COCO III SECRETS REVEALED

SPECTRUM PROJECTS

COCO III SECRETS REVEALED is a filled with useful information and powerful secrets that will help you utilize new features of your Color Computer III. It will present several unique routines and show you some powerful features that were not available with the Color computer or Color Computer II.

COCO III SECRETS REVEALED does not require any knowledge of machine language. It should be noted that this book was written to explain the features of the new Coco III and since some of the new features are of a technical nature, there will be sections of this book that may not be fully understood by everyone.

TABLE OF CONTENTS

INTRODUCTION

The Color Computer has been around for quite a while in one form or another. The first version was known as the "C" board. Most people have never heard about this version; there were only a few hundred produced and it was quickly upgraded to what everybody thinks of as the first Color Computer, the "D" board.

This computer came with either 4 or 16K of memory, with Standard or Extended basic. It wasn't long before 16K of memory just wasn't enough, so someone figured out a way to install up to 32K of memory. Radio Shack followed suit and came out with the "E" board which was able to use 32K of memory without performing major surgery on the computer. As games became more complex, more memory was required so somebody discovered a way to increase the Coco's memory to 64K. Radio Shack again followed suit and produced the "F" board. In the years that followed the computer technology grew tremendously and a cost reduced version of the Color Computer appeared as the Color Computer II and Color Computer IIA.

During all of these changes, not once was anything done about improving the graphics capabilities. Oh sure, aritfacting was discovered, tricky use of the interrupts was used to manipulate the screens in new ways, however these were not actual changes in the computer, but instead the work of some extremely clever and creative programmers.

It is again time for an upgrade, a few companies have already come out with memory upgrades to 128K, 256K and even 512K. In the spirit of keeping a tradition going, Radio Shack has again followed suit, but this time they have gone a step or two further by addressing some of the other limitations of the Color Computer:

1. Only allowed the use of a standard television.
2. Only allowed 32 characters by 16 lines for text display.
3. True lower case was not available.
4. Graphics resolution was limited to 256 by 192.
5. Only 4 colors were available in the high resolution modes.
6. 64K was no longer enough memory.
7. No smooth scrolling abilities.
8. Double speed did not work on all computers.
9. Only one fire button per joystick.
10. Limited interrupt capability.

The Color Computer III is Radio Shack's solution to these limitations. Listed below are some of the Coco III's new and advanced features:

1. Three display interfaces are included, Standard TV, Composite monitor (monochrome or color) and Analog RGB.
2. Three character text modes are available, 32x16, 40x24 and 80x24.

3. True lower case is available in the 40 and 80 column modes however, basic still does not seem to be able to understand lower case commands.
4. Graphics resolution has increased to a maximum of 640x225.
5. Up to 16 colors can be displayed at a time and can be chosen from a palette of 64 different colors.
6. Memory starts at 128K and can be increased to 512K.
7. Smooth scrolling is available in both the horizontal and vertical directions.
8. Double speed is now available on all machines.
9. There are now two fire buttons per joystick.
10. The IRQ and FIRQ interrupts are each divided into six seperate sources. Programmable timer, horizontal border, vertical border, serial data, keyboard data and cartridge interrupts.
11. Along with all of these new features, the Coco III will still run about 90% of the current Color Computer software.

Almost anytime you add new features to something, you also create new problems to go along with them. This is true in this case also, listed below are a few of these. Most of the problems are minor ones but deserve to be mentioned.

1. Since artifacting is primarily a product of your video display, it is something that the computer cannot really correct. In the higher graphics modes, some detail in resolution is lost if you are using a TV or a composite monitor. If you are using an analog RGB monitor, the new graphics modes work very well however, the old modes which rely on artifacting to produce colors won't work since RGB monitors don't artifact.
2. In order to fit 40 and 80 column text on the screen, the screen area has been widened. This is great so far as the readability of the text is concerned, because the 40 column characters are exactly the same size as the 32 column characters, and the 80 column characters are a little bit larger than the old software driven 51 character screens. A problem occurs because when the screen was widened, it was also shifted slightly to the left. This has the annoying effect on some TV and Composite displays of shifting the first few characters off of the left side of the screen totaly out of view. THIS DOES NOT APPEAR TO EFFECT THE RGB MONITORS.
3. Smooth scrolling in the horizontal direction requires 48K of memory to allow proper wrap around. Neither the horizontal or vertical scrolling are supported by basic commands, Peek and Poke must be used.
4. Basic has not been re-written, the new commands have been patched into the old code. This point is both good and bad. On the good side, most of the existing software will work without modification, also since Basic is now always in RAM, it is very easy to add

patches and modifications of your own. On the bad side, any short commings that Basic originally had will still exist. For example, only 32K can be used for basic programs, the PCOPY bug still exists and PCLEAR 0 still is not allowed.

6. Because of the new interrupts, some multi-pak interfaces will require a small modification.

New Basic commands have been added to the existing commands by Microware Systems Corp. These new commands allow access to some of the Coco III's new features. Following is a brief list:

HSCREEN	PALETTE	HCLS	HPOINT
HLINE	HCOLOR	HPAINT	HDRAW
HSTAT	HBUFF	HSET	HRESET
HCIRCLE	HGET	HPUT	HPRINT
BUTTON	LOCATE	ATTR	WIDTH
LPEEK	LPOKE	ONERR	ONBRK
ERLN	ERNO		

These commands will be listed in a later chapter along with a brief explaination of what each one does. No attempt will be made and it is not the intention of this book to show you how to use these commands, most of them are high resolution counterparts of already existing commands with syntaxes that are the same or similar. The bulk of this book will concentrate on the new hardware features of the Color Computer III. Short Basic programs which use some of the new commands will be used to help further your understanding of these features.

CHAPTER 1
LET'S GET STARTED

In order to get started, there are a few things we need to know about the way the Color Computer III does things. The Coco III, as we said earlier, is capable of using 512K of memory but the CPU, a 68B09E, is an 8 bit microprocessor and can only directly address 64K of memory at a time. Because of this a special method known as MEMORY MANAGEMENT must be used. Memory management is a scheme which maps a block of memory into the CPU's 64K workspace when it is needed. In some systems this process is automatic, but in the Coco III this is not the case.

In the Color Computer III, the CPU's 64K bank or workspace is a seperate entity than that of the memory. This 64K workspace is divided into eight 8K slots by the Memory Management Unit (MMU) and each slot is controlled by an MMU register. The actual memory used in these slots is determined by the values stored in the MMU registers. Basic initializes these registers to the highest part of the 512K memory space (even if you only have a 128K system). This is actually the address range of $70000 to $7FFFF but the CPU will see it as $0000 to $FFFF.

The MMU registers are located in memory at $FFA0-$FFAF, please note that there are 16 registers but only 8 of them are needed to define the CPU's 64K workspace. This is because there are actually two sets of MMU registers that you can toggle between with the use of what is called the TASK REGISTER (TR). This method allows you to map the CPU's 64K workspace two different ways and quickly toggle between the two set ups by using the task register.

Each MMU register controls a specific 8K slot of the CPU's workspace as follows:

TR	MMU REGISTER	8K BLOCK
0	$FFA0	$0000-$1FFF
0	$FFA1	$2000-$3FFF
0	$FFA2	$4000-$5FFF
0	$FFA3	$6000-$7FFF
0	$FFA4	$8000-$9FFF
0	$FFA5	$A000-$BFFF
0	$FFA6	$C000-$DFFF
0	$FFA7	$E000-$FFFF
1	$FFA8	$0000-$1FFF
1	$FFA9	$2000-$3FFF
1	$FFAA	$4000-$5FFF
1	$FFAB	$6000-$7FFF
1	$FFAC	$8000-$9FFF
1	$FFAD	$A000-$BFFF
1	$FFAE	$C000-$DFFF
1	$FFAF	$E000-$FFFF

The Color Computer III's memory is also divided into 8K blocks. There are a total of 64 8K blocks of memory (0-$3F) and each one is referenced by a number. The first 8K block is block 0, the second is block 1 and so on. If you only have a 128K system then blocks 0-$F, $10-1F and $20-2F will be mirrors of blocks $30-$3F, in other words if you only have 128K and you try to place block 0, block $10 or block $20 into the CPU's memory space, block $30 will appear to be there instead. So in a 512K system you have memory blocks 0-$3F available for mapping into the CPU's workspace, but in a 128K system you only have blocks $30-$3F available.

Moving a block of memory into the CPU's workspace is done by simply placing the number of the block that you want into the CPU workspace slot of your choice. It is possible to put the same block of memory into more than one CPU memory slot. It is important to note that placing a new block of memory into one of the CPU's workspace slots, does not effect the information of the block that was in that slot previously. For example, if the CPU workspace slot controlled by the MMU register at $FFA0 currently contains a $38, and we replace it with block $39, the information stored in block $38 does not get hurt or destroyed in any way, it is simply moved out of the CPU's workspace and can be brought back at any time by storing a $38 into one of the CPU's memory slots (it could be the one at $FFA0, but it doesn't have to be). In a later chapter we will manipulate the MMU registers to allow a 32K graphics screen to be saved from basic.

PALETTE REGISTERS are another item that need a little explanation. On the old Color Computer, colors were generated on the graphics screen by placing the proper bit pattern on the screen for the color you wanted. The Color Computer III does this in nearly the same way, a bit pattern is still placed on the screen, but instead of this pattern defining the color, it points to a palette register. The value that is stored in the palette register is what actually defines the color. There are 16 palette registers available but the number of active ones is determined by the graphics color mode selected. In the 16 color modes all 16 registers are active, in the 4 color modes only 4 registers are active and in the 2 color modes just 2 registers are active. Regardless of the color mode, the active palette registers may be set to display any of the 64 different colors simply by storing the code for that color into the proper register. It is possible to store the same color into any or all of the active palette registers. Following is a list of the 64 available colors and their codes. Please note that the codes do not necessarily generate the same colors on a composite monitor as they do on an RGB monitor.

BINARY	HEX	DEC	PATTERN	RGB COLOR	COMPOSITE COLOR
00000000	00	0	.....	BLACK	BLACK
00000001	01	1	B0.....	DARK BLUE	DARK BLUE
00000010	02	2	..G0.....	DARK GREEN	DARK GREEN
00000011	03	3	B0G0.....	DARK CYAN	DARK CYAN
00000100	04	4	....R0.....	DARK RED	DARK RED
00000101	05	5	B0..R0.....	DARK MAGENTA	DARK MAGENTA
00000110	06	6	..G0R0.....	BROWN	BROWN
00000111	07	7	B0G0R0.....	DARK GREY	DARK GREEN/BLUE
00001000	08	8	......B1....	MEDIUM BLUE	DARK SKY BLUE
00001001	09	9	B0....B1....	BRIGHT BLUE	MEDIUM PEACOCK
00001010	0A	10	..G0..B1....	LGT BLUE/CYAN	MEDIUM GREEN/CYAN
00001011	0B	11	B0G0..B1....	LIGHT BLUE	DARK RED/MAGENTA
00001100	0C	12	....R0B1....	INDIGO	DARK RED/ORANGE
00001101	0D	13	B0..R0B1....	MED BLUE/PURPLE	DARK ORANGE
00001110	0E	14	..G0R0B1....	MED SKY BLUE	MED YELLOW/GREEN
00001111	0F	15	B0G0R0B1....	MEDIUM PEACOCK	MEDIUM BLUE/PURPLE
00010000	10	16	.....G1..	MEDIUM GREEN	DARK GREY
00010001	11	17	B0.....G1..	MED GREEN/CYAN	MEDIUM BLUE
00010010	12	18	..G0....G1..	BRIGHT GREEN	MEDIUM GREEN
00010011	13	19	B0G0....G1..	BRIGHT GRN/CYAN	MEDIUM CYAN
00010100	14	20	....R0..G1..	MED YELLOW/GRN	MEDIUM RED
00010101	15	21	B0..R0..G1..	LIGHT GRN/CYAN	MED BLUE/MAGENTA
00010110	16	22	..G0R0..G1..	BRGT YELLOW/GRN	YELLOW/BROWN
00010111	17	23	B0G0R0..G1..	LIGHT GREEN	MEDIUM GREEN/BLUE
00011000	18	24	......B1G1..	MEDIUM CYAN	MEDIUM SKY BLUE
00011001	19	25	B0....B1G1..	PEACOCK	BRIGHT PEACOCK
00011010	1A	26	..G0..B1G1..	LIGHT GRN/CYAN	BRIGHT GREEN/CYAN
00011011	1B	27	B0G0..B1G1..	BRIGHT CYAN	MEDIUM RED/MAGENTA
00011100	1C	28	....R0B1G1..	LIGHT PEACOCK	MEDIUN RED/ORANGE
00011101	1D	29	B0..R0B1G1..	PALE PEACOCK	MEDIUM ORANGE
00011110	1E	30	..G0R0B1G1..	PALE GRN/CYAN	BRIGHT YELLOW/GRN
00011111	1F	31	B0G0R0B1G1..	LIGHT CYAN	BRIGHT PURPLE
00100000	20	32	.....R1	MEDIUM RED	LIGHT GREY
00100001	21	33	B0.....R1	MED RED/MAGENTA	BRIGHT BLUE
00100010	22	34	..G0.....R1	YELLOW/ORANGE	BRIGHT GREEN
00100011	23	35	B0G0.....R1	LIGHT RED	BRIGHT CYAN
00100100	24	36	....R0....R1	BRIGHT RED	BRIGHT RED
00100101	25	37	B0..R0....R1	LGT RED/MAGENTA	BRIGHT MAGENTA
00100110	26	38	..G0R0....R1	ORANGE	MEDIUM YELLOW
00100111	27	39	B0G0R0....R1	PALE RED/MAGENTA	BRIGHT GREEN/BLUE
00101000	28	40	......B1..R1	MED BLUE/MAGENTA	BRIGHT SKY BLUE
00101001	29	41	B0....B1..R1	BLUE/PURPLE	LIGHT PEACOCK
00101010	2A	42	..G0..B1..R1	LIGHT MAGENTA	LIGHT GREEN/CYAN
00101011	2B	43	B0G0..B1..R1	PURPLE	BRIGHT RED/MAGENTA
00101100	2C	44	....R0B1..R1	LIGHT PURPLE	BRIGHT ORANGE
00101101	2D	45	B0..R0B1..R1	BRIGHT MAGENTA	BRGT YELLOW/ORANGE
00101110	2E	46	..G0R0B1..R1	PALE BLUE/MAGEN	LIGHT YELLOW/GREEN
00101111	2F	47	B0G0R0B1..R1	PALE PURPLE	LIGHT PURPLE
00110000	30	48	.....G1R1	MEDIUM YELLOW	WHITE
00110001	31	49	B0.....G1R1	LIGHT YELLOW	LIGHT BLUE

BINARY	HEX	DEC	PATTERN	RGB COLOR	COMPOSITE COLOR
00110010	32	50	..GO....G1R1	LGT YELLOW/GRN	LIGHT GREEN
00110011	33	51	BOGO....G1R1	PALE YELLOW/GRN	LIGHT CYAN
00110100	34	52	....RO..G1R1	LGT YELLOW/ORANG	LIGHT RED
00110101	35	53	BO..RO..G1R1	MEDIUM YELLOW	LIGHT BLUE/MAGENTA
00110110	36	54	..GORO..G1R1	BRIGHT YELLOW	LIGHT YELLOW
00110111	37	55	BOGORO..G1R1	PALE YELLOW	LIGHT GREEN/BLUE
00111000	38	56	......B1G1R1	LIGHT GREY	LIGHT SKY BLUE
00111001	39	57	BO....B1G1R1	PALE BLUE	PALE PEACOCK
00111010	3A	58	..GO..B1G1R1	PALE CYAN	PALE GREEN/CYAN
00111011	3B	59	BOGO..B1G1R1	PALE BLUE/CYAN	LIGHT RED/MAGENTA
00111100	3C	60	....ROB1G1R1	PALE RED	LIGHT ORANGE
00111101	3D	61	BO..ROB1G1R1	PALE MAGENTA	LIGHT YELLOW/ORANG
00111110	3E	62	..GOROB1G1R1	VERY PALE YELLOW	PALE YELLOW/GREEN
00111111	3F	63	BOGOROB1G1R1	WHITE	PALE PURPLE

Short Basic programs will be presented in later chapters which will demonstrate how the palette registers can be put to work in some powerful ways.

CHAPTER 2
NEW COMMANDS

To help make use of the Color Computer III's enhanced features, a set of new commands has been added. Basically the commands deal in 3 areas, grahpics, character display and miscellaneous enhancements. Each new command, along with a brief description of the function it performs, will be discussed in this chapter. The commands will be broken into the three groups listed above and presented in that order.

GRAPHICS:

PALETTE R,C - Places the color code indicated by "C" into the palette register indicated by "R". The command PALETTE 15,63 would place the color code for white into palette register 15. Instead of "R" and "C", the words CMP or RGB may be used to set up system defaults for composite (CMP) or RGB monitors.

HSCREEN M - Activates and displays the graphics mode selected by "M". 0 = text mode, 1 = the 320x192 4 color mode, 2 = the 320x192 16 color mode, 3 = the 640x192 2 color mode, 4 = the 640x192 4 color mode.

HCLS R - Clears the graphics screen to the palette specified by "R". The actual color of the screen will be determined by the color code stored in that palette register.

HCOLOR F,B - Sets the Foreground and Background defaults to the palettes specified by "F" and "B". The palettes specified will be used as defaults during certain graphics commands such as line and circle if no palettes are specified.

HSET (X,Y,R) - Sets the point at horizontal coordinate X, vertical coordinate Y to the palette specified by "R". If "R" is omitted, the foreground palette specified by the HCOLOR command will be used.

HRESET (X,Y,R) - Sets the point at horizontal coordinate X, vertical coordinate Y to the palette specified by "R". If "R" is omitted, the background palette specified by the HCOLOR command will be used.

HPOINT (X,Y) - Returns the palette value located at the horizontal coordinate X and vertical coordinate Y. HPOINT is considered a function because it returns a result to Basic rather than performing an action. The proper syntax for this command is A=HPOINT(X,Y) where "A" is the variable that will contain the result and "X" and "Y" are the horizontal and vertical coordinates.

HLINE - Draws a line from X1,Y1 to X2,Y2. The syntax for this command is the same as the LINE command of Extended Basic.

HDRAW - Allows you to draw a shape by giving an imaginary graphics cursor direction and color instructions. The syntax for this

command is the same as the DRAW command of Extended Basic.

HCIRCLE - Allows a circle to be drawn on the screen. The syntax for this command is the same as the CIRCLE command of Extended Basic.

HPAINT - Allows an area on the screen to be filled with a palette. The syntax for this command is the same as the PAINT command of Extended Basic.

HPRINT (X,Y),"STRING" - Allows text messages to be displayed on the graphics screen. X and Y are the horizontal and vertical coordinates at which to start. "STRING" is the message to be printed, up to 40 characters (80 for HSCREEN 4) may be displayed on a line. The character color and the color of it's background is determined by the foreground and background colors set with the HCOLOR command.

HBUFF N,A - Reserves a memory buffer for HGET and HPUT where "A" is the number of bytes to reserve and "N" is the buffer number. This method is used instead of dimensioning an array to reserve space (Buffer is limited to 8K).

HGET - Gets an area of screen memory and places it in the buffer specified. The syntax for this command is the same as the GET command of Extended Basic.

HPUT - Takes the screen memory that was saved by HGET and puts it onto the screen at the coordinates specified. The syntax for this command is the same as the PUT command of Extended Basic.

TEXT COMMANDS:

WIDTH W - Changes the character width of the display to the value specified by "W". Legal values are 32, 40 and 80.

LOCATE X,Y - This command is used instead of the PRINT@ statement of Basic for the 40 and 80 column screens. The cursor will be positioned at the horizontal and vertical coordinates specified by "X" and "Y".

HSTAT A$,A,X,Y - Returns the X/Y position, the attributes and the character located at the current cursor position.

ATTR F,C,B,U - Sets the attributes of the character located at the current cursor position. Foreground color, Background color, Blink and Underline options may be specified.

MISCELLANEOUS COMMANDS

BUTTON - Returns the status of the specified joystick button. The command A=BUTTON(0) will return the status of button 0. 0=Right button 1, 1=Right button 2, 2=Left button 1, 3=Left button 2.

ONERR - Allows the user to trap system errors. The command ONERR

GOTO 100 would transfer program control to line 100 anytime a system error occurs.

ERNO - Contains the number of the system error that just occured.

ERLN - Contains the line number where the last system error occured.

ONBRK - Allows the user to trap the break key. ONBRK GOTO 1000 would transfer control to line 1000 when the BREAK key is pressed.

LPEEK - Allows peek access into the entire 512K memory range.

LPOKE - Allows poke access into the entire 512K memory range.

Modifying old programs to work with the new graphics and text features if the Color Computer III is NOT a very difficult job. In the case of most graphics commands, it is just a matter of adding an "H" in front of the old command. PAINT becomes HPAINT, DRAW becomes HDRAW, CIRCLE becomes HCIRCLE and so on. In some cases other changes must also be made, for example PRINT@ will not work on the 40 or 80 column screens, LOCATE must be used instead, another example would be when using HGET and HPUT, instead of dimensioning an array to hold the graphics information, HBUFF must now be used to reserve this space.

The following program, CC2TOCC3.BAS, will aid in converting your old programs. It won't do everything, but it will handle the majority of the work and will flag out most of the problem areas. CC2TOCC3.BAS works with DISK ONLY, it reads in a normally saved basic file and writes out a converted ASCII file. Notice that the command LPEEK was included twice in the area for the new secondary functions, this is not a mistake. Due to a bug in basic, the new secondary functions skip token 168 and start with 169, the first LPEEK is simply a dummy to take this error into account.

```
10 CLS:CLEAR1500:DIM TK$(120),SF$(45):TK=120:SF=45
40 PA=0 'set to 1 if print@ to be left alone
50 FORX=0TOTK:READTK$(X):NEXTX:FORX=0TOSF:READSF$(X):NEXTX
70 LINEINPUT"ENTER FILENAME>";FL$
80 IF FL$=""THEN END
90 IF LEN(FL$)<=4 AND LEFT$(FL$,3)="DIR" THEN
   A=VAL(RIGHT$(FL$,1)):DIR A:GOTO70
100 CLS:PRINT:PRINT"SCREEN, PRINTER OR DISK(S/P/D)?"
110 A$=INKEY$:IF A$=""THEN110
120 IF NOT(A$="S" OR A$="P" OR A$="D") THEN110
130 IF A$="P" THEN DN=-2 ELSE IF A$="S" THEN DN=0 ELSE DN=2
140 FX$=LEFT$(FL$+"          ",8):EXT$="BAS"
150 IF A$="D" THEN LINEINPUT"OUTPUT FILENAME>";FO$:IF FO$=""
    THEN FO$="OUTFILE"
160 GOSUB900:IF FL=0 THEN PRINT:PRINTFL$"."EXT$;
    " NOT FOUND...":PRINT:GOTO70
170 OPEN"D",#1,FL$+".BAS",1
180 FIELD#1,1ASBY$
190 OPEN"O",#DN,FO$+".BAS"
```

```
200 EN=LOF(1)-1
210 'set flag (FL) to zero if basic program is tokenized,
    set to one if ascii
220 GET#1,1:A$=BY$:IF ASC(A$)=255 THEN FL=0 ELSE FL=1
230 ON FL GOTO 400
240 X=4:AD=1
250 IF AD=1 THEN GOSUB410:IF EX THEN 380
260 GET#1,X:A=ASC(BY$):A$=CHR$(A)
270 IF A$=CHR$(0) THEN A$=CHR$(13):A=13:AD=1
280 IF A$=":" THEN GOSUB850
290 IF A=255 THEN GOSUB880:GOTO370
300 IF A=128+7 THEN GOSUB1020
310 IF A=128+59 THEN GOSUB990
320 IF A=128+68 THEN GOSUB1040
330 IF A=128+61 THEN GOSUB1060
340 IF A=128+62 THEN GOSUB1060
350 IF A>=128 THEN A=A-128:PRINT#DN,TK$(A);:GOTO370
360 PRINT#DN,A$;
370 X=X+1:IF INKEY$<>"Q" THEN GOTO250
380 CLOSE
390 PRINT:GOTO70
400 PRINT"FILE NOT TOKENIZED":GOTO380
410 IF X=EN THEN EX=1:GOTO430 ELSE EX=0
420 X=X+2:GET#1,X:A$=BY$:A=ASC(A$):X=X+1:GET#1,X:A$=BY$:
    B=ASC(A$):A=A*256+B:A$=MID$(STR$(A),2,LEN(STR$(A)))
    :PRINT#DN,A$;" ";:X=X+1:AD=0
430 RETURN
435 'Start of Basic's commands
440 DATA FOR,GO,REM,REM,ELSE,IF,DATA,PRINT,ON,INPUT
460 DATA END,NEXT,DIM,READ,RUN,RESTORE,RETURN,STOP
480 DATA POKE,CONTINUE,LIST,CLEAR,NEW,CLOAD,CSAVE
500 DATA OPEN,CLOSE,LLIST,SET,RESET,CLS,MOTOR,SOUND
520 DATA AUDIO,EXEC,SKIPF,TAB(,TO,SUB,THEN,NOT
540 DATA STEP,OFF,+,-,*,/,^,AND,OR,>,=,<
560 'Start of Extended Basic's commands
570 DATA DEL,EDIT,TRON,TROFF,DEF,LET,LINE,HCLS,SET
590 DATA RESET,**HSCREEN,**PCLEAR,HCOLOR,HCIRCLE,HPAINT,GET
610 DATA HPUT,HDRAW,**PCOPY,**PMODE,PLAY,DLOAD,RENUM,FN
630 DATA USING
640 'Start of Disk Basic's commands
650 DATA DIR,DRIVE,FIELD,FILES,KILL,LOAD,LSET,MERGE
670 DATA RENAME,RSET,SAVE,WRITE,VERIFY,UNLOAD,DSKINI
690 DATA BACKUP,COPY,DSKI$,DSKO$,DOS
695 'CoCo III's commands start here
696 DATA WIDTH,PALETTE,HSCREEN,LPOKE,HCLS,HCOLOR
697 DATA HPAINT,HCIRCLE,HLINE,HGET,HPUT,HBUFF,HPRINT
698 DATA ERR,BRK,LOCATE,HSTAT,HSET,HRESET,HDRAW
699 DATA CMP,RGB,ATTR
710 'Start of Basic's secondary functions
720 DATA SGN,INT,ABS,USR,RND,SIN,PEEK,LEN,STR$,VAL,ASC
740 DATA CHR$,EOF,JOYSTK,LEFT$,RIGHT$,MID$,POINT
760 DATA INKEY$,MEM
770 'Start of Extend Basic's secondary functions
780 DATA ATN,COS,TAN,EXP,FIX,LOG,POS,SQR,HEX$,VARPTR
```

```
800 DATA INSTR,TIMER,HPOINT,STRING$
820 'Start of Disk Basic's secondary functions
830 DATA CVN,FREE,LOC,LOF,MKN$,AS
835 'CoCo III's secondary functions
836 DATA LPEEK,LPEEK,BUTTON,HPOINT,ERNO,ERLIN
840 'End of tokens
850 GET#1,X+1:T=ASC(BY$):T$=CHR$(T)
860 IF T=&H83 OR T=&H84 THEN X=X+1:A$=T$:A=T
870 RETURN
880 X=X+1:GET#1,X:A=ASC(BY$):A$=CHR$(A)
890 A=A-128:PRINTSF$(A);:RETURN
900 FL=0:FORS=3 TO 17:DSKI$0,17,S,A$,B$
910 FORM=1TOLEN(A$)STEP32
920 IF MID$(A$,M,11)=FX$+EXT$ THEN FL=1
930 NEXTM
940 FORM=1TOLEN(B$)STEP32
950 IF MID$(B$,M,11)=FX$+EXT$ THEN FL=1
960 NEXTM
970 IF FL=1 THEN S=17
980 NEXTS:RETURN
990 GET#1,X+1:T=ASC(BY$)
1000 IF T<>128+9 THEN PRINT#DN,"H";
1010 RETURN
1020 IF PA=0 THEN GET#1,X+1:IF BY$="@"THEN PRINT#DN,"**";
1030 RETURN
1040 GET#1,X+1:IF BY$<>"#" THEN PRINT#DN,"H";
1050 RETURN
1060 GET#1,X+1:IF BY$="("THEN PRINT#DN,"H"; ELSE PRINT#DN,"P";
1070 RETURN
```

CHAPTER 3
PLAYING WITH PALETTES

One of the nicest new features of the Color Computer III is the ability to display your choice of 64 different colors, 16 at a time on a high resolution screen.

On the Color Computer and Color Computer II, each pixel (picture element) was defined by two bits (1/4 of a byte). If we use two bits to count in binary, the most different combinations we can attain is 4 (00, 01, 10, 11). These bit pairs were hard wired to a specific color and the only time that a color could change was if they all changed by switching color modes.

In the Coco III 4 color mode, the description would be the same except that the colors are not hard wired anymore. Instead of the bit pairs defining a color, they point to a byte in the I/O space called a palette register. Each palette register may be programed individually with one of 64 different color codes. In the 2 color mode, two palette registers are used, in the 4 color mode, 4 of the palette registers are used and in the 16 color mode, all 16 of the palette registers are used. (The 2 color mode uses one bit per pixel, the four color mode uses 2 bits per pixel and the 16 color mode uses 4 bits per pixel).

The palette registers are located in memory from $FFB0 to $FFBF and may be set by either POKEing new values or by using Basic's PALETTE command. The only way to read what is stored in a palette register is to use the PEEK command and AND the result with 63. PEEK($FFB1) AND 63 would read the value stored in palette register 1.

Any palette register may contain any color at anytime. If desired, all palette registers may be set to the same thing, Changing a palette register to a new color will cause all pixels pointing to that palette to instantly change to that color. The following Basic program called CIRCLES will draw 4 circles on the screen in different palettes and set them all to the same color. It will then wait for the keys "1", "2", "3" or "4" to be pressed, each key will turn on a different circle when pressed and turn it off when released. Notice in line 110 that the last statement is CMP. This is the same as the command PALETTE CMP. Also take a peek at lines 2 and 3 for some useful POKES.

```
1 ONBRK GOTO110
2 'Set computer to double speed, disable HCLS during HSCREEN
3 POKE&HFFD9,0:POKE&HE6C6,18:POKE&HE6C7,18
10 HSCREEN 2:HCLS8:DIM OP(4)
15 'Draw the circles
20 FOR X=1 TO 4:HCIRCLE(X*40+60,192/2),15,X:
   HPAINT(X*40+60,192/2),X,X:NEXTX
25 'Get current values for palette registers 1-4
30 FOR X=1 TO 4:OP(X)=PEEK(&HFFB0+X)AND 63:NEXTX
35 'Set palettes 1-4 to black
40 FOR X= 1 TO 4:PALETTE X,0:NEXTX
41 'Set colors for Hprint and print message
42 HCOLOR 11,0:HPRINT (11,1),"PRESS 1, 2, 3 OR 4"
```

```
45 'Wait for keys and respond accordingly
50 A$=INKEY$
51 'Cause the keys to repeat and do counter for message blink
52 FORPJ=0TO7:POKE&H152+PJ,255:NEXTPJ:LL=LL+1:IF LL<4 THEN60
53 'Blink characters by changing the palette value
54 LL=0:MB=1-MB:IF MB THEN PALETTE11,0 ELSE PALETTE 11,63
58 'Act on keyboard response
60 IF A$="1" THEN PALETTE 1,OP(1) ELSE PALETTE 1,0
70 IF A$="2" THEN PALETTE 2,OP(2) ELSE PALETTE 2,0
80 IF A$="3" THEN PALETTE 3,OP(3) ELSE PALETTE 3,0
90 IF A$="4" THEN PALETTE 4,OP(4) ELSE PALETTE 4,0
100 IF A$<>"Q" THEN 50
108 'Go back to single speed and restore palette defaults
110 POKE&HFFD8,0:CMP
```

The next example is a little more elaborate and involves a fairly long program. It will build a large ball onto the graphics screen and make it appear to rotate by simply changing the palette registers through a series of colors. To save space, and typing time, the DATA statements from lines 310 to 720 only contain the top left corner of the ball, the program will take this data and mirror it into a complete ball. The DATA statements from lines 240 to 290 contain the ball pattern information and may be changed to create different patterns on the ball. Notice that the last value of each line is an "F", this determines the palette used for the background area of the ball and should not change. Each of these values is the number of the palette register to use for that area of the ball.

```
1 POKE&HFFD9,0 'Double speed
5 WIDTH32
10 PCLEAR1:CLS:PRINT"        DEMO BALL GENERATOR "
20 CLEAR10000:DIM RW$(6,25),A$(42),CL(11,11)
21 ONBRK GOTO 1030
30 FORX=1TO6:FORY=1TO25:READ RW$(X,Y):NEXTY,X
40 READ A$:IF A$<>"END1" THEN PRINT "BALL DEFINITION ERROR!":END
50 M=1:FORX=1TO42:READ A$(X)
60 A$="":FORJM=LEN(A$(X))TO 1 STEP-1:A$=A$+MID$(A$(X),JM,1)
   :NEXTJM:GOSUB730:A$(X)=A$(X)+B$+"YY"
70 PRINT@32*4,"READING ROW"X:NEXTX
75 HSCREEN2:HCLS 15
80 BW=88:BD=84:SA=&H60000:HO=128/2-(BW/2):VO=192/2-(BD/2)
90 FORL=1TO42:GOSUB180:NEXTL
100 FORL=42TO1STEP-1:GOSUB180:NEXTL:GOTO230
170 ''''' subroutine '''''
180 SC=INT((L-1)/14)+1:M=M+1
190 B$="":FORX=1TOLEN(A$(L))
    :B$=B$+RW$(SC,ASC(MID$(A$(L),X,1))-ASC("A")+1)
    :NEXTX:FORX=1TOLEN(B$)STEP2
    :LPOKE SA+(VO*160)+HO+INT(X/2),VAL("&H"+MID$(B$,X,2))
    :NEXTX:VO=VO+1
200 RETURN
210 ''' end of subroutine '''
230 GOTO 1000
```

```
240 DATA 0,1,2,3,4,5,6,7,8,9,A,B,0,1,2,3,4,5,6,7,8,9,A,B,F
250 DATA 3,4,5,6,7,8,9,A,B,0,1,2,3,4,5,6,7,8,9,A,B,0,1,2,F
260 DATA 6,7,8,9,A,B,0,1,2,3,4,5,6,7,8,9,A,B,0,1,2,3,4,5,F
270 DATA 9,A,B,0,1,2,3,4,5,6,7,8,9,A,B,0,1,2,3,4,5,6,7,8,F
280 DATA 0,1,2,3,4,5,6,7,8,9,A,B,0,1,2,3,4,5,6,7,8,9,A,B,F
290 DATA 3,4,5,6,7,8,9,A,B,0,1,2,3,4,5,6,7,8,9,A,B,0,1,2,F
300 DATA END1
310 DATA YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYEEFGHIL
320 DATA YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYCDDEEFFGHIJK
330 DATA YYYYYYYYYYYYYYYYYYYYYYYYYYYYYBBCCDDEEFFGHHIJL
340 DATA YYYYYYYYYYYYYYYYYYYYYYYYYYYBBBCCDDEEFFGGHIJKL
350 DATA YYYYYYYYYYYYYYYYYYYYYYYYYABBCCCDDEEFFGGHHIJKL
360 DATA YYYYYYYYYYYYYYYYYYYYYYYAABBCCCDDEEFFGGHHIIJKL
370 DATA YYYYYYYYYYYYYYYYYYYYYAABBBCCCDDEEFFGGHHIIJKKL
380 DATA YYYYYYYYYYYYYYYYYYYYAABBBCCCDDEEFFGGGHHIIJKKL
390 DATA YYYYYYYYYYYYYYYYYYAAABBBCCCDDEEFFFGGHHIIJJKLL
400 DATA YYYYYYYYYYYYYYYYYAAABBBCCCDDEEFFFGGHHIIIJKKLL
410 DATA YYYYYYYYYYYYYYYYAAABBBCCCDDEEEFFGGGHHIIJJKKLL
420 DATA YYYYYYYYYYYYYYAAABBBCCCDDDEEFFGGGHHHIIJJKKLL
430 DATA YYYYYYYYYYYYYAAABBBCCCDDDEEFFFGGGHHIIIJJKKLL
440 DATA YYYYYYYYYYYYAAABBBBCCCDDEEEFFGGGHHHIIJJJKKLL
450 DATA YYYYYYYYYYYAAABBBBCCCDDEEEFFGGGGHHIIIJJJKKLL
460 DATA YYYYYYYYYYAAAABBBCCCDDDEEEFFGGGHHHIIIJJKKKLL
470 DATA YYYYYYYYYAAAABBBCCCDDDEEEFFFGGGHHIIIIJJKKKLL
480 DATA YYYYYYYYAAAABBBCCCDDDEEEFFGGGHHHIIIJJJKKLLL
490 DATA YYYYYYYAAAABBBCCCDDDEEEFFFGGGHHHIIIJJJKKLLL
500 DATA YYYYYYYAAABBBBCCCDDDEEEFFFGGGHHHIIIJJJKKLLL
510 DATA YYYYYYAAAABBBCCCDDDEEEFFFGGGHHHIIIJJJKKKLLL
520 DATA YYYYYAAAABBBCCCCDDDEEEFFFGGGHHHIIIJJJKKKLLL
530 DATA YYYYYAAABBBBCCCDDDEEEFFFGGGHHHHIIIJJJKKKLLL
540 DATA YYYYAAAABBBCCCCDDDEEEFFFGGGHHHHIIIJJJKKKLLL
550 DATA YYYYAAAABBBCCCDDDEEEEFFGGGGHHHIIIIJJJKKKLLL
560 DATA YYYAAAABBBCCCCDDDEEEFFFGGGGHHHIIIIJJJKKKLLL
570 DATA YYYAAAABBBCCCDDDEEEEFFFGGGHHHHIIIJJJKKKKLLL
580 DATA YYYAAABBBCCCCDDDEEEEFFGGGGHHHHIIIJJJKKKKLLL
590 DATA YYAAAABBBCCCDDDDEEEFFFGGGGHHHIIIIJJJKKKKLLL
600 DATA YYAAABBBBCCCDDDEEEEFFFGGGGHHHIIIIJJJKKKKLLL
610 DATA YYAAABBBCCCCDDDEEEEFFFGGGHHHHIIIIJJJKKKKLLL
620 DATA YAAAABBBCCCDDDDEEEFFFFGGGHHHHIIIIJJJKKKKLLL
630 DATA YAAAABBBCCCDDDEEEEFFFGGGGHHHHIIIJJJJKKKKLLL
640 DATA YAAABBBCCCCDDDEEEEFFFGGGGHHHHIIIJJJJKKKKLLL
650 DATA YAAABBBCCCCDDDEEEEFFFGGGGHHHHIIIJJJKKKKKLLL
660 DATA AAAABBBCCCDDDDEEEEFFFGGGGHHHIIIIJJJKKKKLLLL
670 DATA AAAABBBCCCDDDDEEEFFFFGGGHHHHIIIJJJJKKKKLLLL
680 DATA AAAABBBCCCDDDDEEEFFFFGGGGHHHIIIJJJJKKKKLLLL
690 DATA AAABBBBCCCDDDEEEEFFFFGGGGHHHIIIJJJJKKKKLLLL
700 DATA AAABBBCCCCDDDEEEEFFFFGGGGHHHIIIJJJJKKKKLLLL
710 DATA AAABBBCCCCDDDEEEEFFFFGGGGHHHIIIJJJJKKKKLLLL
720 DATA AAABBBCCCCDDDEEEEFFFFGGGGHHHIIIJJJJKKKKLLLL
730 B$="":FORC=1TOLEN(A$):A=ASC(MID$(A$,C,1))-1
    :IF A=ASC("Y")-1THEN A=A+1:GOTO740 ELSE A=88-(A-64)
740 B$=B$+CHR$(A):NEXTC:RETURN
999 ' NOW THAT THE BALL IS BUILD, MAKE IT SPIN
1000 FOR P=0 TO 11:FOR C=0 TO 11:READ CL(P,C):NEXT C,P
```

```
1010 FOR P=0 TO 11:FOR C=0 TO 11:POKE&HFFB0+C,CL(P,C)
1020 NEXT C,P:IF INKEY$<>CHR$(13) THEN 1010
1030 POKE&HFFD8,0:CMP:END
1050 DATA 9,9,9,26,26,26,16,16,16,63,63,63
1060 DATA 63,9,9,9,26,26,26,16,16,16,63,63
1070 DATA 63,63,9,9,9,26,26,26,16,16,16,63
1080 DATA 63,63,63,9,9,9,26,26,26,16,16,16
1090 DATA 16,63,63,63,9,9,9,26,26,26,16,16
2000 DATA 16,16,63,63,63,9,9,9,26,26,26,16
2010 DATA 16,16,16,63,63,63,9,9,9,26,26,26
2020 DATA 26,16,16,16,63,63,63,9,9,9,26,26
2030 DATA 26,26,16,16,16,63,63,63,9,9,9,26
2040 DATA 26,26,26,16,16,16,63,63,63,9,9,9
2050 DATA 9,26,26,26,16,16,16,63,63,63,9,9
2060 DATA 9,9,26,26,26,16,16,16,63,63,63,9
```

All color displayed on the screen is controlled by the palette registers, this is true even for the text modes. In the Color Computer compatible mode's 32X16 screen format, the background screen color is controlled by palette 13 and the color of the text is controlled by palette 12. This is the only Coco III text screen where these registers are forced. In the 40 or 80 column screens the ATTR command may be used to point the text and individual character backgrounds to different palettes. In fact the 40 and 80 column screens are set up entirely different, instead of one byte per text character, the 40 and 80 column screens use two. All even numbered bytes contain the value of the character to display, and the odd bytes contain the attributes for that character. The attribute bit definitions are defined in chapter 5.

The 40 and 80 column screens are located in memory at address $6C000 and is only moved into the CPU's 64K workspace when a character needs to be printed. This nice because it means that the high resolution text screens do not use any of Basic's program space. The following routine will LPOKE all of the available characters onto the text screen.

```
10 WIDTH 40:LOCATE0,10
20 AT=16
30 FOR X=0 TO 510 STEP 2
40 LPOKE &H6C000+X,INT(X/2):LPOKE &H6C000+X+1,AT
50 NEXTX
```

The second portion of line 40 pokes in the attribute byte for the character preceding it, you can easily play around with this by changing the value of AT which is set in line 20.

It should be noted that the blink rate of a character that has the blink attribute bit set is controlled by the programmable timer interrupt ($FF94 and $FF95). If both of these bytes are zero'd, the characters will not blink.

The following program called "CC3WORD" will give you an example of how the 40 and 80 column text mode commands may be used to create some very powerful programs with almost no effort. "CC3WORD" is a simple single screen word processor, it allows you fill the screen

with text, save it and print it (Press BREAK to get access to the EXIT, SAVE, LOAD and PRINT options.). There are no fancy insert or delete functions, but there is full screen cursor control and type over.

There are a few interesting things to note about this program. The first is the use of ONBRK at various lines to change the function of the break key. Second, is the error trap routine near the end of the program. Third, the Save/Load routine was designed to work with disk and can be made to work with cassette, by changing the SAVEM in line 590 to a CSAVEM and the LOADM in line 600 to CLOADM. Finally, notice the color of the cursor (HINT: It is not the normal underline).

```
10 CLEAR5000,&H5FFF
15 DIM A(7)
20 DS=1 '1= use double speed
30 IF DS THEN POKE&HFFD9,0
40 SS=&H6C000 'START OF SCREEN
45 GOSUB800
50 ONERR GOTO630:ONBRK GOTO300
60 WD=40:IF NOT(WD=40 OR WD=80) THEN WD=40
70 WIDTH WD
80 DIM WN$(WD),WN(WD):X=0:Y=0
90 LOCATE X,Y
100 A$=INKEY$:IF A$=""THEN100
110 IF A$=CHR$(3) THEN 300
120 IF A$=CHR$(8) THEN GOSUB200:GOTO100
130 IF A$=CHR$(9) THEN GOSUB220:GOTO100
140 IF A$=CHR$(94) THEN GOSUB240:GOTO100
150 IF A$=CHR$(10) THEN GOSUB260:GOTO100
160 IF A$=CHR$(12) THEN GOSUB 280:GOTO100
170 IF A$=CHR$(13) THEN X=0:GOSUB260:GOTO100
180 LPOKE SS+(Y*(WD*2))+X*2,ASC(A$):GOSUB220
190 GOTO100
200 X=X-1:IF X<0 THEN X=WD-1
210 LOCATE X,Y:RETURN
220 X=X+1:IF X>WD-1 THEN X=0:GOTO260
230 LOCATE X,Y:RETURN
240 Y=Y-1:IF Y<0 THEN Y=23
250 LOCATE X,Y:RETURN
260 Y=Y+1:IF Y>=24 THEN Y=0
270 LOCATE X,Y:RETURN
280 CLS:X=0:Y=0:LOCATEX,Y:RETURN
290 'Break was pressed, print options and wait for response
300 CA=PEEK(&H11A):POKE&H11A,255:GOSUB390:GOSUB460:ONBRK
    GOTO300:IF A$="C"THEN POKE&H11A,CA:GOTO90 ELSE IF A$=CHR$(3)
    THEN 450
310 IF A$="I" THEN 480 ELSE IF A$<>"P" THEN300
315 POKE&H11A,CA
320 IF DS THEN POKE&HFFD8,0
330 FORMY=0TO23:FORMX=0TOWD-1
340 LOCATEMX,MY:HSTAT A$,A,XP,YP:PRINT#-2,A$;
350 NEXTMX:PRINT#-2,CHR$(13);:NEXTMY
```

```
355 PRINT#-2,CHR$(12);
360 IF DS THEN POKE&HFFD9,0
370 ONBRK GOTO300:GOTO300
380 'Print options and wait for key
390 ONBRK GOTO400
400 TY=0:FORTX=0TOWD-1:LOCATETX,TY
    :HSTAT WN$,WN(TX),M0,M1:WN$(TX)=WN$ :NEXTTX
410 LOCATE0,0:ATTR0,4,B:PRINTSTRING$(WD-1," ");
    :LOCATE (WD-39)/2,0
    :PRINT"BREAK=EXIT  P=PRINT  I=I/O  C=CONTINUE";
420 ONBRK GOTO450
430 A$=INKEY$:IF A$=""THEN430 ELSE RETURN
440 'Restore text under message window and end
450 GOSUB460:LOCATE0,22:GOSUB810:POKE&HFFD8,0:END
460 LOCATE0,0:ATTR0,0:FORTX=0TO(WD*2)-1STEP2
    :LPOKE SS+TX,ASC(WN$(INT(TX/2))):LPOKE SS+TX+1,0
    :NEXT:LOCATE0,0:RETURN
470 'Save or Load text
480 ONBRK GOTO610:FX=LPEEK(SS+(WD*2)):GOSUB700
    :LOCATE (WD-17)/2,0:PRINT"SAVE OR LOAD TEXT";
490 A$=INKEY$
500 IF NOT(A$="S" OR A$="L") THEN490
510 GOSUB700:LOCATE 4,0:PRINT"FILENAME TO ";:IF A$="S" THEN
    PRINT "SAVE>";:ELSE PRINT"LOAD>";
520 LINEINPUT FL$:LOCATE0,0:LPOKE SS+(WD*2),FX
    :LPOKE SS+(WD*2)+1,0:FL$=LEFT$(FL$+"        ",8)
530 GOSUB460:IF FL$=STRING$(8," ")THEN580
540 IF DS THEN POKE&HFFD8,0
550 IF A$="S" THEN GOSUB590
560 IF A$="L" THEN GOSUB600
570 IF DS THEN POKE&HFFD9,0
580 POKE&H11A,CA:GOTO300
590 POKE&HFFA3,&H36:SV=WD*2*24:SAVEM FL$,&H6000,&H6000+SV,&H6000
    :RETURN
600 POKE&HFFA3,&H36:LOADM FL$:RETURN
610 LOCATE0,0:LPOKE SS+(WD*2),FX:LPOKE SS+(WD*2)+1,0:GOSUB460
    :GOTO300
620 'Process errors here
630 ER=ERNO:LN=ERLIN:OPEN"O",0,""
640 IF DS THEN POKE&HFFD9,0
650 IF ER=26 THEN EM$="FILE NOT FOUND, PRESS ANY KEY"
    :EL=LEN(EM$)  ELSE EM$="ERROR ENCOUNTERED, PRESS ANY KEY"
    :EL=LEN(EM$)
660 GOSUB700:LOCATE (WD-EL)/2,0:PRINTEM$;
670 M$=INKEY$:IF M$=""THEN670
680 GOSUB460:GOTO300
690 'Print white message window
700 LOCATE0,0:ATTR0,4:PRINTSTRING$(WD-1," ");:RETURN
800 FORX=1TO7:READA(X):POKEA(X),4:NEXTX:RETURN
810 FORX=1TO7:POKEA(X),&H40:NEXTX:RETURN
900 DATA &HF797,&HF7A3,&HF7EC,&HF80F,&HF84F,&HF91B,&HF89C
```

Some new enhancments are also available for the 32X16 screen, these include true lower case, border color change and an invert

screen color mode. Basic was not re-written to allow these features to work, but since it now always resides in RAM, a simple POKE may be used to correct this problem.

```
POKE &H95C9,&H7F
```

This will prevent the console out vector from reseting the values at $FF22. To enable true lower case, POKE &HFF22,&H10, To enable the inverted screen mode, POKE &HFF22,&H20 and to enable the border color invert, POKE &HFF22,&H40. To get a combination of these features, add the values of the features desired together and POKE address $FF22 with the result.

CHAPTER 4
SMOOTH SCROLLING, PEEKS AND POKES, AND OTHER TIDBITS

The most interesting new feature of the Color Computer III is its ability to smooth scroll in both the vertical and horizontal directions. Scrolling is not supported by Basic except through the use of the POKE command.

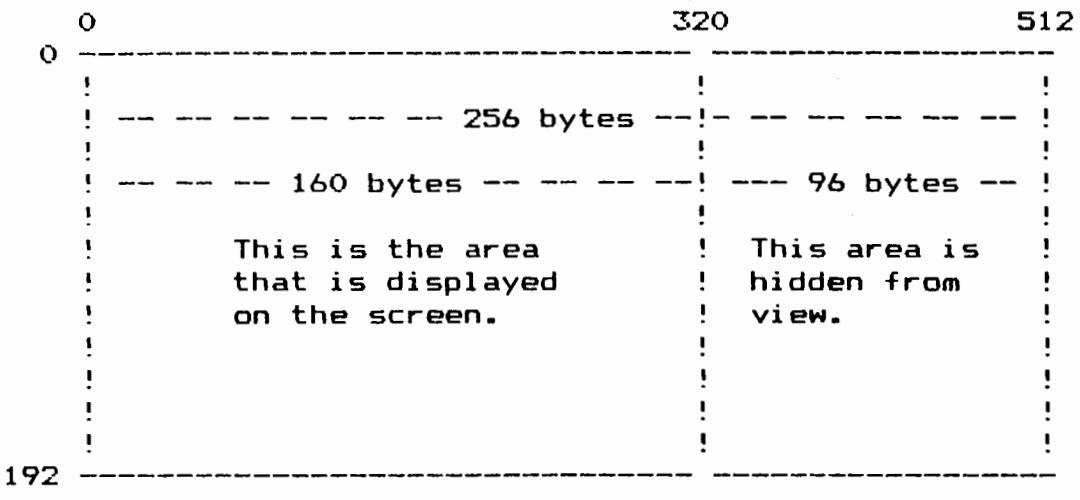
Vertical scrolling is controlled by three registers of the GIME chip, $FF9D, $FF9D AND $FF9E. These registers work together to display addresses within the 512K system, in register $FF9C only bits 5-7 are used. Each time these registers are incremented, the display moves by 8 bytes, in order to scroll an entire row, the registers need to be incremented by a value which is equal to the NUMBER OF BYTES PER HORIZONTAL ROW divided by 8. The following example will start at Basic's graphics page ($60000) and scroll the screen according to the position of the joystick. The particular screen being viewed has 160 bytes per horizontal row.

```
10 ONBRK GOTO 190
20 HSCREEN 2:HCLS
30 HCIRCLE(160,96),40,4
40 HPAINT(160,96),5,4
50 ST=49152
60 J0=JOYSTK(0):J1=JOYSTK(1):J1=J1-32
70 IF INKEY$="Q"THENST=49152:GOSUB150:END
80 S=SGN(J1):J1=ABS(J1)
90 IF J1<15 THEN S=0
100 IF J1>23 THEN S=S*2
120 IF J1>30 THEN S=S*3
130 ST=ST-(S*(160/8)):GOSUB 150
140 GOTO60
150 A=INT(ST/65536):A0=A*32
160 A1=INT(ST/256):A2=ST-(A1*256):A1=A1 AND 255
170 POKE&HFF9C,A0:POKE&HFF9D,A1:POKE&HFF9E,A2
180 RETURN
190 ST=49152:GOSUB150
```

Line 130 is where the registers get incremented, "S" will equal -1, 0 or 1 depending upon the position of the joystick. This will be multiplied by the number of bytes per horizontal row (160) divided by 8. This value is then converted by the subroutine starting at line 150, into the 3 bytes necessary for storage into registers $FF9C, $FF9D and $FF9E. To make the scroll faster, add the DOUBLE SPEED poke to line 10 (POKE&HFFD9,0).

The horizontal scroll register is located at address $FF9F. Only 7 bits (0-6) of this register are used to control the scroll, bit 7 is used to activate the HORIZONTAL VIRTUAL ENABLE (HVEN) mode. Horizontal Virtual Enable uses 48K of memory, is not accessable through Basic except with pokes and is required anytime horizontal scrolling needs to do a complete wrap around. HVEN works by forcing the bytes per horizontal row to 256, the graphics mode selected has no effect on this except to define how much of the 256 horizontal bytes to display. In other words, if a 320X192 (160 bytes across)

graphics mode is selected while HVEN is turned on, the screen will show the normal 160X192 bytes and an area of 96X192 bytes will be hidden off of the edge of the screen. The following diagram will help clarify this.



The following short program will set up a horizontal virtual enable screen, clear it with a small machine language routine (Basic will only clear a 32000 byte screen), LPOKE a colored block on the screen and allow it to scroll according to the position of the joystick.

```
10 CLEAR200,&H5FFF-256
20 ON BRK GOTO180
30 HO=0
40 HSCREEN 2:GOSUB 160
50 FOR X=&H5F00 TO &H5F10:READ A:POKEX,A:NEXTX
60 FORX=&H30 TO &H35:POKE&HFFA3,X:EXEC &H5F00:NEXTX
80 AD=416838
90 FORY=0 TO 7:FOR X=0 TO 19
100 LPOKE AD+X,1:NEXTX
110 AD=AD+256:NEXTY
120 JO=JOYSTK(0):JO=JO-32:S=SGN(JO):JO=ABS(JO)
130 IF JO<15 THEN S=0
140 HO=(HO-S)AND 255
150 GOSUB 160:GOTO120
160 POKE&HFF9F,(HO OR &H80)
170 RETURN
180 HO=0:GOSUB160
190 ' The following machine language code is contained
200 ' in the DATA statements that follow:
210 ' PSHS X,D,U,Y  SAVE REGISTERS
220 ' LDY #$2000    CLEAR THIS MANY BYTES (8K)
230 ' LDX #$6000    START CLEARING AT THIS ADDRESS
240 'LOOP
250 ' CLR ,X+       CLEAR BYTE AT X, INCREMENT X
260 ' LEAY -1,Y     COUNT DOWN HOW MANY TO CLEAR
270 ' BNE LOOP      KEEP GOING IF COUNT NOT=0
```

```
280 ' PULS X,D,U,Y,PC  RETURN TO BASIC
290 DATA &H34,&H76,&H10,&H8E,&H20,&H00,&H8E,&H60,&H00
300 DATA &H6F,&H80,&H31,&H3F,&H26,&HFA,&H35,&HF6
```

Line 50 pokes in a small machine language routine that will zero the 8K block of memory locate at $6000 of the CPU's workspace. Line 60 then swaps each 8K block of memory required for the graphics screen into the slot at $6000 and executes the routine to clear it. Notice that at line 160, the Horizontal Offset (HO) is OR'd with $80, this will insure that HVEN will remain set. If for some reason it was desirable to not be in the HVEN mode, HO would need to be ANDed with $7F to insure that the HVEN bit was forced off.

Along with the blessing of more memory comes the greater possibility that part of it may be bad, it's simply the law of averages and somewhere down the line the law will catch someone. The following routine is a simple 128/512K memory test program, written partially in Basic with a small machine language routine that will check the 8K block of memory located at $6000 of the CPU's workspace. The Basic program will be used to print messages, sequentially swap 8K blocks of memory into the slot at $6000 and execute the machine language routine to check the block.

```
10 CLEAR200,&H5FFF-256
15 WIDTH32
20 PB=PEEK(&HFFA2)AND &H3F
25 DIM BB(&H3F)
27 FOR X=0 TO 48:READ A:POKE&H5F00+X,A:NEXTX
30 CLS:PRINT@32*5,"MEMORY SIZE (128 OR 512) >";:INPUTMS
40 IF NOT(MS=128 OR MS=512) THEN 30
50 IF MS=128 THEN SB=&H30 ELSE SB=0
60 FOR X=SB TO &H3F:IF X=PB THEN 90
70 POKE&HFFA3,X:EXEC &H5F00:IF PEEK(&H5F02)<>0 THEN BB(X)=1
80 IF (X AND 1) THEN A$="WORKING" ELSE A$="       "
90 PRINT@32*7+12,A$:NEXTX
100 F1=0:F2=0
110 FOR X=SB TO &H3F:IF BB(X)<>0 AND F1=0 THEN F1=1
    :PRINT@32*9,"BYTE(S) BAD IN BLOCK(S):"
111 IF F1=1 THEN PRINTX",";
120 NEXT X:PRINT CHR$(8):IF F1=0 THEN
    PRINT@32*9,"     ALL MEMORY CHECKS GOOD"
130 PRINT"      MEMORY TEST COMPLETE"
140 END
150 'The following machine code is contained in the DATA
160 'statements that follow:
170 ' START
180 '  BRA START1        GOTO PROGRAM START
190 ' ERBYTE
200 '  FCB 0             STORE ERROR CODE HERE
210 ' START1
220 '  PSHS D,X,U,Y      SAVE ALL REGISTERS
230 '  LEAU ERBYTE,PCR   POINT U TO ERROR STORAGE BYTE
240 '  CLR ,U            START WITH NO ERROR
250 '  LDY #$2000        CHECK THIS MANY BYTES
```

```
260 '  LDX #$6000        START CHECKING FROM HERE
270 ' LOOP
280 '  LDA ,X            SAVE ORIGINAL BYTE
290 '  LDB #$55
300 '  STB ,X            STORE A 0101 BIT PATTERN
310 '  LDB ,X            GET IT BACK
320 '  CMPB #$55         SEE IF THE SAME AS STORED
330 '  BNE BAD           BRANCH IF NOT THE SAME
340 '  COMB
350 '  STB ,X            NOW STORE 1010 PATTERN
360 '  LDB ,X            GET IT BACK
370 '  CMPB #$AA         SEE IF THE SAME AS STORED
380 '  BEQ NOTBAD        BRANCH IF IT IS THE SAME
390 ' BAD
400 '  STB ,U            SET ERROR BYTE
410 ' NOTBAD
420 '  STA ,X+           PUT BACK ORIGINAL, MOVE TO NEXT BYTE
430 '  LEAY -1,Y         DECREMENT COUNTER
440 '  BNE LOOP          BRANCH IF NOT REACHED ZERO YET
450 '  PULS D,X,U,Y,PC   RETURN TO BASIC
460 '
500 DATA &H20,&H01,&H00,&H34,&H76,&H33,&H8D,&HFF
510 DATA &HF9,&H6F,&HC4,&H10,&H8E,&H20,&H00,&H8E
520 DATA &H60,&H00,&HA6,&H84,&HC6,&H55,&HE7,&H84
530 DATA &HE6,&H84,&HC1,&H55,&H26,&H09,&H53,&HE7
540 DATA &H84,&HE6,&H84,&HC1,&HAA,&H27,&H02,&HE7
550 DATA &HC4,&HA7,&H80,&H31,&H3F,&H26,&HE3,&H35
560 DATA &HF6
```

The new high resolution screens are fantastic, very detailed pictures, graphs and charts can be drawn and painted with a variety of different colors. The 320X192 screen uses 32K bytes of memory, fortunately this memory is not taken from the Basic program area. What this means is that your program size doesn't have to suffer anymore when using the new high resolution screens, it also means however, that you can't directly save the screen to tape or disk. A Basic program must now be used to save the screen a block at a time. The number of blocks to save is determined by the size of the screen, remember, each block is 8K, so a 32K screen would use 4 blocks. Basic always puts it's graphic screen starting at block $30, so to save a 32K screen blocks $30, $31, $32 and $33 would all need to be saved. The following routine will illustrate how this is done.

```
5 WIDTH 40
10 CLEAR200,&H5FFF  'Reserves 8K of memory from $6000 to $7FFF
15 ONERR GOTO 200
16 POKE&HE6C6,18:POKE&HE6C7,18 'disable clear screen
20 CLS:PRINT"(S)AVE OR (L)OAD A SCREEN?"
30 A$=INKEY$:IF NOT(A$="S" OR A$="L") THEN 30
40 IF A$="S" THEN AC$="SAVE":A=0 ELSE AC$="LOAD":A=1
50 PRINT"ENTER FILENAME TO "AC$:LINEINPUT FL$
60 IF FL$="" THEN END
```

```
65 C=INSTR(FL$,"."):IF C=0 THEN C=INSTR(FL$,"/")
67 IF C<>0 THEN FL$=LEFT$(FL$,C-1)
69 HSCREEN2
70 FOR X=&H30 TO &H33:POKE&HFFA3,X
80 IF A=0 THEN SAVEM FL$+"/"+STR$(X),&H6000,&H7FFF,&H6000
90 IF A=1 THEN LOADM FL$+"/"+STR$(X)
100 NEXT X:PRINT AC$;" SUCCESSFUL":END
200 OPEN"O",0,"":PRINT:PRINT "ERROR ENCOUNTERED DURING ";AC$:END
```

Notice the OPEN statement in line 200, it opens a file to the screen. This may seem like a strange thing to do, but it is necessary in this case because the routine that handles the ONBRK control does not reset the device number to the screen. Most of the time this will not effect anything, but here the error could occured while accessing the disk which would cause the message in line 200 to be printed to the disk buffer instead of to the screen. Other commands that will reset the device number are CLS and POKE&H6F,0.

PEEK and POKE are a couple of commands that allow direct access to memory within the CPU's 64K workspace. Some very powerful things can be accomplished if they are used properly, to include modifying Basic. Listed below are a few interesting and usefull changes that can be made.

To prevent HSCREEN command from clearing the screen:

```
POKE&HE6C6,18:POKE&HE6C7,18
```

To change the rate of blink rate of characters with the blink attribute set:

```
POKE&HFF94,(MSB OF BLINK RATE)
POKE&HFF95,(LSB OF BLINK RATE)
```

To change the color values for the CMP command, poke a value from 0 to 63 into the memory between $E654 - $E663. ($E654=PALETTE 0, $E655=PALETTE 1, ETC.)

To change the color values for the RGB command, poke a value from 0 to 63 into the memory between $E664 - $E673. ($E664=PALETTE 0, $E665=PALETTE 1, ETC.)

To fix a bug and make the CMP and RGB commands change all 16 palette registers:

```
POKE&HE649,16
```

To change the depth of the HSCREEN graphics modes from 192 to 200 and to allow the graphics commands to reach down that far:

```
POKE&HE06C,&H35
POKE&HE06D,&H3E
POKE&HE06E,&H34
POKE&HE06F,&H3D
POKE&HEB75,199
POKE&HE7BA,200
POKE&HE7BE,199
POKE&HEF8F,18
```

To change the cursor on the Width40 and Width80 screens:

```
POKE&HF797,X
POKE&HF7A3,X
POKE&HF7EC,X
POKE&HF80F,X
POKE&HF84F,X
POKE&HF91B,X
POKE&HF89C,X
```

(Where X equals the attribute value to use. See character attributes in chapter 5 for more information.)

To find out the current screen width:

```
PRINT PEEK(&HE7)
```

(0=32 Characters, 1=40 Characters, 2=80 characters)

To find out the current HSCREEN mode:

```
PRINT PEEK(&HE6)
```

(0=TEXT, 1=HSCREEN 1, 2=HSCREEN 2, 3=HSCREEN 3, 4=HSCREEN 4)

To find the current default foreground color for HSCREEN graphics modes:

```
PRINT PEEK(&HFE0A)
```

To find the current default background color for HSCREEN graphics modes:

```
PRINT PEEK(&HFE0B)
```

To find the current ON BRK line number:

```
PRINT PEEK(&HFE0C)*256+PEEK(&HFE0D)
```

To find the current ON ERR line number:

```
PRINT PEEK(&HFE0E)*256+PEEK(&HFE0F)
```

High resolution character set for HPRINT is located between:

```
&HF09D - &HF39C
```

(8 bytes are required to define one character)

CHAPTER 5
COCO III MEMORY MAP

The Color Computer III has two modes, the COCO mode which acts just like a Color Computer or Color Computer II, and an ADVANCED VIDEO PROCESSOR (AVP) mode which uses memory management, new high resolution screens and the other new features of the Color Computer III. In brief, the memory map looks something like this.

```
Total range:  0000 - $7FFFF     (512 Kilobytes)

I/O and Control: XFF00 - XFFFF     (All banks)

ROM:      $78000 - $7FEFF   (Deselectable)
     or
          $78000 - $7FDFF   (Deselectable)

RAM:
     64K Coco mode:    X0000 - XFEFF  (Except for ROM)

    128K Coco mode:    X0000 - XFEFF  (Except for ROM)
                       4 additional 16K pages at X4000 - X7FFF
    128K  AVP mode:    $60000 - $7FEFF (Except ROM, I/O & CTRL)
                       Duplicated at... $40000 - $5FFFF
                                        $20000 - $3FFFF
                                        $00000 - $1FFFF
    256K Coco mode:    (Same as 128K Coco mode)
    256K  AVP mode:    $40000 - $7FEFF (Except ROM, I/O & CTRL)
                       Duplicated at... $00000 - $3FFFF
    512K Coco mode:    (Same as 128K Coco mode)
    512K  AVP mode:    $00000 - $7FEFF (Except ROM, I/O & CTRL)

I/O:       XFF00 - XFFFF
     XFF00 - XFF03     PIA0  (Same as old Coco)
     XFF10 - XFF1F     RESERVED
     XFF20 - XFF23     PIA1  (Same as old Coco)
     XFF30 - XFF3F     RESERVED
     XFF40 - XFF5F     SCS
     XFF60 - XFF7F     UNDECODED (Current peripherals)
     XFF90 - XFF9F     GIME CHIP CONTROL
     XFFA0 - XFFAF     MMU
     XFFB0 - XFFBF     COLOR PALETTE
     XFFC0 - XFFDF     SAM CONTROL REGISTERS
     XFFE0 - XFFFF     INTERRUPT VECTORS
```

It is possible for a device to respond to more than one address, but only those listed above should be used.

Following is a detailed breakout of the I/O section.

FF00 - FF03 PIA0

```
        BIT 0= KEYBOARD ROW 1 and right joystick button one
        BIT 1= KEYBOARD ROW 2 and left joystick button one
        BIT 2= KEYBOARD ROW 3 and right joystick button two
 FF00   BIT 3= KEYBOARD ROW 4 and left joystick button two
        BIT 4= KEYBOARD ROW 5
        BIT 5= KEYBOARD ROW 6
        BIT 6= KEYBOARD ROW 7
        BIT 7= JOYSTICK COMPARISON INPUT

        BIT 0= (0=IRQ to CPU disabled; 1=IRQ to CPU enabled)
        BIT 1= (0=IRQ occurs on falling edge of Horiz sync)
               (1=IRQ occors on rising edge of Horiz sync)
        BIT 2= Normally 1 (0 changes data direction reg to $FF00)
 FF01   BIT 3= LSB of the two analog MUX select lines
        BIT 4= ALWAYS 1
        BIT 5= ALWAYS 1
        BIT 6= NOT USED
        BIT 7= HORIZONTAL SYNC INTERRUPT FLAG

        BIT 0= KEYBOARD COLUMN 1
        BIT 1= KEYBOARD COLUMN 2
        BIT 2= KEYBOARD COLUMN 3
 FF02   BIT 3= KEYBOARD COLUMN 4
        BIT 4= KEYBOARD COLUMN 5
        BIT 5= KEYBOARD COLUMN 6
        BIT 6= KEYBOARD COLUMN 7
        BIT 7= KEYBOARD COLUMN 8

        BIT 0= (0=IRQ to CPU disabled; 1=IRQ to CPU enabled)
        BIT 1= (0=IRQ occurs on falling edge of Field sync)
               (1=IRQ occors on rising edge of Field sync)
        BIT 2= Normally 1 (0 changes data direction reg to $FF02)
 FF03   BIT 3= MSB of the two analog MUX select lines
        BIT 4= ALWAYS 1
        BIT 5= ALWAYS 1
        BIT 6= NOT USED
        BIT 7= FIELD SYNC INTERRUPT FLAG
```

FF20 - FF23   PIA1

```
        BIT 0= CASETTE DATA INPUT
        BIT 1= RS-232 DATA OUTPUT
        BIT 2= 6 BIT D/A LSB
 FF20   BIT 3= 6 BIT D/A
        BIT 4= 6 BIT D/A
        BIT 5= 6 BIT D/A
        BIT 6= 6 BIT D/A
        BIT 7= 6 BIT D/A MSB
```

```
        BIT 0= (0=FIRQ to CPU disabled; 1=FIRQ to CPU enabled)
        BIT 1= (0=Set flag on falling edge of CD)
               (1=Set flag on rising edge of CD)
        BIT 2= NORMALLY 1; 0 Changes Data direction reg to $FF20
 FF21   BIT 3= CASETTE MOTOR CONTROL: 0=OFF  1=ON
        BIT 4= ALWAYS 1
        BIT 5= ALWAYS 1
        BIT 6= NOT USED
        BIT 7= CD Interrupt flag

        BIT 0= RS-232 DATA INPUT
        BIT 1= SINGLE BIT SOUND OUTPUT
        BIT 2= NOT USED
 FF22   BIT 3= VDG CTRL OUTPUT          CSS
        BIT 4= VDG CTRL OUTPUT          GM0 & UPPER/LOWER CASE NOT
        BIT 5= VDG CTRL OUTPUT          GM1 & INVERT
        BIT 6= VDG CTRL OUTPUT          GM2
        BIT 7= VDG CTRL OUTPUT          A NOT/G

        BIT 0= (0=FIRQ to CPU disabled; 1=FIRQ to CPU enabled)
        BIT 1= (0=Set flag on falling edge of CART)
               (1=Set flag on rising edge of CART)
        BIT 2= NORMALLY 1; 0 Changes Data direction reg to $FF20
 FF23   BIT 3= SIX BIT SOUND ENABLE
        BIT 4= ALWAYS 1
        BIT 5= ALWAYS 1
        BIT 6= NOT USED
        BIT 7= CARTRIDGE Interrupt flag

FF27    USED FOR POWER UP SYSTEM CONFIGURATION, BIT DEFS
        ARE NOT AVAILABLE AT THIS TIME

FFD8    TURN OFF DOUBLE SPEED
FFD9    SET TO DOUBLE SPEED

FFDE    SET TO ROM MODE
FFDF    SET TO ALL RAM MODE

GIME CHIP CONTROL REGISTERS:  FF90 - FF9F

        BIT 7 - 1=Color Computer compatible mode
        BIT 6 - 1=MMU enabled
        BIT 5 - 1=Chip IRQ output enabled
 FF90   BIT 4 - 1=Chip FIRQ output enabled
        BIT 3 - 1=DRAM at XFEXX is constant
        BIT 2 - 1=Standard SCS
        BIT 1 - ROM map control (see table below)
        BIT 0 - ROM map control (see table below)

          BIT 1     BIT 0         ROM MAPPING
        ----------------------------------------
            0         X         16K INTERNAL, 16K EXTERNAL
            1         0         32K INTERNAL
            1         1         32K EXTERNAL (Except vectors)
```

```
       BIT 7 - 0=Two banks of DRAM
       BIT 6 - 0=64K chips, 1=256K chips
       BIT 5 - Timer input select: 0=70us, 1=63us
FF91   BIT 4 - NOT USED
       BIT 3 - NOT USED
       BIT 2 - NOT USED
       BIT 1 - NOT USED
       BIT 0 - MMU Task Register Select (TR)

       BIT 7 - NOT USED
       BIT 6 - NOT USED
       BIT 5 - Interrupt from Timer enabled
FF92   BIT 4 - Horizontal border IRQ enabled
       BIT 3 - Vertical border IRQ enabled
       BIT 2 - Serial data IRQ enabled
       BIT 1 - Keyboard IRQ enabled
       BIT 0 - Cartridge IRQ enabled

       BIT 7 - NOT USED
       BIT 6 - NOT USED
       BIT 5 - Interrupt from Timer enabled
FF93   BIT 4 - Horizontal border FIRQ enabled
       BIT 3 - Vertical border FIRQ enabled
       BIT 2 - Serial data FIRQ enabled
       BIT 1 - Keyboard FIRQ enabled
       BIT 0 - Cartridge FIRQ enabled

FF94 - TIMER MOST SIGNIFICANT BYTE
FF95 - TIMER LEAST SIGNIFICANT BYTE
```

The above timer is a 16 bit interval timer, the count automatically begins when a value is stored in the MSB. The input clock is either 14 MHz or horizontal sync as selected by BIT 5 of $FF91. As the count falls through zero, an interrupt is generated (if enabled), and the count is automatically reloaded.

```
FF96 - Reserved for future use
FF97 - Reserved for future use

       BIT 7 - 0=alphanumeric, 1=bit plane graphics
       BIT 6 - 1=individual attributes enabled in alpha
       BIT 5 - 1=color set flip for old articfacting screens
FF98   BIT 4 - 1=Monochrome signal output (on composite)
       BIT 3 - 1=50 Hz vertical sync
       BIT 2 - lines per row  (see table below)
       BIT 1 - lines per row  (see table below)
       BIT 0 - lines per row  (see table below)
```

BIT2	BIT1	BIT0	lines per character row
0	0	0	one
0	0	1	two
0	1	0	three
0	1	1	eight
1	0	0	nine
1	0	1	ten
1	1	0	twelve
1	1	1	(reserved)

```
      BIT 7 - NOT USED
      BIT 6 - lines per field (see table below)
      BIT 5 - lines per field (see table below)
FF99  BIT 4 - Horizontal resolution (HRES2) ( see video
      BIT 3 - Horizontal resolution (HRES1)     resolution
      BIT 2 - Horizontal resolution (HRES0)     page )
      BIT 1 - Color resolution (CRES1)      (see video
      BIT 0 - Color resolution (CRES0)       resolution page)
```

BIT6	BIT5	Lines per field
0	0	192
0	1	200
1	0	210
1	1	225

```
      BIT 7 - NOT USED
      BIT 6 - NOT USED
      BIT 5 - MSB of RED border color
FF9A  BIT 4 - MSB of GREEN border color
      BIT 3 - MSB of BLUE border color
      BIT 2 - LSB of RED border color
      BIT 1 - LSB of GREEN border color
      BIT 0 - LSB of BLUE border color

      BIT 7 - Make characters blink
      BIT 6 - Underline characters
      BIT 5 - Foreground color bit (palette address)
FF9B  BIT 4 - Foreground color bit (palette address)
      BIT 3 - Foreground color bit (palette address)
      BIT 2 - Background color bit (palette address)
      BIT 1 - Background color bit (palette address)
      BIT 0 - Background color bit (palette address)

      BIT 7 - NOT USED
      BIT 6 - Vertical offset address Y18
      BIT 5 - Vertical offset address Y17
FF9C  BIT 4 - Vertical offset address Y16
      BIT 3 - Vertical scroll bit (alpha mode)
      BIT 2 - Vertical scroll bit (alpha mode)
      BIT 1 - Vertical scroll bit (alpha mode)
      BIT 0 - Vertical scroll bit (alpha mode)
```

```
        BIT 7 - Vertical offset address Y15
        BIT 6 - Vertical offset address Y14
        BIT 5 - Vertical offset address Y13
FF9D    BIT 4 - Vertical offset address Y12
        BIT 3 - Vertical offset address Y11
        BIT 2 - Vertical offset address Y10
        BIT 1 - Vertical offset address Y9
        BIT 0 - Vertical offset address Y8

        BIT 7 - Vertical offset address Y7
        BIT 6 - Vertical offset address Y6
        BIT 5 - Vertical offset address Y5
FF9E    BIT 4 - Vertical offset address Y4
        BIT 3 - Vertical offset address Y3
        BIT 2 - Vertical offset address Y2
        BIT 1 - Vertical offset address Y1
        BIT 0 - Vertical offset address Y0

        BIT 7 - Horizontal virtual enable (HVEN)
        BIT 6 - Horizontal offset address
        BIT 5 - Horizontal offset address
FF9F    BIT 4 - Horizontal offset address
        BIT 3 - Horizontal offset address
        BIT 2 - Horizontal offset address
        BIT 1 - Horizontal offset address
        BIT 0 - Horizontal offset address
```

NOTE: HVEN enables a horizontal screen width of 256 bytes regardless of the resolution or color mode bits selected. This will allow a "virtual" screen somewhat larger than the displayed screen. The user can move the "window" (the displayed screen) by means of the horizontal offset address bits. In character mode, the screen width is 128 characters regardless of attribute (or 64, if double wide is selected).

VIDEO RESOLUTION

The combination of HRES and CRES bits determine the resolution of the screen. Listed below are the resolutions which are supported. Any combinations not listed below may not be supported in future versions.

ALPHANUMERICS: (Bit 7 of FF98=0, Bit 7 of FF90=0)

HRES2	HRES1	HRES0	CRES1	CRES0	MODE
0	--	0	--	--	32 character
0	--	1	--	--	40 character
1	--	0	--	--	64 character
1	--	1	--	--	80 character

NOTE: If character by character attributes are desired in the 64 or 80 character modes, two banks of DRAM are required (Bit 7 of FF91=0).

GRAPHICS: (Bit 7 of FF98=1, Bit 8 of FF90=0)

HRES2	HRES1	HRES0	CRES1	CRES0	BANKS REQD	PIXELS	COLORS	BYTES ACROSS
1	1	1	0	1	2	640	4	160
1	0	1	0	0	1	640	2	80
1	1	0	0	1	2	512	4	128
1	0	0	0	0	1	512	2	64
1	1	1	1	0	2	320	16	160
1	0	1	0	1	1	320	4	80
0	1	1	0	0	1	320	2	40
1	1	0	1	0	2	256	16	128
1	0	0	0	1	1	256	4	64
0	1	0	0	0	1	256	2	32
1	0	1	1	0	1	160	16	80
0	1	1	0	1	1	160	4	40
0	0	1	0	0	1	160	2	20
1	0	0	1	0	1	128	16	64
0	1	0	0	1	1	128	4	32
0	0	0	0	0	1	128	2	16

In addition to the above modes, the previous Coco modes are available. These result when Bit 7 of FF90 is set, the HRES and CRES bits have no effect on these modes. The number of required banks of ram listed above is a minimum requirement. Please note that in the 2 color modes there are 8 pixels per byte, in the 4 color modes there are 4 pixels per byte and in the 16 color modes there are 8 pixels per byte.

CHARACTER ATTRIBUTE MODES

Page attribute mode: (Bit 6 of FF98=0)

In this mode, the attributes are selected by the bits of $FF9B, and are enabled by the Most Significant Bit of the character code.

Character bit definitions

BIT 7 = Attribute enable
BIT 6 = Character bit 6
BIT 5 = Character bit 5
BIT 4 = Character bit 4
BIT 3 = Character bit 3
BIT 2 = Character bit 2
BIT 1 = Character bit 1
BIT 0 = Character bit 0

Individual attribute mode: (Bit 6 of FF98=1)

In this mode, each character on the screen has it's own attribute byte, this allows for greater flexibility but requires twice as many bytes per screen.

Character bit definitions (Even byte)

BIT 7 = NOT USED
BIT 6 = Character bit 6
BIT 5 = Character bit 5
BIT 4 = Character bit 4
BIT 3 = Character bit 3
BIT 2 = Character bit 2
BIT 1 = Character bit 1
BIT 0 = Character bit 0

Attribute bit definitions (Odd byte)

BIT 7 = Blink this character
BIT 6 = Underline this character
BIT 5 = Character color bit (palette address)
BIT 4 = Character color bit (palette address)
BIT 3 = Character color bit (palette address)
BIT 2 = Background color bit (palette address)
BIT 1 = Background color bit (palette address)
BIT 0 = Background color bit (palette address)

Individual character attributes are not available if Bit 7 of FF90=1 (Coco compatible mode).

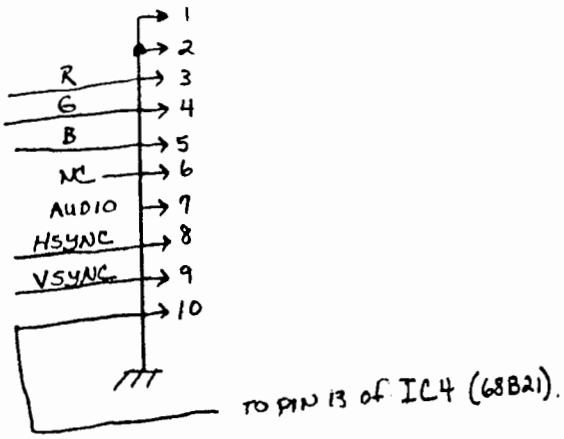
CHAPTER 6
COCO III SUMMARY

The Color computer III has turned out to be a fine machine, it contains many features that until now have only been available in the more expensive machines. 512K of memory, 640 by 200 high resolution graphics mode, 16 colors at a time on some screens, a choice of 64 different colors.... and the list goes on. I will admit that the Coco III is not the most powerful home computer available, but it is the best buy on the market today. You will not be able to find a computer anywhere that has all of the features of the color computer III and sells for $219 dollars.

Can you imagine COCO MAX running on a 512K machine, using the 320 by 192, 16 color graphics screen! How about a 512K graphics adventure! What about spreadsheets, word processors and other business programs! Level II OS9 for the Coco III is amazing, it features a windowing enviroment that will make MAC owners envy you! The possibilities for this machine are endless.

There are a couple of hidden tricks within the Basic ROM that I would like to mention at this point. First, type WIDTH 40, then type CLS 100. Thank you T. Harris and T. Earls, they are the ones who wrote the new Basic commands. (If you type CLS 100 again, you will find that the names are gone). Now for one more thing to try. Turn off the computer, press and hold down the ALT and CTRL keys while turning it back on. Pictured from left to right are M. Hawkins, T. Harris and T. Earls. (Nice photo guys!)

Don't worry, code space was not wasted, not only was there enough space left over in the ROM for that picture, but probably a couple more as well. Hmm, I wonder if...



RGB CONNECTOR PINOUT

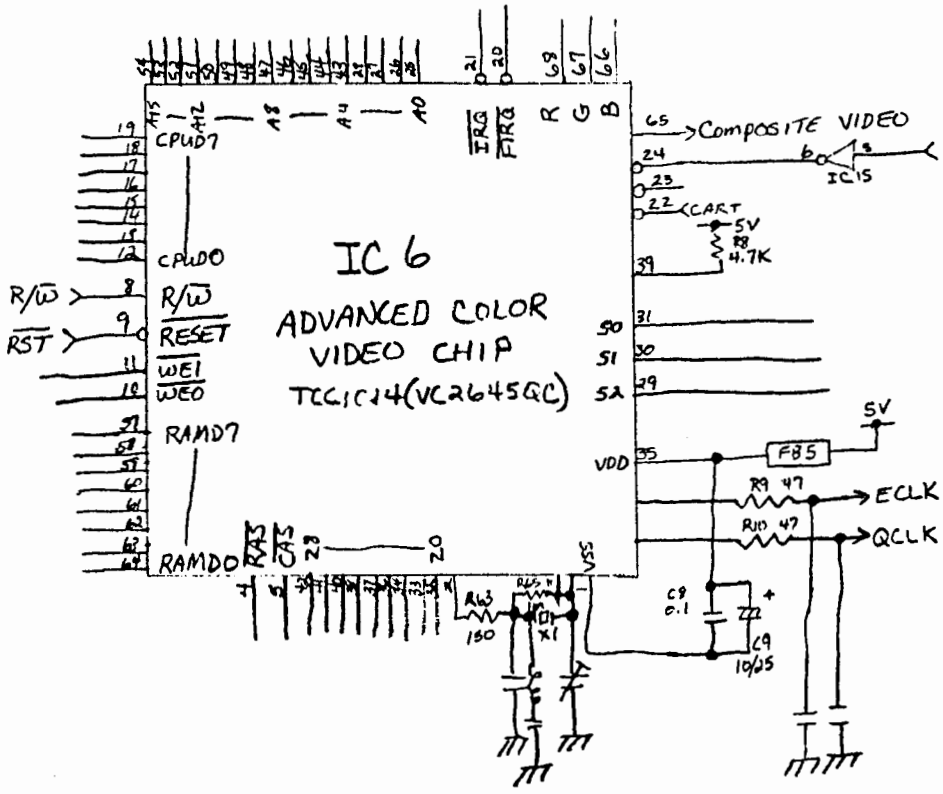


DIAGRAM OF THE GIME CHIP