monitor the shift key on the Apple II's Keyboard.

This manual is a user's manual for ROMPLUS+. In this manual, we cover installation, hardware features of ROMPLUS+, using the ROMPLUS+, and writing your own PROMs.

*Apple II is a trademark of Apple Computer, Cupertino, CA.

INSTALLATION

To install ROMPLUS+ simply follow these instructions:

1. Turn off the power switch at the back of the Apple II. The removal or insertion of any card with power on could cause severe damage to both the computer and ROMPLUS+.

2. Remove the cover from the Apple II by pulling up on the cover at the rear edge.

3. Now choose an Apple II slot number. Slot number 0 should never be used as it is reserved for Apple's language cards. In general, we recommend that you install ROMPLUS+ into a slot immediately below the disk controller card. For example, if the disk is in slot #6, place ROMPLUS+ into slot #5. The only restriction is that you may not place ROMPLUS+ into slot number 0.

4. Plug ROMPLUS+ into the slot you have chosen. Make sure the board is firmly seated in the socket.

5. Replace the cover on your Apple II and turn on your computer.

Chapter 1

HARDWARE FEATURES

General

In this section, we discuss in detail the hardware features of ROMPLUS+. The four basic parts are the ROM sockets, the RAM, the TTL inputs and the control ROM.

The ROMPLUS+ board is shown in Figure 1. This figure gives the layout of the board's features.

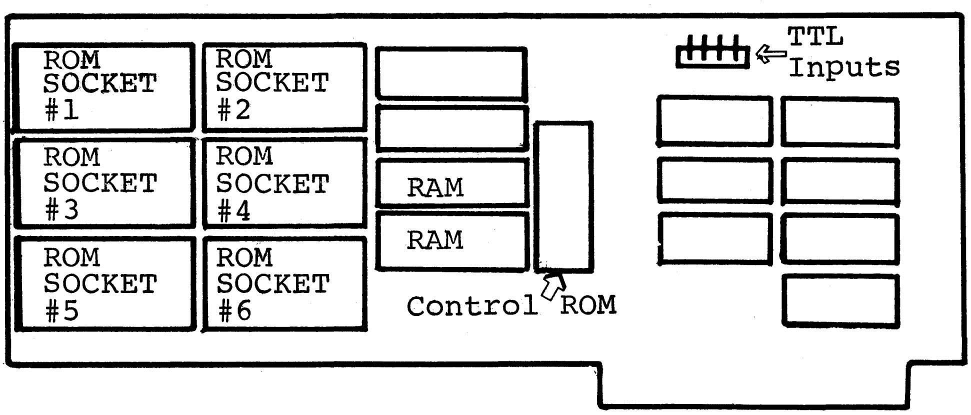


FIGURE 1.

ROM Space

ROMPLUS+ has six 24-pin sockets located on the left side of the board. These sockets accept the 5 volt 2316 type of ROM chips, with each chip holding 2048 bytes. A pin for pin compatible cEPROM, such as the 2716 may also be used in the ROM sockets.

All of the ROM chips are mapped into the $C800-$CFFF memory address space, but only one chip is mapped at any one time. If your application program is larger than 2048 bytes, do not worry. There is a scheme for switching control from one chip to another chip. This scheme, plus information for creating your own chips, is given in the Advanced Programmers Information chapter.

RAM Space

ROMPLUS+ has 256 bytes of read-write memory (RAM) on-board. This RAM may be activated or deactivated under program control. When activated, the RAM maps into the $CF00-$CFFE memory address space. Notice that only 255 bytes are available. The last byte at location $CFFF may not be used. This is because of the Apple II's peripheral convention which deactivates all peripheral boards when memory address $CFFF is referenced. Also, when the RAM is active, the top 256 bytes of the selected ROM chip are not available. This is because the RAM maps into the same space used by the ROM chip. If your ROM chip uses all of its 2048 bytes, simply deactivate the RAM. See chapter 5 for information on the control word used to activate or deactivate the RAM.

The RAM will retain its contents whether ROMPLUS+ is active or not in use. The RAM, of course, loses its contents when power is switched off.

The RAM provides the ROM chips with their own private storage area. This will help to minimize memory conflicts. However, the RAM may be used by any program in the Apple II.

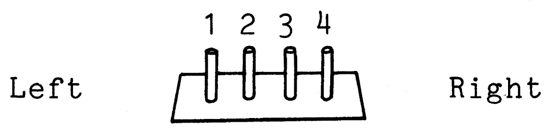
We recommend that the RAM be allocated in the following way:

Address	Use
$CF00-$CF03	Scratch area for control ROM
$CF04-$CF5F	Scratch area for ROM socket #1
$CF60-$CF7F	Scratch area for ROM socket #2
$CF80-$CF9F	Scratch area for ROM socket #3
$CFA0-$CFBF	Scratch area for ROM socket #4
$CFC0-$CFDF	Scratch area for ROM socket #5
$CFE0-$CFFE	Scratch area for ROM socket #6

Chapter 1 HARDWARE FEATURES

TTL Inputs

A four pin connector on ROMPLUS+ provides two TTL level inputs and two ground pins. A matching four pin plug with wire is available from Mountain Computer. Order Part No. MHP-X021. Price $3.00.



Pins 2 and 3 are grounded. Pins 1 and 4 are the TTL inputs. The inputs are held high by pull-up resistors. Therefore, an unused input will be read as a high level, or a "1". The TTL inputs are read through the control word. Chapter 5 has more information on the control word. Pin 1 on the connector maps to bit 4 of the control word. Pin 4 on the connector maps to bit 5 of the control word.

Control ROM

The control ROM provides the "intelligence" which makes ROMPLUS+ easy to use. It controls input and output functions and allows for easy ROM socket selection and entry point selection. Many of its features are in the next chapter, Using ROMPLUS+.

The control ROM occupies the memory address space $CN00-$CNFF, where N is the slot number. The ROM is supplied with power whenever it is addressed. This results in a power-saving.

Chapter 2

USING ROMPLUS+

This chapter covers the basic information you need for typical operation of ROMPLUS+. This chapter should be read carefully. We will cover such topics as selecting ROMPLUS+, activating RAM, ROM socket selection, and entry-point selection.

Activating ROMPLUS+

ROMPLUS+ is a peripheral that is activated in the same manner as other Apple II peripherals. From BASIC, ROMPLUS+ is turned on by a "IN#n" or "PR#n" command, where n is the slot number. From the monitor, a "nCTRL-K" or "nCTRL-P" command will turn on ROMPLUS+. If you are running BASIC under DOS, use the regular DOS procedure of printing a CTRL-D followed by the command. Whenever the board is activated, the RAM is also activated.

The board is deactivated by using both the "IN#0" and "PR#0" commands. Hitting the "RESET" key will also deactivate ROMPLUS+. If another peripheral card is accessed via the "IN#n" or "PR#n" commands, ROMPLUS+ will be deactivated. Of course, any reference to address $CFFF will deactivate ROMPLUS+ (or any other peripheral board).

Once ROMPLUS+ has been activated, all input and output operations are vectored through the control ROM. This is transparent to the user, i.e., nothing seems different. However, the control ROM is looking for one of two special command characters. If the character passed on input or output is not a special command character, it is passed to the input or output routine. If the character is a command, then the next two characters are interpreted as parameters of the command.

Commands

The two command characters are CTRL-SHIFT-M and CTRL-SHIFT-N (ASCII codes $9D and $9E respectively). You may obtain these characters by pressing the CONTROL, SHIFT, and letter keys simultaneously. These characters were chosen to minimize typing accidents.

The syntax of the commands are:

CTRL-SHIFT-M<ROM socket #><entrypoint>.
CTRL-SHIFT-N<ROM socket #><entry point>.

There are no spaces between the command character, the ROM socket #, and the entry point. The brackets are not entered. No return is necessary after the command. Notice the command is three characters long.

ROM socket number is a value from 0 to 6 which specifies which ROM socket you want to select. Only one ROM socket is active at one time, but one ROM socket may call another ROM socket. Selecting chip number 0 will deactivate the current ROM socket without deactivating ROMPLUS+. If an invalid ROM socket is selected, the "bell" will beep.

Entry point is a letter, starting with A, and ending with a letter depending on the particular ROM chip selected. The number of entry points on any ROM is determined by information on that particular ROM. The first entry point is always "A", the second entry point is "B", and so on. If an illegal entry point is specified, the bell will beep. The documentation accompanying any commercially available ROM for ROMPLUS+ will detail the valid entry points of that ROM. If you write your own ROM, you will place a table of entry points on the ROM. The number of entry points determines the valid entry point characters. More information on writing your own ROM chips is in the next chapter.

Chapter 2 USING ROMPLUS+

CTRL-SHIFT-M

This command selects one of the two operating modes of ROMPLUS+. The CTRL-SHIFT-M command will let the selected ROM gain control every time a character is inputed or outputed. When this command is issued, all subsequent input and output is vectored through two hooks which are located on the selected ROM.

Recall that when ROMPLUS+ is activated, the input and output is vectored through the control ROM. This means that when a character is input, a call is placed to the control ROM which calls the input driver. The control ROM inspects this character and then passes it along to the program requesting input. Similarly, on output of a character, a call is placed to the control ROM, which inspects the character and then calls the output driver. Whenever ROMPLUS+ is not active, input and output are not vectored through the control ROM. Instead, they are vectored to the normal input and output drivers of the Apple.

When the CTRL-SHIFT-M command is given, the input and output are now vectored through the input and output hooks on the selected ROM. Normally, these input and output hooks point to locations within the selected ROM. More information about the hooks is in the next chapter.

In general, all of the hooks and vectors are transparent to the user. When ROMPLUS+ is deactivated, I/O vectors through the normal Apple II I/O drivers. When ROMPLUS+ is active, I/O is vectored through the control ROM. When a CTRL-SHIFT-M command is given, all subsequent I/O is vectored through the selected ROMs' I/O hooks. The ROMs' I/O hooks are located in the branch table. More information about the branch table is in the next chapter.

The net effect of the CTRL-SHIFT-M command is that the selected ROM gains control on every input or output character. This continues until ROMPLUS+ is deactivated, or the particular ROM is deactivated. Examples of the type of program which use this mode of operation are printer drivers, or Mountain Hardware's Keyboard Filter. These programs need to execute with every input or output operation.

CTRL-SHIFT-N

This command selects one of two operating modes of ROMPLUS+. The CTRL-SHIFT-N command will pass control to the selected ROM program. This program is executed immediately and then control returns. If this command was printed as part of a BASIC program, then control returns to BASIC. If this command was entered immediately from the keyboard, then control returns to the keyboard.

A program executed by the CTRL-SHIFT-N command in one ROM may execute another ROMPLUS+ program in another ROM by outputting anothr CTRL-SHIFT-N command. However, a program executed by the CTRL-SHIFT-N command may not output a CTRL-SHIFT-M command. In the former case, the control ROM keeps track of control. In the later case, the control ROM keeps track of control. In the later case, we have a situation which is logically meaningless. It does not make sense to have a routine type of program calling a special driver type program.

It does make sense however, to have a driver type program (activated by CTRL-SHIFT-M) call upon a routine type program (CTRL-SHIFT-N). For example, a program such as Keyboard Filter might call upon a routine on another ROM. It would output a CTRL-SHIFT-N command. The control ROM keeps track of the calling ROM and the called ROM. It returns control to the calling ROM when the called ROM returns.

Selecting RAM

Any time ROMPLUS+ is activated, or any ROM is activated via the CTRL-SHIFT-M or CTRL-SHIFT-N commands, the on-board RAM is activated. Whenever this RAM is active, the top 256 bytes of the selected ROM are not available. If your program uses the top 256 bytes of the ROM, you must deactivate the RAM before the code is executed. Otherwise, the computer will read the contents of RAM and interpret that data as instructions. This usually results in disaster. It is necessary to reactivate RAM before returning control. The next chapter contains a few routines used for controlling the state of the RAM.

Notes

The control ROM on ROMPLUS+ makes use of two locations in memory normally used by the monitor. These two locations are $3A and $3B. As a result, whenever ROMPLUS+ is active, the monitor "L" command for disassembly and the Apple II mini-assembler will not work properly. To restore these commands, deactivate ROMPLUS+.

The Apple II peripheral scheme states that all ROM's in the $C800-$CFFF space must be de-selected whenever $CFFF is referenced. Therefore, take care that your programs never reference location $CFFF.

Chapter 3

ADVANCED PROGRAMMER'S INFORMATION

This chapter contains information for the advanced use of ROMPLUS+. The sections about the control word and the control ROM should be read by anyone using ROMPLUS+. The other sections about the branch table, preparing your ROM, and programs greater than 2K bytes are intended for the user that will prepare their own ROM chip for use in ROMPLUS+. However, anyone using ROMPLUS+ will benefit from the information in those sections.

The Control Word

The features of ROMPLUS+ are controlled by the control word. The control word is a read/write word located at a slot dependent memory address. The address of the control word is $C080+$N0 (or -16256+16*N from BASIC), where N is equal to the slot number. The following table summarizes:

Slot #	Hex Address	BASIC Address
1	$C090	-16240
2	$C0A0	-16224
3	$C0B0	-16208
4	$C0C0	-16192
5	$C0D0	-16176
6	$C0E0	-16160
7	$C0F0	-16144

A write to the control word location may be used to select a ROM socket, activate or deactivate the board, or activate or deactivate the RAM. The function of the particular bits are described below:

|7|6|5|4|3|2|1|0| control word

Bit 7: This bit controls the RAM. If a "0" is written, the RAM is deactivated. If a "1" is written, the RAM is activated.

Bit 6-4: Unused.

Bit 3: This bit controls the board. If a "0" is written, the ROMPLUS+ is deactivated. If a "1" is written, the board is activated.

Bit 2-0: These bits select the ROM socket to be enabled. Bit two is the most significant bit of the value. If the value=0, then none of the ROMs are enabled. If set from 1 to 6, the corresponding ROM is enabled. The value should never equal 7.

A read to the control word is used to check the status of the RAM, find the currently enabled ROM socket number, or to sense the value of the two TTL inputs. The function of the particular bits are described below.

|7|6|5|4|3|2|1|0| control word

Bit 7: This bit reads the status of RAM. If equal to "0", then RAM is deactivated. If equal to "1", then RAM is active.

Bit 6: Unused.

Bit 5: TTL input from pin 4.

Bit 4: TTL input from pin 1.

Bit 3: Unused.

Bit 2-0: These bits indicate which Rom socket is currently enabled. The value is determined the same way as the bits 2-0 of the written control word.

We next examine several programming examples of control word use. First, if we wish to activate ROMPLUS+ and select ROM socket number one, we use these machine language instructions:

```
LDA #$89        :RAM active, board active, ROM #1

STA $C080,X     :Write control word
```

In that example, and in the examples to follow, we assume that the X register contains the slot number (1-7) multiplied by 16. This is the standard convention for slot independent I/O on Apple II.

To do the same thing in BASIC, we use a statement like this:

```
POKE -16256+16*SLOT,137
```

Now suppose you wish to activate ROMPLUS+, deactivate the RAM, and select ROM #5. You would do one of the following:

```
LDA #$0D        :Deactivate RAM, activate ROMPLUS+, select
                 ROM #5.

STA $C080,X     :Write control word
or POKE -16256+16*SLOT,13
```

If you wish to toggle the state of the RAM (i.e., turn off when it is on and turn on when it is off), you would use this code:

```
LDA  $C080,X :Read control word
EOR  #$80    :
ORA  #$80    :
STA  $C080,X :Write control word
```

From BASIC, use these statements:

```
S=(PEEK(-16256+16*SLOT)+128)MOD 256
IF S MOD 16<8 THEN S=S+8
POKE  -16256+16*SLOT,S
```

It is necessary to set bit 3 so that you don't deactivate ROMPLUS+. This final example will test the TTL input at bit 4.

```
LDA $C080,X   :Read control word
BIT #$10      :Mask bit #4
BNE           :If bit is set
BEQ           :If bit is clear
```

In BASIC:

```
IF (PEEK(-16256+16*SLOT)MOD 32)>15 THEN BIT IS SET
```

Remember that when writing the control word, bit 3 must be set to activate ROMPLUS+. Even if ROMPLUS+ is already active, bit 3 must be set if you do not want to deactivate ROMPLUS+.

If a read of the current ROM chip yields ROM socket number zero as the active ROM, then no ROM is active. If ROMPLUS+ is not active, then the current ROM chip will read as ROM socket number zero.

Control ROM

The control ROM provides "intelligence" for ROMPLUS+. It is a 256 byte memory which controls the functions of ROMPLUS+. A complete source listing is in the Appendix. In this section, we will detail memory usage and entry points of the control ROM.

Chapter 3 ADVANCED PROGRAMMERS INFORMATION

The control ROM uses two bytes of memory in the zero page. These two locations are $3A and $3B. These two locations were chosen to take advantage of the monitor indirect jump at $FEBC. The use of the two page zero memory locations ($3A & $3B) causes a memory conflict with two of the monitor's commands. As mentioned erlier, when ROMPLUS+ is activated, the mini-assembler and the disassembler will not work.

Additionally, the control ROM uses seven bytes in the screen space. These locations are slot dependent, and they are summarized in the following table.

Symbolic Name	Byte Location	Usage
CHIP	$478+SLOT#	Contains active ROM socket # for CTRL-SHIFT-M commands
MODE	$4F8+Slot#	Used to parse commands
WHICH	$578+Slot#	Used to hold the entry point letter
CURCHIP	$5F8+Slot#	Contains number of most recently used ROM socket
TCHIP	$678+Slot#	A scratch location
SO	$6F8+Slot#	Contains the value (Slot # * 16)
MSLOT	$7F8	Contains the value ($CN where N=Slot#)

The control ROM has three entry points. Assuming that N = Slot number, the entry points are:

$CN00	Initial entry point, used when ROMPLUS+ is activated. It will initialize variables and I/O hooks.
$CN06	Output entry point. Vector here to output a character.
$CN08	Input entry point. Vector here to input a character.

The Branch Table

Every ROM that is to be used on ROMPLUS+ must have a branch table at the beginning of the ROM. The branch table allows the user to select an entry point into the ROM by using just a letter to designate the entry point. A summary of the branch table is as follows:

Address	
$C800	Address of output hook routine
$C802	Address of input hook routine
$C804	Value which indicates length of Branch table
$C805	Address for entry point #1
$C807	Address for entry point #2
$C805+(2*(n-1))	Address for entry point #n

All of the addresses are 2 bytes long, with the low order byte first. All branch tables must have at least one entry point. With only one entry point, the branch table would end at $C806 and the value of the byte at $C804 would be $07. The value contained at $C804 is the total number of bytes in the branch table. Therefore, if there are "N" entry point address, the value of $C804 is (2*N+5).

The input and output hook address ($C800 and $C802) are used by the CTRL-SHIFT-M command. $C800 contains the address of the routine to be called every time a character is to be outputed. This output hook address is usually the address of a routine on that particular ROM. $C802 contains the address of the routine on a particular ROM to be called every time a character is to be inputted. All character I/O routines should end with a return from subroutine instruction. If the ROM that you write does not use the CTRL-SHIFT-M command, then these I/O hooks will not point to a routine on the ROM. Instead, you should use the addresses of the standard Apple I/O drivers. The output hook, $C800, should contain the address $FDF0, with the low oder byte first. Likewise, the input hook, $C802, should contain the address $FD1B. These I/O hooks on the ROM must always point to valid I/O routine addresses.

The branch table is the only requirement for ROM's. The application program's code may begin immediately after the branch table.

Writing Your Own ROM

There are a few things you should remember when writing your own ROMs. First, your program should never reference location $CFFF. Any reference to that address will disable all memory that maps into $C800-$CFFF. If you do reference that address, you will disable ROMPLUS+.

The slot number of ROMPLUS+ may be found by your program by reading $7F8. It will contain the value $CN where N is the slot number. Location $6F8+N contains the value $N0.

The control ROM makes sure that RAM is active whenever a ROM socket is selected. If your program must deactivate the RAM, it <u>must</u> reactivate RAM before it finishes executing.

Programs On Two ROMs

The 2K bytes of storage on each ROM is large enough for all but the larger programs. If you have an application program that is larger than 2K bytes, there is a scheme allowing you to use two ROMs in conjunction.

ROMPLUS+ will map any <u>one</u> of the six ROMs into the $C800-$CFFF address space at one time. If you simply had the first ROM write a control word which switches the ROM socket number to the new ROM socket number, your program will immediately switch to the other ROM. This usually blows up the program.

One solution to this problem is to write a subroutine dispatching subroutine, and place this subroutine into <u>identical</u> addresses on the two ROMs. This way, you enter the subroutine dispatching subroutine on the first ROM, the switching of ROM occurs, and the dispatching routine continues on the second ROM, because the <u>identical</u> addresses contain <u>identical</u> code.

Here is the code which will do the task:

```
*The A register contains the ROM socket number
*you wish to use. The Y register contains a
*value which determines which routine is run (routine
*number *2). You must preserve the X register.

MSLOT     EQU   $7F8
CONTROL   EQU   $C080
CHIPNUM   EQU   $0
SUBADDR   EQU   $1
```

```
CHIPCALL  STA  CHIPNUM        :save ROM number
          LDX  MSLOT          :get $CN
          LDA  $638,X         :get $NO
          TAX                 :x contains value $NO
          LDA  CONTROL,X      :get control word
          ORA  #$08           :turn on activate bit 3
          PHA                 :save so we can restore
                               later
          AND  #$F8           :set ROM number to zero
          ORA  CHIPNUM        :or in new ROM number
          STA  CONTROL,X      :write to control word
```

At this point, we are now on the other ROM. Call routine specified by Y.

```
          LDA  SUBTABLE,X     :get low byte of address
          STA  SUBADDR        :and store here
          LDA  SUBTABLE+1,y   :get high byte of address
          STA  SUBADDR+1      :and store here
          JSR  CALLSUB        :indirect subroutine call
          PLA                 :return, get old state
          STA  CONTROL,X      :restore old ROM
          RTS                 :return out of this routine
CALLSUB   JMP  (SUBADDR)      :indirect jump to routine
SUBTABLE  DA   SUB1           :table of routine addresses
          DA   SUB2           :low byte first, high byte
                               second
```

It is necessary for this routine to be located at <u>identical</u> addresses on the two ROMs. Otherwise it will not work. SUBADDR may be located anywhere in memory as long as there are no possible memory conflicts. We recommend the page zero addresses of $1 and $2.

The program "CHIPCALL" is a subroutine, and should be called with the "JSR" instruction. Before you call the subroutine, set up the "A" and "Y" registers. The value of the X register must be preserved.

Chapter 4

REFERENCE

This chapter is a concise description of the hardware and software of ROMPLUS+. It is intended to serve as a reference section only.

The hardware features of ROMPLUS+ are:

1. Sockets for six 2K ROMs (2316) or EPROMs (2716). Total ROM capacity is 12K bytes. ROM is selected by software.

2. 256 bytes of RAM which can be enabled or disabled under software control.

3. Two TTL levels inputs which are held high by pull-up resistors. The inputs are read from the control word.

4. A 256 byte control ROM which controls the operation of ROMPLUS+.

The software features of ROMPLUS+ are summarized below:

1. ROMPLUS+ is activated by the "IN#n" or "PR#n" commands from BASIC. ROMPLUS+ is deactivated by <u>both</u> "IN#n" and "PR#n" commands, or by RESET, or by referencing location $CFFF.

2. There are two modes of operation available. These modes are selected by these commands:

   a) CTRL-SHIFT-M: This mode will run the selected ROM program every time a character is inputted or outputted.

   b) CTRL-SHIFT-N: This mode will run the selected ROM program immediately, and then return control to the calling program.

3. The command structure is:

   CTRL-SHIFT-M<ROM socket number><entry point>
   CTRL-SHIFT-N<ROM socket number><entry point>

   The "CTRL-SHIFT-letter" character is typed by holding down the CONTROL and SHIFT keys while typing either "M" or "N".

   <ROM socket number> is a value from 0 to 6, and selects

a ROM socket. ROM socket zero will disable all the ROMs without disabling ROMPLUS+. <entry point> is a character used to select the entry point into the ROM.

All ROMs must have at least one entry point. Entry point A is the first entry point, B is the second entry point, etc.

There are no spaces between the command character, the Ram socket number, and the entry point character. The brackets are not typed.

4. RAM is enabled and disabled by bit 7 of the control word. The top 256 bytes of any selected ROM is not available when RAM is enabled. If RAM is disabled by any ROM, then it must be enabled before the ROM returns.

Appendix A
CONTROL ROM SOURCE LISTING

General Information

The following pages contain the Control ROM Source listing. Once ROMPLUS+ has been activated, all input and output operations are vectored through the control ROM. This is transparent to the user. The Control ROM is looking for one of two special command characters (CTRL-SHIFT-M or CTRL-SHIFT-N.) If the character passed on input (or output) is not one of these special commands, then it is passed on to the input routine. If the character is one of the special commands, then the next two characters are interpreted as parameters of the Control ROM command.

NOTE

> The Control ROM makes use of two locations in memory that are normally used by the monitor. These two locations are $3A and $3B. Whenever the ROMPLUS+ card is active, the monitor "L" command for disassembly and the Apple II mini-assembler will not work properly.

Appendix A CONTROL ROM SOURCE LISTING

```
*         PRT ON
*
********************************
*
*   CONTROL PROM FOR MOUNTAIN
*   HARDWARE ROM BOARD
*
*       BY ANDY HERTZFELD
*
* (C) 1979 BY ANDY HERTZFELD
*
*
* VERSION 1.6, 4/16/79
*
********************************
*
*  EQUATES FOR SCREEN SPACE
*
MSLOT    EQU   $7F8
CHIP     EQU   $3B8
MODE     EQU   $438
WHICH    EQU   $4B8
CURCHIP  EQU   $538
TCHIP    EQU   $5B8
S0       EQU   $638
*
* MISC EQUATES
*
IORTS    EQU   $FF58
CSW      EQU   $36
STACK    EQU   $100
RDKEY    EQU   $FD1B
CHAROUT  EQU   $FDF0
BELL     EQU   $FBDD
CONTROL  EQU   $C080
ENTRIES  EQU   $C800
CHIPLIM  EQU   $C804
GOVECTOR EQU   $FEBC
PC       EQU   $3A
CTLA     EQU   $9D
CR       EQU   $8D
CTLB     EQU   $9E
SCTLA    EQU   $3A
*
*
         ORG   $6300
         OBJ   $6300
```

```
                *
                *
                *  WE USE 3 DIFFERENT ENTRY
                *  POINTS: "FIRST", FOR THE
                *  INITIAL ENTRY AND "OENTRY"
                *  AND "IENTRY" FOR THE OUTPUT
                *  AND INPUT RE-ENTRIES.  THE
                *  C AND V BITS ARE USED TO
                *  REMEMBER WHICH ENTRY OCCURED.
                *
6300:  2C 58 FF FIRST    BIT   IORTS  SET VFLAG FOR INITIAL ENTRY
6303:  38                SEC   ; MAKE INITIAL ENTRY OUTPUT
6304:  70 04             BVS   ENTRY  ALWAYS TAKEN
6306:  38       OENTRY   SEC
6307:  90                HEX   90      TRICK TO SAVE A BYTE
6308:  18       IENTRY   CLC   ;HIDE AS BRANCH OFFSET
6309:  B8                CLV
                *
                * COMMON ENTRY POINT
                *
630A:  48       ENTRY    PHA
630B:  8A                TXA
630C:  48                PHA
630D:  98                TYA
630E:  48                PHA
630F:  08                PHP
                *
                * NOW WE MUST FIND OUT WHAT SLOT
                * WE'RE IN.  THIS IS ACHIEVED BY
                * MAKING A DUMMY JSR WHICH WILL
                * LEAVE OUR ADDRESS ABOVE THE
                * STACK.  INTERRUPTS MUST BE
                * DISABLED.
                *
6310:  78                SEI
6311:  20 58 FF          JSR   IORTS  DUMMY JSR
6314:  BA                TSX
6315:  68                PLA
6316:  68                PLA
6317:  68                PLA
6318:  68                PLA   ;RECOVER INPUT CHARACTER
6319:  A8                TAY   ; AND KEEP IN Y REGISTER FOR NOW
631A:  CA                DEX
631B:  9A                TXS
631C:  68                PLA   ; GET $CN FROM STACK
631D:  8D F8 07          STA   MSLOT
6320:  AA                TAX   ; SLOT # IN X
6321:  0A                ASL
6322:  0A                ASL
6323:  0A                ASL
6324:  0A                ASL
6325:  9D 38 06          STA   S0,X
6328:  BD B8 03          LDA   CHIP,X
632B:  9D 38 05          STA   CURCHIP,X
                *
```

```
                *
                *  NOW RECOVER STATUS AND GO TO
                * THE PROPER ROUTINE ACCORDINGLY
                *
632E:  28                PLP   ; RE-ENABLE INTERRUPTS
632F:  08                PHP   ; SAVE STATUS
6330:  50 16             BVC   REENTRY
                *
                * THE FOLLOWING CODE IS FOR THE
                * INITIAL ENTRY ONTO THE BOARD.
                * WE INITIALIZE OUR VARIABLES
                * AND SET THE HOOKS TO POINT TO
                * THE RE-ENTRY POINT.
                *
6332:  AD F8 07 INIT     LDA   MSLOT
6335:  85 37             STA   CSW+1
6337:  85 39             STA   CSW+3
6339:  A9 06             LDA   #<OENTRY
633B:  85 36             STA   CSW
633D:  A9 08             LDA   #<IENTRY
633F:  85 38             STA   CSW+2
6341:  A9 00             LDA   #$00
6343:  9D B8 03          STA   CHIP,X
6346:  F0 35             BEQ   RESET   ALWAYS TAKEN
                *
                *
                * WE COME HERE FOR A RE-ENTRY.
                * WE CHECK FOR COMMANDS JUST
                * ON OUTPUT. AT THIS POINT THE
                * CARRY STILL MARKS WHERE
                * WE CAME FROM.
                *
6348:  B0 07    REENTRY  BCS   OUTHOOK
                *
                * SET WHICH TO INPUT HOOK
                *
634A:  A9 02             LDA   #$02
634C:  9D B8 04          STA   WHICH,X
634F:  D0 56             BNE   VECTOR   ALWAYS TAKEN
                *
                * HERE WE HANDLE THE OUTPUT HOOK.
                * WE SET WHICH AND UPDATE THE
                * CURRENT CHIP AND THEN GO CHECK
                * FOR COMMANDS.
                *
6351:  A9 00    OUTHOOK  LDA   #$0
6353:  9D B8 04          STA   WHICH,X
                *
                * THE FOLLOWING ROUTINE CHECKS
                * FOR THE CHIP INITIALIZATION
                * COMMAND.   IT IS CALLED ONLY
                * ON OUTPUT TO PREVENT THE SAME
                * CHARACTER FROM PASSING THROUGH
                * TWICE.  THE MODE VARIABLE KEEPS
                * TRACK OF OUR CURRENT STATE.
```

```
                      *
6356:  98             COMMAND  TYA
6357:  BC 38 04                LDY   MODE,X
635A:  30 0F                   BMI   GETNUM
635C:  D0 29                   BNE   GETINIT
                      *
635E:  C9 9D                   CMP   #CTLA
6360:  F0 04                   BEQ   SAVEMODE
6362:  C9 9E                   CMP   #CTLB
6364:  D0 41                   BNE   VECTOR
6366:  9D 38 04       SAVEMODE STA   MODE,X
6369:  F0 3C                   BEQ   VECTOR  ALWAYS TAKEN
                      *
                      * PARSE THE NUMBER, CHECKING TO
                      * MAKE SURE ITS FROM 0 TO 6.
                      *
636B:  49 B0          GETNUM   EOR   #$B0  MUST BE >=0
636D:  C9 07                   CMP   #$07
636F:  B0 09                   BCS   NOGOOD AND < 7
6371:  1E 38 04                ASL   MODE,X
6374:  9D B8 05                STA   TCHIP,X
6377:  D0 2E                   BNE   VECTOR  ALWAYS TAKEN
                      *
                      * THE FOLLOWING CODE HANDLES
                      * ERRORS BY RINGING THE BELL
                      * AND CANCELLING ANY PARTIAL
                      * COMMANDS. ITS IN THIS WEIRD
                      * PLACE BECAUSE OF THE 6502'S
                      * RELATIVE ADDRESSING CONSTRAINT.
                      *
6379:  48             NOGOOD2  PHA
637A:  20 DD FB       NOGOOD   JSR   BELL
                      *
637D:  A9 00          RESET    LDA   #$0
637F:  9D 38 05                STA   CURCHIP,X
6382:  9D 38 04                STA   MODE,X
6385:  F0 20                   BEQ   VECTOR  ALWAYS TAKEN
                      *
                      * HANDLE THE SELECTION PARAMETER
                      * BUT DON'T ERROR CHECK IT TILL
                      * THE CHIP IS ACTIVATED
                      *
6387:  0A             GETINIT  ASL   ; CARRY IS SET
6388:  E9 7D                   SBC   #$7D  ; 2*(A-5
638A:  9D B8 04       SETWHICH STA   WHICH,X
638D:  A9 00                   LDA   #$0
638F:  9D 38 04                STA   MODE,X
6392:  BD B8 05                LDA   TCHIP,X
6395:  9D 38 05                STA   CURCHIP,X
6398:  C0 3A                   CPY   #SCTLA
639A:  D0 0B                   BNE   VECTOR
639C:  9D B8 03                STA   CHIP,X
639F:  BC 38 06                LDY   S0,X
63A2:  09 88                   ORA   #$88
63A4:  99 80 C0                STA   CONTROL,Y
```

```
                 *
                 *
                 *
                 * THE FOLLOWING ROUTINE HANDLES
                 * THE VECTORING TO CHIP I/O HOOKS
                 * FIRST WE ENABLE THE SELECTED CHIP.
                 *
63A7:  28        VECTOR    PLP    ; RECOVER STATUS
63A8:  BC 38 06            LDY    S0,X
63AB:  B9 80 C0            LDA    CONTROL,Y
63AE:  48                  PHA
63AF:  AD FF CF            LDA    $CFFF  DISABLE OTHER ROMS
63B2:  BD 38 05            LDA    CURCHIP,X
63B5:  09 88               ORA    #$88
63B7:  99 80 C0            STA    CONTROL,Y
63BA:  68                  PLA
63BB:  8D 02 CF            STA    $CF02
63BE:  8C 03 CF            STY    $CF03
63C1:  BD 38 05            LDA    CURCHIP,X
63C4:  D0 0E               BNE    VECHOOK
                 *
                 * NO CHIP HAS BEEN ACTIVATED YET
                 * SO GO TO STANDARD KEYIN OR KEYOUT
                 *
63C6:  A9 FD               LDA    #>CHAROUT
63C8:  85 3B               STA    PC+1
63CA:  A9 F0               LDA    #<CHAROUT
63CC:  B0 02               BCS    ITSOUTPUT
63CE:  A9 1B               LDA    #<RDKEY
63D0:  85 3A     ITSOUTPUT STA    PC
63D2:  D0 12               BNE    EXIT    ALWAYS TAKEN
                 *
                 *
                 * NOW WE OBTAIN THE PROPER ADDRESS
                 * TO VECTOR TO BY INDEXING INTO
                 * THE INITIALIZATION TABLE ON THE
                 * CHIP. WE STORE THE ADDRESS
                 * IN LOCASL RAM AND THEN VECTOR
                 * THERE BY AN INDIRECT JUMP
                 *
63D4:  BC B8 04  VECHOOK   LDY    WHICH,X ; GET INDEX
63D7:  CC 04 C8            CPY    CHIPLIM
63DA:  B0 9D               BCS    NOGOOD2
63DC:  B9 01 C8            LDA    ENTRIES+1,Y
63DF:  85 3B               STA    PC+1
63E1:  B9 00 C8            LDA    ENTRIES,Y
63E4:  85 3A               STA    PC
```

xxxxxx

```
                *
                * NOW WE RESTORE REGISTERS AND GOTO
                * THE HOOK ROUTINE.
                *
63E6:  68       EXIT     PLA
63E7:  A8                TAY
63E8:  68                PLA
63E9:  AA                TAX
63EA:  68                PLA
63EB:  20 BC FE          JSR   GOVECTOR
                *
63EE:  48                PHA
63EF:  98                TYA
63F0:  48                PHA
63F1:  AC 03 CF          LDY   $CF03
63F4:  AD 02 CF          LDA   $CF02
63F7:  09 08             ORA   #$08
63F9:  99 80 C0          STA   CONTROL,Y
63FC:  68                PLA
63FD:  A8                TAY
63FE:  68                PLA
63FF:  60                RTS
                *
                *
                *    ALL DONE!
                *
--- END ASSEMBLY ---
TOTAL ERRORS: 00
256 BYTES OF OBJECT CODE
WERE GENERATED THIS ASSEMBLY.
```

Mountain Computer
INCORPORATED

Located in the Santa Cruz Mountains of Northern California. Mountain Computer, Inc. is a computer peripheral manufacturer dedicated to the production of use-oriented high technology products for the microcomputer. On-going research and development projects are geared to the continual supply of unique, innovative products that are easy to use and highly complementary in a broad variety of applications.