Color Computer 1/2/3 Hardware Programming

Chris Lomont, Aug 2007, version 0.82

This document collects and details hardware programming information for the TRS-80 Color Computer, versions 1, 2, and 3. Although it has some tutorial information in it, it is designed to be a reference. Many areas also apply to the Color Computer Clones such as the British Dragon 32/64.

It is compiled and edited by Chris Lomont, www.lomont.org. Send comments, corrections, and errors to CoCo3 at the above domain. Please don't repost this on the web, but point to this copy, so eventually all information is corrected and integrated.

This document is compiled from many sources, listed in the Bibliography. If you feel this infringes any of your copyrighted material, email me with your material, and I will remove or rewrite from scratch the offending sections.

Hex numbers start with a $, as in 255=$FF. Addresses like $FFFE (65534) give the decimal in parentheses. 16-bit addresses like $B0F1 are in 6809 CPU address space. On the Color Computer 3, 20-bit addresses like $70FFF are in GIME address space. Also on the Color Computer 3, the Memory Mapping Unit (MMU) maps eight 8K pages from the GIME space into CPU space.

Many sections (marked TODO) need a lot more work, which I will do given time.

DISCLAIMER: All information provided as is, etc. Use at your own risk.

Version History		
0.8	June 2006	Initial version and organization of material.
0.81	July 2007	One pass of cleanup and some new material added.
0.82	Aug 2007	Rewrite, reformat, complete overhaul.

Table of Contents

Hardware Introduction

This document covers the hardware in the Color Computer, versions 1, 2, and 3, often called the CoCo 1, CoCo 2, and CoCo 3.

The original version of the Color Computer, the CoCo 1, was in a silver-gray case with a chiclet keyboard, and was available with a memory sizes of 4K (26-3001[1]), 16K (26-3002), or 32K (26-3003). Many actually had 64K of RAM, which could be accessed with special utilities. The second generation CoCo 2 came in 16K (standard and extended BASIC) and 64K RAM sizes, removed the 12V power line, and the new BASIC ROMs fixed some bugs. The CoCo 3 was a major upgrade using the ASIC Graphics Interrupt Memory Enhancement (GIME) chip, which added many new features, detailed below. Some of the new features included up to 512K of RAM, lowercase letters, 40 and 80 column text, higher clock speeds, new interrupt sources, and many new video modes.

TODO – pics?

The CoCo3 supports the CoCo 1 and 2 hardware in CoCo 1/2 compatibility mode, described in the CoCo 1/2 Compatibility Section.

All three versions of the CoCo run on a Motorola 6809 chip, details of which are in a different document. A brief note about the 6809 is below.

The main hardware interfaces are:

CoCo 1/2/3:

PIA	Peripheral Interface Adapter	General hardware Input/Output
SAM	Synchronous Address Multiplexer	Determines how data moves
VDG	Video Display Generator	Converts RAM to images

CoCo 3 only:

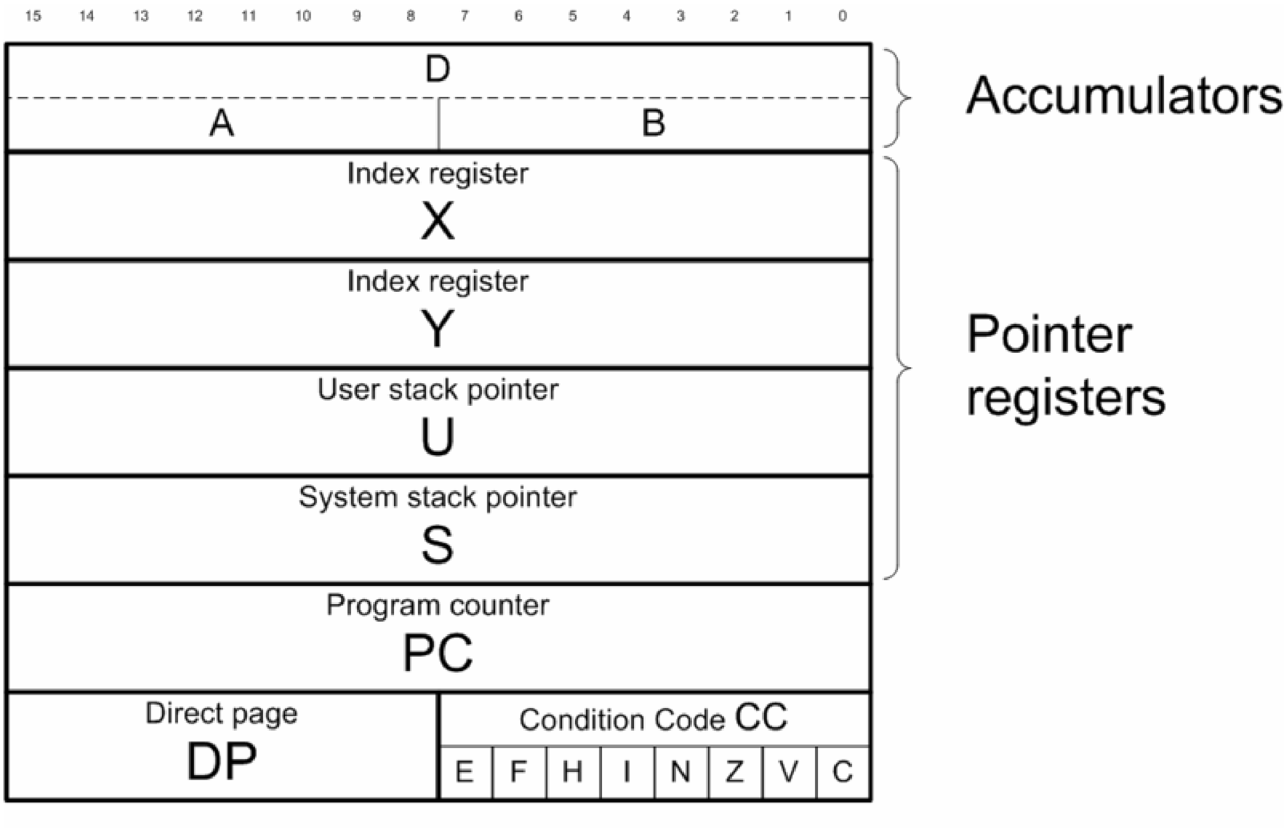
GIME	Graphics Interrupt Memory Enhancement	What it says...

Miscellaneous hardware items cover disk drives, cassette, sound, joysticks, speech packs, modem packs, multi-pak, and more.

6809 CPU Notes

The Motorola 6809 is an 8-bit CPU with some 16-bit instructions and registers. Here is a rough picture of the programming model:

[1] These type of numbers are from Radio Shack catalogs.



6809 Internal Registers

The condition code bits in the Condition Code register are used in this document, and are

Bit	Function
E	During interrupt, if 1, indicated all registers on stack, else only PC and CC. Needed for RTI (Return From Interrupt) opcode.
F	FIRQ Disabled if 1. Set to 1 on power up and during interrupt processing.
H	Half Carry from low nibble to high nibble, used for DAA (Decimal Addition Adjust) opcode.
I	IRQ Disabled if 1. Set to 1 on power up and during interrupt processing.
N	Last operation resulted in Negative.
Z	Last operation resulted in Zero.
V	Signed arithmetic overflow.
C	Carry generated.

Many hobbyists have replaced the 6809 with the pin compatible Hitachi 6309EP, which offers higher performance, more registers, and many more opcodes. For details on the 6809 and 6309 see http://www.lomont.org/Software/Misc/CoCo/Lomont_6809.pdf.

Color Computer 1/2 Hardware Topics (PIA, VDG, SAM)

The main hardware interfaces in the CoCo 1 and 2 (also in the CoCo 3) are the 2 PIAs (Peripheral Interface Adapter), a SAM (Synchronous Address Multiplexer) and a VDG (Video Display Generator). Details follow.

The CoCo 1 came in 4K, 16K, and 32K RAM versions, with RAM starting at address $0000 and going through $1FFF, $3FFF, and $7FFF respectively. ROM addresses are $8000-$FFFF, with addresses $FF00-$FFFF being hardware access ports.

The CoCo 2 came with 64K of RAM, and 32K of ROM. The upper 32K was selected to be RAM or ROM by setting a bit in $FFDE/$FFDF.

The CoCo 2 has a RAM/ROM mode, and an all RAM mode, selected by SAM control bit TY, accessed from $FFDE/$FFDF.

32K RAM $0000-$7FFF /32K ROM $8000-$FFFF or
64K RAM $0000-$FFFF (TODO - vectors?)

All 3 CoCos have hardware interface registers in the 256 bytes from $FF00-$FFFF.

PIA (Peripheral Interface Adapters)

The PIA is a Motorola MC6821 or MC6822. There are two PIA chips, PIA0 and PIA1, each consisting of 4 addresses. Each PIA has two data registers and two control registers.

PIA0 uses addresses $FF00-$FF03. Data registers $FF00 and $FF02 are mostly keyboard and printer interfaces, and control registers $FF01 and $FF03 handle horizontal and vertical sync interrupts and joystick direction.

PIA1 uses addresses $FF20-$FF23, handling cassette, printer, CoCo 1/2 video modes, audio, and cartridge info. Details are in the hardware section under the respective addresses.

TODO

VDG (Video Display Generator)

The VDG is a Motorola MC6847 (later, the enhanced MC6847T1), capable of displaying text and graphics contained within a roughly square display 256 pixels wide by 192 lines high. It is capable of displaying 9 colors: black, green, yellow, blue, red, buff, cyan, magenta, and orange. It can generate a few modes: text modes, graphics modes, and "semigraphics" modes. The semigraphics modes replace each character position from a text mode with blocks containing pixels.

The CoCo is physically wired such that its default alphanumeric display is semigraphics-4 mode.

In alphanumeric mode, each character is a 5 dot wide by 7 dot high character in a box 8 dots wide and 12 lines high. This display mode consumes 512 bytes of memory and is a 32 character wide screen with 16 lines. The internal ROM character generator only holds 64 characters, so no lower case characters are provided. Lower case is instead "simulated" by inverting the color of the character.

Semigraphics is a hybrid display mode where alphanumerics and block graphics can be mixed together on the same screen. See other sections for details.

By setting the SAM such that it believes it is displaying a full graphics mode, but leaving the VDG in Alphanumeric/Semigraphics 4 mode, it is possible to subdivide the character box into smaller pieces. This creates the "virtual" modes Semigraphics 8, 12, and 24. These modes were not implemented on the CoCo 3.

There were several full graphics display modes, -C (for "color) modes and -R (for "resolution") modes. See elsewhere in this document for details.

The 256x192 two-color mode allows "artifact colors" on an NTSC TV, due to limitations of the phase relationship between the VDG clock and colorburst signal. In the white and black colorset, alternating dots bleed together to give red or blue, in effect giving a 128x192 four color mode with red, black, white, and blue. Reversing dot order reverses artifact colors. However, the color formed is somewhat random on RESET, so many games have the player press RESET until the colors are correct for the game. The CoCo 3 fixed this problem, always starting the same, and holding F1 during reset would reverse the colors. Artifacting does not work on the RGB monitors.

Graphics modes are covered in the section on Graphics Modes.

The VDG is programmed through PIA1.

SAM (Synchronous Address Multiplexer)

The SAM is a Motorola MC6883 or SN74LS785.

The SAM performs the following functions:

- Clock generation and synchronization for the 6809E CPU and 6847 VDG
- Up to 64K Dynamic Random Access Memory (DRAM) control and refresh
- Device selection based on CPU memory address to determine if the CPU access is to DRAM, ROM, PIA, etc.
- Duplication of the VDG address counter to "feed" the VDG the data it is expecting
- Divides the internal 4x NTSC freq (14.31818MHz for NTSC) by 4, passes it to the VDG for its own internal timing (3.579545MHz for NTSC).
- Divides the master clock by 16 (or 8 in certain cases) for the two phase CPU clock - in NTSC this is .89MHz (or 1.8MHz if div by 8).

The SAM's 16-bit configuration register is spread across 32 memory addresses ($FFC0-$FFDF). Writing even addresses sets that register bit to 0; writing to odd addresses sets it to 1.

The SAM contains a duplicate of the VDG's 12-bit address counter, and usually is programmed to be in sync. Mixing modes between the two results in other possible modes.

There are actually three speed settings on the CoCo. The default is to run at .895 MHz all the time. There is another setting that makes it run at twice that speed when accessing the ROM memory, but still at the slower speed when accessing RAM, called "address dependent" or "AD". Finally, there is a speed setting that uses the double speed all the time.

When the CPU chip runs faster, it generates more heat. Most CoCo 6809's can take the heat of running double-speed, but some might burn out, so do it at your own risk, especially on older, early version CoCos.

Note that a lot of the timing-dependent things in the CoCo BASIC ROMs won't work right at any speed other than "slow", like reading or writing cassettes and disks, and making sounds. On CoCo 1 and 2 it also causes problems with the display.

Clock speed is controlled by addresses $FFD6-$FFD9 in the SAM.

TODO

Color Computer 3 Hardware Topics (GIME)

The CoCo 3 supports the hardware of the CoCo 1 and 2, and adds a multifunction chip, the GIME. There were two versions, the 1986 and 1987 versions.

TODO – version differences? 256 color mode conjecture?

GIME (Graphics Interrupt Memory Enhancement)

The GIME is a custom ASIC chip designed to replace and extend many parts in the original CoCo 1 and 2. The main features added are support for more than 64K of memory (128K was the standard, and a 512K upgrade was common), advanced graphics modes, and more interrupt options. A mode bit (bit 7) in INIT0 ($FF90) (bit 7) switched between CoCo 1/2 mode and CoCo 3 mode.

There are many other features, covered in the hardware section for the GIME, which uses hardware registers $FF90-$FFBF. Here are a few features.

The GIME adds

- Many more graphics and text options.
- New interrupt sources, like timer and keyboard.
- Ability to address more memory (128K in original CoCo 3's, 512K after upgrade. There are other, bigger upgrades available). The Memory Management Unit (MMU, registers $FFA0-$FFAF) handles this by paging 8K blocks into the address space used by the CPU

MMU (Memory Management Unit)

The first thing to learn about the GIME is to understand the MMU, and how addresses from the GIME memory space map to the CPU memory space. The MMU is controlled by addresses $FFA0-$FFAF, and more details are under the section on Memory Mapping.

Graphics

See the GIME hardware section.
TODO – table?

Palettes

See the GIME hardware section on Palettes ($FFB0-$FFBF) and the Colors section.
TODO – palettes?

Interrupts

See the GIME hardware section on interrupts.
TODO

CoCo 1/2 compatibility

A mode bit (bit 7) in INIT0 ($FF90) (bit 7) switched between CoCo 1/2 mode and CoCo 3 mode. To use CoCo 1/2 graphics modes, set this bit. To use CoCo 3 graphics modes, clear this bit.

Memory Mapping

This section covers how memory is mapped into the CPU space on the CoCo 1, 2, and 3.

CoCo 1/2:

32/64K maps see TODO
TODO

CoCo 2 memory map

Address	Usage
$0000-$0069	Direct Page RAM: available for M/L Programs
$006A-$00FF	Internal Use
$006F	DEVNUM: 0=screen, FE=printer
$007D	BLKLEN: number of bytes in a block (0-255)
$007C	BLKTYP: block type: 0=header, 1=data, FF=end
$007E	CBUFAD: buffer address (two bytes?)
$0081	CSRERR: cassette error code
$0100-$0111	Interrupt Vectors
$0112-$0114	USRJMP - Jump to BASIC USR function
$0115-$0119	Unassigned - available for M/L Programs
$011A	Keyboard Alpha Lock (0 = not locked; FF = Locked)
$011B-$011C	Keyboard Delay Constant

$011D-$0151	Unassigned - available for M/L Programs
$0152-$0159	Keyboard Rollover Tables
$015A-$015D	POTVAL: Joystick values $15A Right joystick, left/right value $15B Right joystick, up/down value $15C Left joystick, left/right value $15D Left joystick, up/down value
$015E - $03FF	Internal Use
$0400-$05FF	Video Text memory
$0600-$1FFF $0600-$3FFF $0600-$7FFF	User's BASIC program (4K RAM) User's BASIC program (16K RAM) User's BASIC program (32K or 64K RAM)
$8000-$9FFF	Extended Color BASIC ROM
$A000-$BFFF	Color BASIC
$C000-$DFFF	Cartridge ROM Space
$E000-$FEFF	Unused
$FF00-$FFFF	I/O, machine configuration, reset vectors

CoCo 3

The GIME chip can access 512K of memory, yet the 6809 CPU can only access 64K. The barrier is broken by a MMU (Memory Management Unit) that splits the access into 8 blocks visible to the CPU of 8K each.

To use the memory mapping, bit 6 of INIT0 ($FF90) must be set to 1,

There are two possible memory maps, Map 0 and Map 1, selected by bit 0 of the INIT1 ($FF91) register. Setting this bit to 0 enables Map 0 (using the pages stored in $FFA0-$FFA7), and setting this bit to 1 enables Map 1 (using pages in $FFA8-$FFAF).

TODO see http://www.coco25.com/wiki/index.php/MMU_RAMROM_Mode

A memory page is an 8K block in the GIME address space. A 128K system has 128/8=16 blocks, numbered hex $30-$3F. A 512K system has 64 blocks, numbered hex $00-$3F. To place a page in CPU memory for access, write the page number in the appropriate memory select register.

In RAM/ROM mode, the ROM pages ($3C-$3F) can be written to any of the eight available MMU slots. In all cases the last two bits of the page are ignored by the MMU and substituted by the last two bits of the slot number. This might cause some addressing confusion, and should be noted.

A memory page number is a 6-bit value. When reading the memory select registers, be sure to mask off the top two bits, since they can contain garbage.

The memory select registers are registers $FFA0-$FFAF. A write of a page value to the address on the left makes the page visible at the CPU address on the right.

Map 0	Map 1

$FFA0 -> $0000-$1FFF	$FFA8 -> $0000-$1FFF
$FFA1 -> $2000-$2FFF	$FFA9 -> $2000-$2FFF
$FFA2 -> $4000-$5FFF	$FFAA -> $4000-$5FFF
$FFA3 -> $6000-$7FFF	$FFAB -> $6000-$7FFF
$FFA4 -> $8000-$9FFF	$FFAC -> $8000-$9FFF
$FFA5 -> $A000-$BFFF	$FFAD -> $A000-$BFFF
$FFA6 -> $C000-$DFFF	$FFAE -> $C000-$DFFF
$FFA7 -> $E000-$FFFF	$FFAF -> $E000-$FFFF

Details are in the hardware reference for the MMU.

Example: to set GIME memory location $60000 to value 0, you could:

```
ORCC    #$50     SHUT OFF INTERRUPTS - TODO - SAVE FOR RESTORE LATER
LDA     $FFA1    GET THE PAGE FOR THE RESTORE
ANDA    #63      STRIP OFF TOP BITS
PSHS    A        SAVE THE PAGE FOR LATER
LDA     #$30     ACCESS TO PAGE $30 = GIME $60000
STA     $FFA1    MAP PAGE $60000-$61FFF TO LOCATIONS $2000-$3FFF
LDA     #$00     THE BYTE IS 0
STA     $2000    SET THE PROPER BYTE IN CPU SPACE
PULS    A        RESTORE THE PAGE VALUE THAT WAS THERE
STA     $FFA1    MAP ORIGINAL PAGE BACK INTO CPU SPACE
```

Notes:

1. Unless you know what you are doing, shut off interrupts when changing pages. If you change a page that has an interrupt handler in it, and an interrupt occurs, you will likely crash the computer.
2. If you are using the stack, be careful if you page out the stack. Return addresses may be changed, and stack values will not likely be the same. Therefore, KNOW WHERE THE STACK IS! In basic, it starts in the $6000-$7FFF page.

Simple CoCo 3 Memory Map

Here is a simple CoCo 3 memory map. Detailed versions are in the section on Detailed Memory Maps.

Here are page values for GIME address, default page values on a power up, and default CPU addresses:

Page	GIME Address	CPU Address	Standard Page Contents
$00-2F	00000-$5FFFF		512K upgrade RAM, unused by BASIC; not present in 128K or smaller systems
$30	$60000-$61FFF		Hi-Res page #1
$31	$62000-$63FFF		Hi-Res page #2
$32	$64000-$65FFF		Hi-Res page #3
$33	$66000-$67FFF		Hi-Res page #4
$34	$68000-$69FFF		HGET/HPUT buffer
$35	$6A000-$6BFFF		Secondary Stack
$36	$6C000-$6DFFF		Hi-Res text screen RAM

$37	$6E000-$6FFFF		Unused by BASIC
$38	$70000-$71FFF	$0000-$1FFF	BASIC memory
$39	$72000-$73FFF	$2000-$3FFF	BASIC memory
$3A	$74000-$75FFF	$4000-$5FFF	BASIC memory
$3B	$76000-$77FFF	$6000-$7FFF	BASIC memory
$3C	$78000-$79FFF	$8000-$9FFF	Extended Basic ROM
$3D	$7A000-$7BFFF	$A000-$BFFF	Color Basic ROM
$3E	$7C000-$7DFFF	$C000-$DFFF	Cartridge or Disk Basic ROM
$3F	$7E000-$7FFFF	$D000-$FFFF	Super Basic, GIME regs, I/O, Interrupts

A little more detail for the default power on situation for the BASIC memory sections, and
GIME Address *Contents*
$70000 - $703FF System RAM
$70400 - $705FF Lowres text screen
Non Disk System
$70600 - $70BFF Page 1 - lowres graphics
$70C00 - $711FF Page 2
$71200 - $717FF Page 3
$71800 - $71DFF Page 4
$71E00 - $723FF Page 5
$72400 - $729FF Page 6
$72A00 - $72FFF Page 7
$73000 - $735FF Page 8
Disk System
$70600 - $70DFF Disk System RAM
$70E00 - Page 1
Either System 1 - 8 graphic pages reserved, Basic program start varies.
$71200 - $77FFF
or Basic programs, variables, and user ml programs
$71400 - $77FFF

High pages:
$7E000 - $7FDFF Super Extended Basic
$7FE00 - $7FEFF Secondary vector table
$7FF00 - $7FF3F PIAs
$7FF90 - $7FBFF GIME in CoCo 3
$7FFC0 - $7FFDF video control, clock, and map type
$7FFE0 - $7FFF1 Unused
$7FFF2 - $7FFFF Interrupt vectors

Note: the Vector Page RAM at $7FE00 - $7FEFF (when enabled), will appear instead of the RAM or ROM at $FE00 - $FEFF. (see INIT0 ($FF90) Bit 3) TODO

The 256 top bytes $FF00-$FFFF in CPU space contain byte-mapped hardware interfaces, covered elsewhere in this doc.

TODO - make sure all this in detailed maps
TODO - Merge memory maps into one section, with two or three levels of detail.

Colors

CoCo 1/2:

TODO

CoCo 3:

Palette colors are defined in registers $FFB0-$FFBF. The format differs depending on if you are in RGB or Composite monitor mode. Mode is selected by setting TODO

Default composite colors on startup:

```
$FFB0 GREEN     18      $FFB8 BLACK      00
$FFB1 YELLOW    36      $FFB9 GREEN      18
$FFB2 BLUE      11      $FFBA BLACK      00
$FFB3 RED       07      $FFBB BUFF       63
$FFB4 BUFF      63      $FFBC BLACK      00
$FFB5 CYAN      31      $FFBD GREEN      18
$FFB6 MAGENTA   09      $FFBE BLACK      00
$FFB7 ORANGE    38      $FFBF ORANGE     38
```

Entering PALETTE CMP or PALETTE RGB will set this palette for the type of monitor you are using.

The format is explained in the GIME Palette ($FFB0-$FFBF) register section.

The table of (hex) colors given below is the conversion used in OS-9 Level II.

```
Monitor Color                     Monitor Color
RGB  CMP                          RGB  CMP
00   00   Black                   32   23   Medium intensity red
01   12   Low intensity blue      33   08   Blue tint red
02   02   Low intensity green     34   21   Light Orange
03   14   Low intensity cyan      35   06   Cyan tint red
04   07   Low intensity red       36   39   Full intensity red
05   09   Low intensity magenta   37   24   Magenta tint red
06   05   Low intensity brown     38   38   Brown tint red
07   16   Low intensity white     39   54   Faded red
08   28   Medium intensity blue   40   25   Medium intensity magenta
09   44   Full intensity blue     41   42   Blue tint magenta
10   13   Green tint blue         42   26   Green tint magenta
11   29   Cyan tint blue          43   58   Cyan tint magenta
12   11   Red tint blue           44   24   Red tint magenta
13   27   Magenta tint blue       45   41   Full intensity magenta
14   10   Brown tint blue         46   40   Brown tint magenta
15   43   Faded blue              47   56   Faded magenta
16   34   Medium intensity green  48   20   Medium intensity yellow
17   17   Blue tint green         49   04   Blue tint yellow
```

18	18	Full intensity green	50	35	Green tint yellow
19	33	Cyan tint green	51	51	Cyan tint yellow
20	03	Red tint green	52	37	Red tint yellow
21	01	Magenta tint green	53	53	Magenta tint yellow
22	19	Brown tint green	54	36	Full intensity yellow
23	50	Faded green	55	52	Faded yellow
24	30	Medium intensity cyan	56	32	Medium intensity white
25	45	Blue tint cyan	57	59	Light blue
26	31	Green tint cyan	58	49	Light green
27	46	Full intensity cyan	59	62	Light cyan
28	15	Red tint cyan	60	55	Light red
29	60	Magenta tint cyan	61	57	Light magenta
30	47	Brown tint cyan	62	63	Light yellow
31	61	Faded cyan	63	48	White

TODO – there is a dup in the entries – check elsewhere, and also make reverse table. 24 is used twice in the CMP side, and 22 is missed on the CMP side.

Graphics Modes

CoCo 1/2

TODO - table? From http://homepage.ntlworld.com/kryten_droid/coco/coco_tm_s3.htm ?

TODO - add semigraphics 8, 12, and 24 modes info?

ALPHANUMERIC DISPLAY MODES – All alphanumeric modes occupy an 8 x 12 dot character matrix box and there are 32 x 16 character boxes per TV frame. Each horizontal dot (dot-clock) corresponds to one half the period duration of the 3.58 MHz clock and each vertical dot is one scan line. One of two colors for the lighted dots may be selected by the color set select pin (pin 39). An internal ROM will generate 64 ASCII display characters in a standard 5 x 7 box. Six bits of the eight-bit data word are used for the ASCII character generator and the two bits not used are used to implement inverse video and mode switching to semigraphics – 4, – 8, – 12, or – 24.

The ALPHA SEMIGRAPHICS – 4 mode translates bits 0 through 3 into a 4 x 6 dot element in the standard 8 x 12 dot box. Three data bits may be used to select one of eight colors for the entire character box. The extra bit is used to switch to alphanumeric. A 512 byte display memory is required. A density of 64 x 32 elements is available in the display area. The element area is four dot-clocks wide by six lines high.

The ALPHA SEMIGRAPHICS – 6 mode maps six 4 x 4 dot elements into the standard 8 x 12 dot alphanumeric box, a screen density of 64 x 48 elements is available. Six bits are used to generate this map and two data bits may be used to select one of four colors in the display box. A 512 byte display memory is required. The element area is four dot-clocks wide by four lines high.

The ALPHA SEMIGRAPHICS – 8 mode maps eight 4 x 3 dot elements into the standard 8 x 12 dot box. This mode requires four memory locations per box and each memory location may specify one of eight colors or black. A 2048 byte display memory is required. A density of 64 x 64 elements is available in the display area. The element area is four dot-clocks wide by three lines high.

The ALPHA SEMIGRAPHICS – 12 mode maps twelve 4 x 2 dot elements into the standard 8 x 12 dot box. This mode requires six memory locations per box and each memory location may specify one of eight colors or black. A 3072 byte display memory is required. A density of 64 x 96 elements is available in the display area. The element area is four dot-clocks wide by two lines high.

The ALPHA SEMIGRAPHICS – 24 mode maps twenty-four 4 x 1 dot elements into the standard 8 x 12 dot box. This mode requires twelve memory locations per box and each memory location may specify one of eight colors or black. A 6144 byte display memory is required. A density of 64 x 192 elements is available in the display are. The element area is four dot-clocks wide by one line high.

FULL GRAPHIC MODES – There are eight full graphic modes available from the VDG. These modes require 1K to 6K bytes of memory. The eight full-graphic modes include an outside color border in one of two colors depending upon the color set select pin (CSS). The CSS pin (pin 39) selects one of two sets of four colors in the four color graphic modes.

The 64 x 64 Color Graphics mode generates a display matrix of 64 elements wide by 64 elements high. Each element may be one of four colors. A 1K x 8 display memory is required. Each pixel equals four dot-clocks by three scan lines.

The 128 x 64 Graphics Mode generates a matrix 128 elements wide by 64 elements high. Each element may be either ON or OFF. However, the entire display may be one of two colors, selected by using the color set select pin. A 1K x 8 display memory is required. Each pixel equals two dot-clocks by three scan lines.

The 128 x 64 Color Graphics mode generates a display matrix 128 elements wide by 64 elements high. Each element may be one of four colors. A 2K x 8 display memory is required. Each pixel equals two dot-clocks by three scan lines.

The 128 x 96 Graphics mode generates a display matrix 128 elements wide by 96 elements high. Each element may be either ON or OFF. However, the entire display may be one of two colors selected by using the color select pin. A 2K x 8 display memory is required. Each pixel equals two dot-clocks by two scan lines.

The 128 x 96 Color Graphics mode generates a display 128 elements wide by 96 elements high. Each element may be one of four colors. A 3K x 8 display memory is required. Each pixel equals two dot-clocks by two scan lines.

The 128 x 192 Graphics mode generates a display matrix 128 elements wide by 192 elements high. Each element may be either ON or OFF, but the ON elements may be one of two colors selected with color set select pin. A 3K x 8 display memory is required. Each pixel equals two dot-clocks by one scan line.

The 128 x 192 Color Graphics mode generates a display 128 elements wide by 192 elements high. Each element may be one of four colors. A 6K x 8 display memory is required. A detailed description of the VDG modes is given in Table 2. Each pixel equals two dot-clocks by one scan line.

The 256 x 192 Graphics mode generates a display 256 elements wide by 192 elements high. Each element may be either ON or OFF, but the ON element may be one of two colors selected with the color set select pin. A 6K x 8 display memory is required. Each pixel equals one dot-clock by one scan line.

<table>
<tr><th colspan="2">VDG PINS</th><th colspan="3">COLOR</th><th colspan="2">TV SCREEN</th><th rowspan="2">VDG DATA BUS</th><th rowspan="2">COMMENTS</th></tr>
<tr><th>CSS</th><th>INV</th><th>Character Color</th><th>Background</th><th>Border</th><th>Display Mode</th><th>Detail</th></tr>
<tr><td>Ø</td><td>Ø
1</td><td>Green
Black</td><td>Black
Green</td><td>Black</td><td>32 Characters in columns</td><td rowspan="2">8 dots
12 dots
7
5</td><td rowspan="2">| Ø | INV | | | | | | |
ASCII code</td><td rowspan="2">The ALPHANUMERIC INTERNAL mode uses an internal character generator which contains the following five dot by seven dot characters: @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [/] ↑ ← SP ! " # $ % & * + , = / 0 1 2 3 4 5 6 7 8 9 : ; = ? .</td></tr>
<tr><td>1</td><td>Ø
1</td><td>Orange
Black</td><td>Black
Orange</td><td>Black</td><td>16 Characters in rows</td></tr>
<tr><td>X</td><td>X</td><td colspan="2">Lx C2 C1 CØ Color
Ø X X X Black
1 Ø Ø Ø Green
1 Ø Ø 1 Yellow
1 Ø 1 Ø Blue
1 Ø 1 1 Red
1 1 Ø Ø Buff
1 1 Ø 1 Cyan
1 1 1 Ø Magenta
1 1 1 1 Orange</td><td>Black</td><td>64 Display elements in columns
32 Display elements in rows</td><td>| L3 | L2 |
| L1 | LØ |
one element</td><td>| 1 | C2 | C1 | CØ | L3 | L2 | L1 | LØ |</td><td>The SEMIGRAPHICS FOUR mode uses an internal "coarse graphics" generator in which a rectangle (eight dots by twelve dots) is divided into four equal parts. The luminance of each part is determined by a corresponding bit on the VDG data bus. The color of illuminated parts is determined by three bits. It requires 512 bytes of display memory.</td></tr>
<tr><td>Ø</td><td rowspan="2">X</td><td colspan="2">Lx C1 CØ Color
Ø X X Black
1 Ø Ø Green
1 Ø 1 Green
1 1 Ø Blue
1 1 1 Red</td><td rowspan="2">Black</td><td>64 Display elements in columns</td><td rowspan="2">| L5 | L4 |
| L3 | L2 |
| L1 | LØ |</td><td rowspan="2">| C1 | CØ | L5 | L4 | L3 | L2 | L1 | LØ |</td><td rowspan="2">The SEMIGRAPHIC SIX mode is similar to the SEMIGRAPHIC FOUR mode with the following difference: The eight dot by twelve dot rectangle is divided into six equal parts. Color is determined by the two remaining bits. It requires 512 bytes of display memory.</td></tr>
<tr><td>1</td><td colspan="2">Ø X X Black
1 Ø Ø Buff
1 Ø 1 Buff
1 1 Ø Magenta
1 1 1 Orange</td><td>48 Display elements in rows</td></tr>
<tr><td>X</td><td>X</td><td colspan="2">Lx C2 C1 CØ Color
Ø X X X Black
1 Ø Ø Ø Green
1 Ø Ø 1 Yellow
1 Ø 1 Ø Blue
1 Ø 1 1 Red
1 1 Ø Ø Buff
1 1 Ø 1 Cyan
1 1 1 Ø Magenta
1 1 1 1 Orange</td><td>Black</td><td>64 Display elements in columns
64 Display elements in rows</td><td>| L1 | LØ |
L3	L2
L5	L4
L7	L6 |</td><td>| 1	C2	C1	CØ	L1	LØ	X	X
1	C1	C1	CØ	L3	L2	X	X
1	C2	C1	CØ	X	X	L5	L4
1	C2	C1	CØ	X	X	L7	L6	</td><td>The SEMIGRAPHIC EIGHT mode requires four column consecutive addresses*, and produces a 2 x 4 block. It requires 2048 bytes of display memory.</td></tr>
<tr><td>X</td><td>X</td><td colspan="2">Lx C2 C1 CØ Color Ø X X X Black 1 Ø Ø Ø Green 1 Ø Ø 1 Yellow 1 Ø 1 Ø Blue 1 Ø 1 1 Red 1 1 Ø Ø Buff 1 1 Ø 1 Cyan 1 1 1 Ø Magenta 1 1 1 1 Orange</td><td>Black</td><td>64 Display elements in columns 96 Display elements in rows</td><td>| L1	LØ
L3	L2
L5	L4
L7	L6
L9	L8
L11	L10 |</td><td>| 1	C2	C1	C2	L1	LØ	X	X
1	C2	C1	CØ	L3	L2	X	X
1	C2	C1	CØ	L5	L4	X	X
1	C2	C1	CØ	X	X	L7	L6
1	C2	C1	CØ	X	X	L9	L8
1	C2	C1	CØ	X	X	L11	L10	</td><td>The SEMIGRAPHIC TWELVE mode requires six column consecutive addresses*, and produces a 2 x 6 block. It requires 3072 bytes of display memory.</td></tr>
</table>

*Four column consecutive addresses starting at HEX Ø4ØØ are Ø4ØØ, Ø42Ø, Ø44Ø, Ø46Ø.

25

VDG PINS		COLOR		TV SCREEN		VDG DATA BUS	COMMENTS
CSS	INV	Character Color / Background	Border	Display Mode	Detail		
X	X	Lx C2 C1 CØ Color Ø X X X Black 1 Ø Ø Ø Green 1 Ø Ø 1 Yellow 1 Ø 1 Ø Blue 1 Ø 1 1 Red 1 1 Ø Ø Buff 1 1 Ø 1 Cyan 1 1 1 Ø Magenta 1 1 1 1 Orange	Black	64 Display elements in columns 192 Display elements in rows	L_1 L_0 L_3 L_2 L_5 L_4 L_7 L_6 L_9 L_8 L_{11} L_{10} L_{13} L_{12} L_{15} L_{14} L_{17} L_{16} L_{19} L_{18} L_{21} L_{20} L_{23} L_{22}	1 C_2 C_1 C_0 L_1 L_0 X X 1 C_2 C_1 C_0 L_3 L_2 X X 1 C_2 C_1 C_0 L_5 L_4 X X 1 C_2 C_1 C_0 L_7 L_6 X X 1 C_2 C_1 C_0 L_9 L_8 X X 1 C_2 C_1 C_0 L_{11} L_{10} X X 1 C_2 C_1 C_0 X X L_{13} L_{12} 1 C_2 C_1 C_0 X X L_{15} L_{14} 1 C_2 C_1 C_0 X X L_{17} L_{16} 1 C_2 C_1 C_0 X X L_{19} L_{18} 1 C_2 C_1 C_0 X X L_{21} L_{20} 1 C_2 C_1 C_0 X X L_{23} L_{22}	The SEMIGRAPHIC TWENTY FOUR mode requires twelve column consecutive addresses*, and produces a 2 x 12 block. It requires 6144 bytes of display memory.
Ø 1	X	C1 CØ Color Ø Ø Green Ø 1 Yellow 1 Ø Blue 1 1 Red Ø Ø Buff Ø 1 Cyan 1 Ø Magenta 1 1 Orange	Green Buff	64 Display elements in columns 64 Display elements in rows	E_3 E_2 E_1 E_0	C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The GRAPHICS ONE C mode uses a maximum of 1024 bytes of display RAM in which one pair of bits specifies one picture element.
Ø 1	X	Lx Color Ø Black 1 Green Ø Black 1 Buff	Green Buff	128 Display elements in columns 64 Display elements in rows	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The GRAPHICS ONE R mode uses a maximum of 1024 bytes of display RAM in which one bit specifies one picture element.
Ø 1	X	Same colors as Graphics one C	Green Buff	128 Display elements in columns 64 Display elements in rows	E_3 E_2 E_1 E_0	C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The GRAPHICS TWO C mode uses a maximum of 2048 bytes of display RAM in which one pair of bits specifies one picture element.
Ø 1	X	Same colors as Graphics one R	Green Buff	128 Display elements in columns 96 Display elements in rows	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The GRAPHICS TWO R mode uses a maximum of 1536 bytes of display RAM in which one bit specifies one picture element.
Ø 1	X	Same colors as Graphics one C	Green Buff	128 Display elements in columns 96 Display elements in rows	E_3 E_2 E_1 E_0	C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The GRAPHICS THREE C mode uses a maximum of 3072 bytes of display RAM in which one pair of bits specifies one picture element.
Ø 1	X	Same colors as Graphics one R	Green Buff	128 Display elements in columns 192 Display elements in rows	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The GRAPHICS THREE R mode uses a maximum of 3072 bytes of display RAM in which one bit specifies one picture element.
Ø 1	X	Same colors as Graphics one C	Green Buff	128 Display elements in columns 192 Display elements in rows	E_3 E_2 E_1 E_0	C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The GRAPHICS SIX C mode uses a maximum of 6144 bytes of display RAM in which one pair of bits specifies one picture element.
Ø 1	X	Same colors as Graphics one R	Green Buff	256 Display elements in columns 192 Display elements in rows	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The GRAPHICS SIX R mode uses a maximum of 6144 bytes of display RAM in which one bit specifies one picture element.

*Four column consecutive addresses starting at HEX Ø4ØØ are Ø4ØØ, Ø42Ø, Ø44Ø, Ø46Ø.

CoCo 3

TODO - table?

See throughout this document
TODO - table?

Text Modes

CoCo 1/2

The character set available for CoCo 1/2 or CoCo 3 in CoCo 1/2 compatible text mode (WIDTH 32) follows.

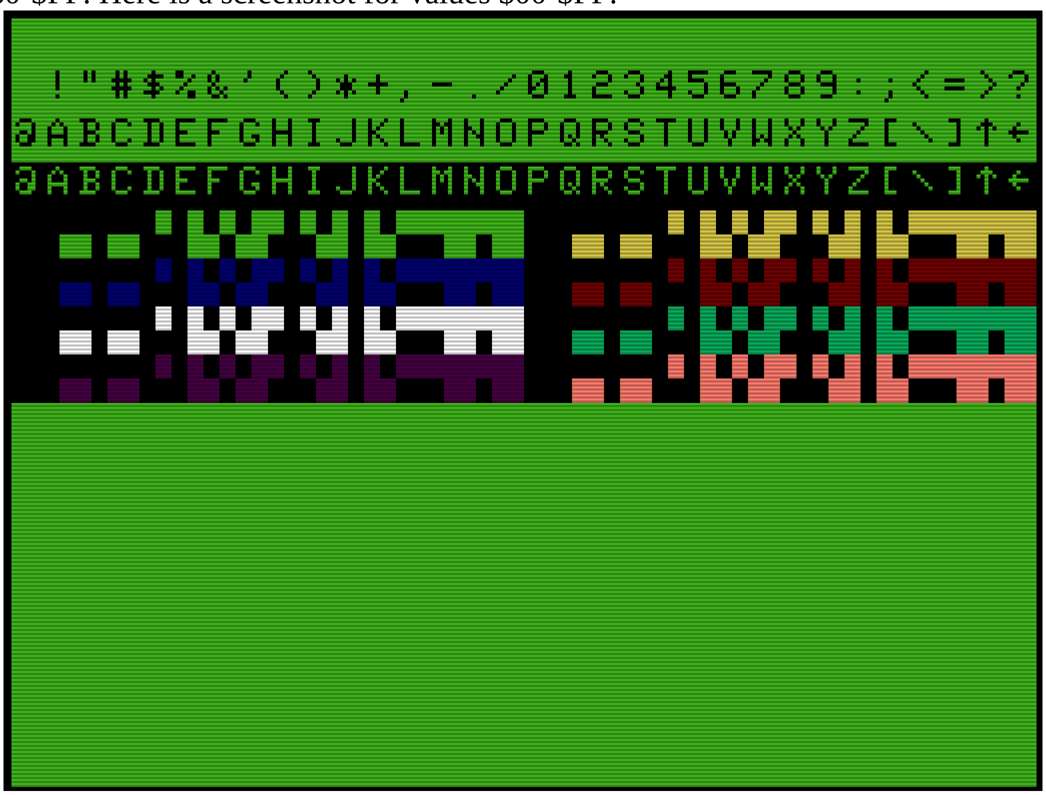
On CoCo 3(?) the character set assumes that bit 4 of $FF22 is set. If that bit is clear, then the characters in the range of $0-$1F must be replaced by the corresponding characters in the range $40-$5F in inverse video.

CoCo 1 and 2, and CoCo3 WIDTH 32 character set:

Each entry is hex for Inverted, NonInverted, Text

```
$00 $40 @    $10 $50 P    $20 $60      $30 $70 0
$01 $41 A    $11 $51 Q    $21 $61 !    $31 $71 1
$02 $42 B    $12 $52 R    $22 $62 "    $32 $72 2
$03 $43 C    $13 $53 S    $23 $63 #    $33 $73 3
$04 $44 D    $14 $54 T    $24 $64 $    $34 $74 4
$05 $45 E    $15 $55 U    $25 $65 %    $35 $75 5
$06 $46 F    $16 $56 V    $26 $66 &    $36 $76 6
$07 $47 G    $17 $57 W    $27 $67 '    $37 $77 7
$08 $48 H    $18 $58 X    $28 $68 (    $38 $78 8
$09 $49 I    $19 $59 Y    $29 $69 )    $39 $79 9
$0A $4A J    $1A $5A Z    $2A $6A *    $3A $7A :
$0B $4B K    $1B $5B [    $2B $6B +    $3B $7B ;
$0C $4C L    $1C $5C \    $2C $6C ,    $3C $7C <
$0D $4D M    $1D $5D ]    $2D $6D -    $3D $7D =
$0E $4E N    $1E $5E up   $2E $6E .    $3E $7E >
$0F $4F O    $1F $5F left $2F $6F /    $3F $7F ?
```

The characters defined by $20-$3F are inverse video. Graphics blocks are printed for character values $80-$FF. Here is a screenshot for values $00-$FF:

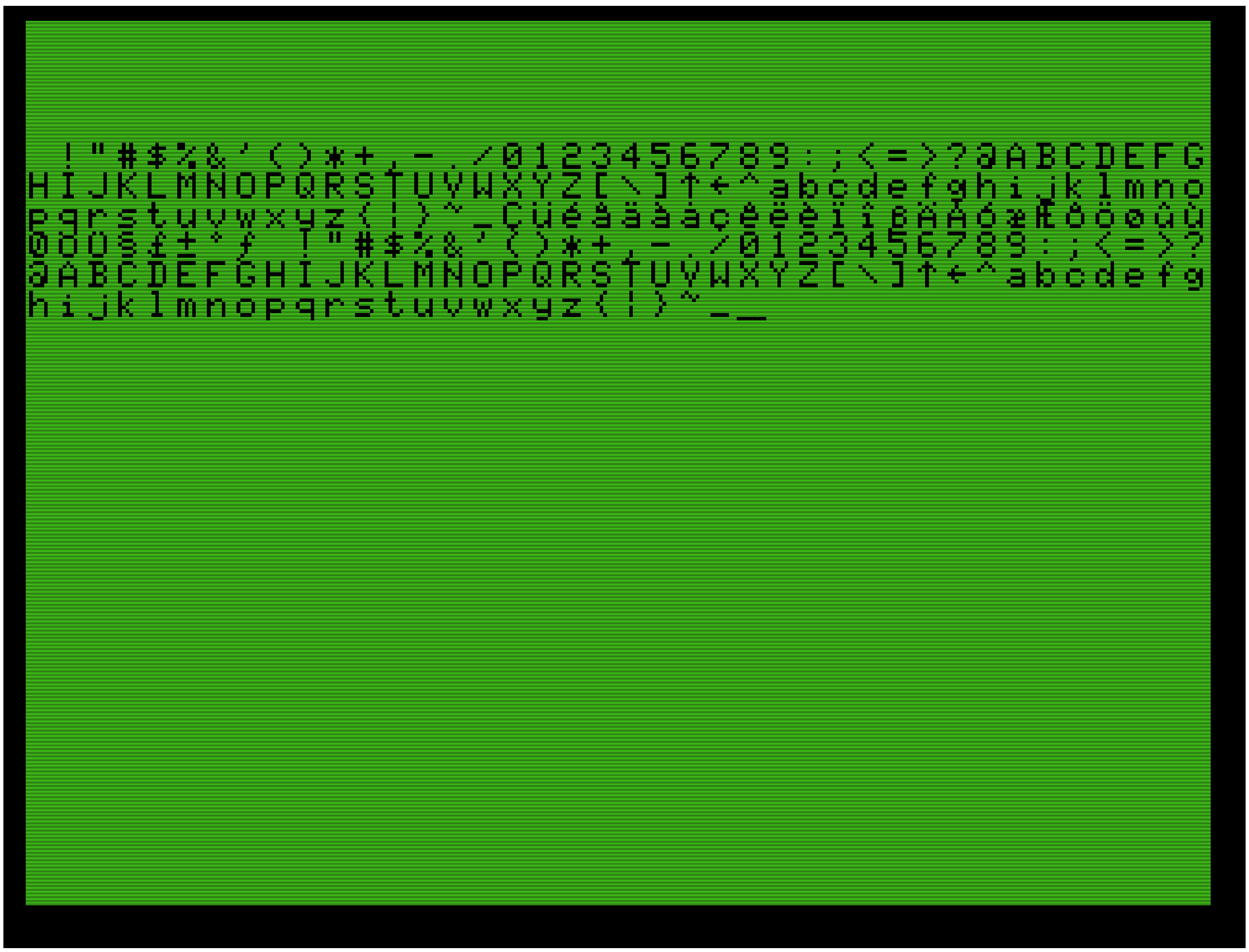


CoCo 3

CoCo 3 high-resolution text modes (WIDTH 40, 80) have more characters. The character set is repeated for character values $80-$FF.

```
$00 Ç   $10 ó   $20     $30 0   $40 @   $50 P   $60 ^   $70 p
```

$01 ü	$11 æ	$21 !	$31 1	$41 A	$51 Q	$61 a	$71 q
$02 é	$12 Æ	$22 "	$32 2	$42 B	$52 R	$62 b	$72 r
$03 â	$13 ô	$23 #	$33 3	$43 C	$53 S	$63 c	$73 s
$04 ä	$14 ö	$24 $	$34 4	$44 D	$54 T	$64 d	$74 t
$05 à	$15 ø	$25 %	$35 5	$45 E	$55 U	$65 e	$75 u
$06 å	$16 û	$26 &	$36 6	$46 F	$56 V	$66 f	$76 v
$07 ç	$17 ù	$27 '	$37 7	$47 G	$57 W	$67 g	$77 w
$08 ê	$18 Ø	$28 (	$38 8	$48 H	$58 X	$68 h	$78 x
$09 ë	$19 Ö	$29)	$39 9	$49 I	$59 Y	$69 I	$79 y
$0A è	$1A Ü	$2A *	$3A :	$4A J	$5A Z	$6A j	$7A z
$0B ï	$1B §	$2B +	$3B ;	$4B K	$5B [	$6B k	$7B {
$0C î	$1C £	$2C ,	$3C <	$4C L	$5C \	$6C l	$7C |
$0D ß	$1D ±	$2D -	$3D =	$4D M	$5D]	$6D m	$7D }
$0E Ä	$1E °	$2E .	$3D >	$4E N	$5E up	$6E n	$7E ~
$0F Å	$1F ƒ	$2F /	$3F ?	$4F O	$5F lft	$6F o	$7F _



For the CoCo3 hi-res screen modes, each character is 2 bytes, in the format char, attrib, char, attrib, etc. where char is a character code and attrib is an attribute byte. The attribute byte looks like this:

Bit 7	Bit 6	Bits 5,4,3	Bits 2,1,0
Flash (1=flash,0 = don't)	Underline (1=underline, 0 =	three foreground color bits (palettes 8-15)	three background color bits (palettes 0-7)

	don't)	$FFB8-$FFBF	$FFB0-$FFB7

Keyboard

The keyboard is accessed through PIA0, addresses $FF00-$FF03. Access is done by setting (for example) $FF00 for input, $FF02 for output, sending a signal down the required bit(s) in $FF02, and reading the inputs from $FF00. The roles of $FF00 and $FF02 can be reversed if desired.

The bit values seem backwards and are 0 for on, and 1 for off, in reading the keyboard.

Example code: Needs work on how to do keyboard: - clean this up and correct it

```
    CLR     $FF01          set $FF00 for direction
    CLR     $FF00          set for input
    CLR     $FF03          set $FF02 for direction
    LDA     #$FF
    STA     $FF02          set for output

    LDA     #%11101111     check only a single column number 4
    STA     $FF02          signal columns (in diagram below)
    LDA     $FF00          read rows (in diagram below)
    COMA                   invert output
    ANDA    #$7F           strip bit
    CMPA    #%011111011    check single bit 2 - we tested for T key
```

Here is the keyboard matrix. Some entries have multiple keys separated by a slash. For example, es/br is the Esc/Break key.

```
 ____________________________________________________________________________
|      LSB              $FF02               MSB                              |
|       0    1    2    3    4    5    6    7                                |
+----------------------------------------------------------------------------+
|       @    A    B    C    D    E    F    G        0       LSB             |
|                                                                            |
|       H    I    J    K    L    M    N    O        1                       |
|                                                                  $         |
|       P    Q    R    S    T    U    V    W        2        F              |
|                                                            F              |
|       X    Y    Z    up   dn   lf   rt   space    3        0              |
|                                                            0              |
|       0    !/1  "/2  #/3  $/4  %/5  &/6  '/7      4                       |
|                                                                            |
|      (/8   )/9  */:  +/;  </,  =/-  >/.  ? /      5                       |
|                                                                            |
|     enter clr es/br  alt  ctrl  F1  F2   shifts   6                       |
|     Joystick comparison result                    7       MSB             |
 ----------------------------------------------------------------------------
```

TODO – check kbd example, and make sure works, add joystick buttons to the mix

Joystick

PIA control registers at $FF01 and $FF03 set control registers CA2 and CB2, which in turn select which joystick to read and which axis to read.

The 6 most significant bits of $FF20 are the digital to analog converter, and any value here is compared to a joystick reading. The high bit of $FF00 will be 1 whenever the joystick value exceeds the D/A value. So set $FF20 to $FC (the highest possible), check the bit, and decrease the value until the bit changes, giving the joystick value.

Since these bits also affect sound, you should mute the CoCo first.

Example:
see http://www.coco25.com/wiki/index.php/Sampling TODO

In the CoCo 1 and 2, the joysticks are read by looking at the single bit output of a comparator; one input to that comparator is one of the joystick values, the other is the output of a digital-to-analog (D/A) converter under software control. When the selected joystick value exceeds the output of the D/A converter, the comparator outputs a one, any other time, if outputs a zero. So the joystick value itself isn't just sitting in memory anywhere; you have to select which joystick to use as input to the comparator, vary the input to the D/A converter, and then check the output of the comparator to see when it changes from a one to a zero or visa-versa.

The input to the D/A is the most-significant 6 bits at address $FF20; write $FC to that address to set the analog value as high as it will go. The comparator output can be read as the most-significant bit at address $FF00; when it's a one, the joystick value is higher than the D/A output.

Note that the remaining bits at those addresses are used for other things: $FF20 also has a bit going to the serial port output, and a bit coming in from the cassette port; while the rest of $FF00 is input from the keyboard matrix and joystick fire-buttons. So you should strip out the bits you need by doing the appropriate ANDs. Ignoring extraneous input bits causes no problem, and (in this case) setting the remaining output bits to zero should be okay. (Though you might want to leave the serial output line high; for that, just OR in $02 before writing the D/A value).

To select which joystick value gets compared (ie: left or right, and X or Y axis), you have to control a pair of special PIA outputs; CA2 and CB2. Each of the four possible values of these bits selects a different joystick line to compare. These are controlled by writing to the PIA control registers, which is a little complicated because they also control a bunch of other things. The control registers you want are at addresses $FF01 (for CA2) and $FF03 (for CB2). For each address, I *think* you want to write the value $3F to set the output to a one, and $37 to set it to zero. These settings enable interrupts that (I think) the CoCo normally uses, and triggers them on the rising edge of the input signals, which I also think is the normal setting. If these are wrong, please let me know!

Another possibility is that the PIA control registers might be readable. In that case, you would just want to read the control register, mask out the bits controlling the select line by ANDing against

$C7, and the fill in the new control values for those bits by ORing against $38 (for high output) or against $30 (for low).

Of course, the easiest way to do this is from BASIC is with the JOYSTK (I) function, where I is a joystick axis number. There are four axes you might want to read: each of the two joysticks has a left-right axis and an up-down axis. Read JOYSTK(0) to cache all four values (BASIC weirdness), and then read the other three values at your leisure.

The second easiest way is to call a ROM routine (POLCAT) that will read all four joystick values and leave them sitting in memory. This routine lives in the Color BASIC ROM at address $A00A and leaves its results in the four bytes at addresses $015A through $015D.

TODO – rewrite section, test, and minimize the example.

Mouse

This is the same as joystick programming.
TODO

Interrupts

CoCo 1/2/3

An interrupt is an external event that alters the normal flow of the microprocessor. There are many possible ways to generate interrupts. The 6809 has 4 hardware interrupts and 3 software interrupts. They are (listed in lowest to highest precedence):

Interrupt	Expanded notation	Default use	Registers pushed	Vector	Vector points to	Code at location
SWI3	Software Interrupt 3	not used?	A,B,X,Y,U,PC,DP,CC	$FFF2	$FEEE	LBRA $0100
SWI2	Software Interrupt 2	not used? some use in OS9?	A,B,X,Y,U,PC,DP,CC	$FFF4	$FEF1	LBRA $0103
FIRQ	Fast Interrupt Request	disk drive access	PC,CC	$FFF6	$FEF4	LBRA $010F
IRQ	Interrupt Request	sound and TIMER functions	A,B,X,Y,U,PC,DP,CC	$FFF8	$FEF7	LBRA $010C
SWI	Software Interrupt 1	unused in Basic, used in EDTASM	A,B,X,Y,U,PC,DP,CC	$FFFA	$FEFA	LBRA $0106
NMI	Non-Maskable Interrupt	not supported	A,B,X,Y,U,PC,DP,CC	$FFFC	$FEFD	LBRA $0109
RESET	Inital power up	resetting the	None	$FFFE	$8C1B	reset code

	and RESET button	machine				

When a FIRQ or IRQ interrupt fires, the microprocessor first sees if the corresponding bit in the Condition Code (CC) register in the 6809 microprocessor is 0. If it is, the exception processing is performed. The microprocessor gets the address to go to from the interrupt vectors, and jumps to the address stored there.

In the CoCo, each of the vectors is in ROM, and cannot be changed. However, the vectors each point to a RAM location that can be changed.

If there are multiple interrupts, only the highest priority one will be taken.

The FIRQ interrupt is fast in the sense that it does not push many registers on the stack.

For example, if an IRQ occurred and the proper CC bit is 0, and location $FFF8-$FFF9 is $A101, the microprocessor would then start executing code at $A101. Interrupts save the listed registers before the interrupt handler is called, and these registers are restored when the Return From Interrupt (RTI) instruction is called at the end of the interrupt routine.

RTI is similar to RTS except that it, in conjunction with the E bit in CC, determines how many registers to pull from the stack.

To disable the interrupts (useful before many changes in the system), use

```
    ORCC    #$10    disables IRQ
    ORCC    #$40    disables FIRQ
    ORCC    #$50    disables them both
```

To enable the interrupts, use

```
    ANDCC   #$EF    enables IRQ
    ANDCC   #$BF    enables FIRQ
    ANDCC   #$AF    enables them both
```

CoCo 3

The GIME chip has the capability of sending interrupts to either the IRQ or FIRQ line. If you are running a 100% ML program you can them as you want. If you are running a combination program, Basic sets the GIME interrupt registers back to Vertical Border only.

The GIME chip enables a host of other interrupts possible to trigger either/both FIRQ and IRQ interrupts through $FF92-$FF93. The interrupt sources include

- A programmable timer
- Horizontal and vertical border
- Serial port
- Keyboard port and joystick buttons
- Pin 8 of the Cartridge port

TODO

Sound

Mute sound:

```
BEGIN   LDA  $FF23  Get current Control Register B value of PIA 2
        ORA  #$30   Set CB2 to be an output. (Set bits 4 and 5.)
```

Now the status of bit 3 of Control Register B will control the CB2 line. If bit 3 is low the line will be low. If bit 3 is high the line will be high. Setting CB2 low will mute the CoCo.

```
        ANDA  #$F7   Clear bit 3 - Mute CoCo
        STA   $FF23  Write value back to Control Register B
```

In general programming sound uses the 6-bit D/A.

Also, there was a magazine article early on about 4-channel sound, but I have been unable to find it and analyze it for this section. Perhaps Rainbow or Hot-CoCo? I think it is named Bells and Whistles 2. TODO

Sockmaster has a MOD Player.

Another source is the single bit sound:
$FF23 bit 2 to 0, (changes $FF22 to data dir register)
$FF22 to output?,
$FF23 bit back to 1 (change $FF22 back)
Store sound bits into $FF22 (top bit?)

$FF03 bit 3	$FF01 Bit 3	Sound Source
0	0	DAC
0	1	Cassette
1	0	Cartridge
1	1	No Sound

Another source is the cassette recorder

Another source is the cartridge slot?

TODO

Cassette Storage

File format

Color BASIC saves a file as a series of blocks, each with 0-255 bytes of data. Some blocks need preceded by a leader to establish timing.

Each bit is recorded as a single cycle of a sine-wave. A "1" is a single cycle at 2400 Hz, and a "0" is a single cycle at 1200 Hz. Bytes are stored least significant bit first. Bits are recognized when the sine wave crosses from positive to negative, so loudness is not as important as one might expect.

A file consists of:

1. a leader
2. a filename block
3. a 1/2 second gap
4. another leader
5. some number of data blocks
6. an end-of-file block

A leader is just hex $80(128 dec) bytes of hex $55 (binary 01010101).

A block contains:

1. two "magic" bytes ($55 and $3C)
2. one byte - block type (00=filename, $01=data, $FF=EOF)
3. one byte - data length ($00 to $FF)
4. 0 to 255 bytes - data
5. one byte - checksum (sum of data, type, and length bytes)
6. another magic byte ($55)

Filename blocks have $F(15) bytes of data; EOF blocks have zero bytes of data; data blocks have $00-$FF bytes of data indicated by length byte.

A filename block contains:

1. eight bytes - the filename
2. one byte - file type ($00=BASIC, $01=data, $02=machine code)
3. one byte - ASCII flag ($00=binary, $FF=ASCII)
4. one byte - gap flag ($00=no gaps, $FF=gaps)
   (The tech manual incorrectly (?) shows 01 as the code for "no gaps")
5. two bytes - machine code starting address
6. two bytes - machine code loading address

There should be no gaps, except preceding the file, and in case the filename blocks requests gaps, in which case there is a 1/2 second gap and leader before each data block and EOF block.

Hardware

The cassette cable has a 5-pin DIN connector on one end, that plugs into the back of the CoCo; the other end has three earphone-style plugs, that plug into the EAR, AUX (or MIC), and REMOTE jacks. The remote-control plug is smaller than the other two. The other two are differentiated by color: the black one plugs into the EAR jack, while the grey one plugs into AUX.

Here is an ASCII drawing of that connector, including a pinout and showing how the pins are numbered. The drawing is of the connector at the end of the cable, with the pins pointing toward you. So if you are looking at the back of the machine, at the connector there, this pinout is backwards. My apologies for the wacky numbering; this is the same numbering as in the CoCo-1 technical manual.

```
       -------       Pin#  Name      Connects to
     /  \___/  \     ----  -------   -----------------------------------
    /           \
   /             \    1    CASSMOT   SG stem

  |               |   2    GND       B stem, LG stem
  |  1o       o3  |
  |               |   3    CASSMOT   SG tip
       o     o
   \  4   o   5  /    4    CASSIN    B tip
    \     2     /
     \         /      5    CASSOUT   LG tip
       -------

                         B=black    SG=small grey    LG=large grey
```

The names are given from the perspective of the computer, so "OUT" means output from the computer, input to the cassette, and it should go into the AUX (or MIC) jack while the cassette is recording.

The "connects-to" column show which of the three earphone-style plugs the wire leads to, and to which part (stem or tip).

Below is another cheesy ASCII drawing, this time of one of those plugs. The cylindrical prong actually consists of two metal parts, separated by a narrow strip of plastic, drawn as "X"s. The stem section is much longer than the tip section, and the tip section has a groove around it, so that a spring loaded contact can hold it in place a little bit when it is inserted into a jack.

```
-------------
|           |
|           |------------xx-\  /--\
|           |            XX  --    |
|           |  "stem"    XX   "tip"|
|           |            XX  --    |
|           |------------xx-/  \--/
|           |
-------------
```

For both of the plugs that carry data (the large ones), the stem is connected to ground, and the tip is the data line.

For the motor-control plug (the small plug), it shouldn't really matter which of its two wires connects to its tip and which connects to its stem; you could just as easily connect 1 to SG tip and 3 to SG stem, and it should still work fine. These two wires are just connected together by a relay in the computer when it is time to let the motor run. The wiring shown here matches my cassette cable.

TODO

Disk Storage

Disk Format

Low-level format

CoCo disks are formatted to contain 35 tracks, numbered 0 through 34. Each track has 18 sectors, numbered 1 through 18. A sector contains 256 bytes.

High-level format

Track number 17 is special; it contains the directory and File Allocation Table (or FAT). Every other track is divided into two granules; in those tracks, sectors 1 through 9 form one granule, and sectors 10 through 18 form the other. So there are 68 granules on a disk, numbered 0 through 67, each containing 2304 bytes. Disk space for files is allocated by the granule, so even if you create a file that contains only one byte, a whole granule of 2304 bytes is reserved for it. While it may seem wasteful at first, this reduces the amount of work in allocating space for the file as you add to it. The computer only has to do that allocation work once for every 2304 bytes that you add. It also reduces fragmentation - by reserving space in such big chunks, your file can't possibly end up scattered all over the disk in little tiny pieces.

The directory track (17) contains the file allocation table in sector 2, and the directory of files in sectors 3 through 11. The remaining sectors on the directory track are unused ("reserved for future use").

The file allocation table is 68 bytes long; one byte for each granule on the disk. If one of these bytes is between 0 and 67, it tells the number of the next granule used by the same file. If it is between 192 and 201 (hex C0 and C9), then this is the last granule allocated for its file, and the least significant four bits tell how many sectors of the granule are used. If it is FF then it is unused, and may be allocated as needed. So the bytes in the FAT form a linked list for each file, telling which granules the file consists of.

Each directory sector contains eight entries of 32 bytes each. So the entire directory has room for 72 files. (There is room in the directory for more files than there are granules on the disk!) Each entry contains:

* eight bytes for the filename (padded with spaces)
* three bytes for the filename extension (padded with spaces)
* one file-type byte

(0=BASIC program, 1=BASIC data, 2=machine code, or 3=ASCII text)
* one format byte (0=binary or FF=ASCII)
* one byte telling the number of the file's first granule
* two bytes telling the number of bytes used in the last sector in the last granule,
* sixteen unused bytes ("reserved for future use" again).

Color Disk BASIC reserves track 17 for the directory because that is the middle position for the read/write head of the disk drive, so it should be efficient for frequent access. When allocating granules to be used in files, it chooses granules that are close to the directory first, so in a half-full disk you would expect the outermost and innermost tracks to be empty, and the tracks near the directory to be full.

Color BASIC Disk Format:
35 decimal tracks, numbered 0-34.
18 sectors, numbered 1-18.
Each sector has 256 bytes.
Total size then 35*18*256 = 161280 decimal bytes.

High-level format
Track 17 contains the directory and File Allocation Table (FAT). Other tracks split the eighteen sectors into two granules: sectors 1-9 make one granule, 10-18 make the other. The granules are then numbered 0-67, each containing 2304 bytes. Files are allocated at the granule level, so a one-byte file still reserves 2304 bytes. Track 17 is the middle of the disk, so is in a good position for disk activity.

Track 17 contains the FAT in sector 2, and the directory on sectors 3 though 11. Other sectors are unused.

The bytes in the FAT contain linked lists of file locations on the disk.

The FAT is 68 bytes long - one byte for each granule on the disk. Values 0-67 denote the NEXT granule used by the file. Values between $C0(192) and $C9(201) denote the last granule for the file, and the least four significant bits tell how many sectors of the granule are used. Value $FF marks an unused granule.

A directory sector contains eight entries of $20(32) bytes, making room for seventy-two files. A directory is:

1. eight bytes for the space-padded filename
2. three bytes for the space padded filename extension
3. one file-type byte ($0=BASIC program, $1=BASIC data, $2=machine code, $3=ASCII text)
4. one format byte ($0=binary or $FF=ASCII)
5. one byte containing the file's first granule
6. two bytes containing the number of bytes used in the last sector of the last granule
7. sixteen unused bytes

Controller

The disk controller consists of a ROM that adds disk commands to Extended Color BASIC, a disk controller chip, and a little glue to make it all work.

The disk controller chip is a Western Digital 1793 (or 1773), and has four registers at addresses $FF48 through $FF4B, and one control register at $FF40. The control register enables the drive motors, select lines, and so on.

Here is a map:

- $FF40 Control register
- $FF48 Command/Status register
- $FF49 Track register
- $FF4A Sector register
- $FF4B Data register

In general, you make the disk controller execute a command by writing a command byte into the Command register, and see the results by reading the Status register. During the execution of a read command, you have to load data bytes from the data register, where they appear as they come from the disk; during a write, you have to put bytes into the data register from where they will be written onto the disk. The speed of reads and writes is constant; if your program does not read or write bytes into the Data register quickly enough, the chance is gone, and the command will not complete successfully.

Control Register
This is not part of the controller chip, but is part of the "glue" that makes it all fit together.

You can force the drive motor to turn off by writing a zero into this. You can also force the motor on by writing non-zero there, but the documentation doesn't explain which bits do what. Don't panic; more details are on the way.

This register is write-only; you can't tell what value was last written there just by reading it. So Disk BASIC probably keeps a copy in somewhere of each value that it writes here. This means that if your programs write here, it might confuse BASIC, because the last thing it put here is no longer true. Be careful about that.

Since this register is not part of the disk controller chip, there are other things to be careful of as well. For instance, the disk controller's Track register normally contains the track number at which the disk drive head is positioned. But if you have more than one disk drive, you can switch which one you are using by writing some value into this Control register. The disk controller does not know that it is now talking to a new disk drive, and the head position of the new disk drive may be different from the old one. So the controller might think it is at track 10, where the old drive was. But maybe the new drive's head is over track 23. As soon as the controller tries to execute any read or write command, it will notice that the data coming from the disk drive is claiming track 23, instead of the 10 it expected. So the controller will return an error instead of executing the command.

Track and Sector registers

The Track and Sector registers just hold track and sector numbers. The Track register normally reflects the current position of the head; you don't normally write into it. To get to a track you use the Seek command and put the desired track number in the data register. When that command has completed successfully, the track number will be in the Track register. In contrast, the Sector register is used to tell the controller which sector you want; you write into it.

You can, of course, write into the Track register. But if it doesn't match the position of the head, the controller will produce an error if you try to read or write a sector - it notices that it is not at the track it expected.

Still, there is a good reason to write into the Track register if you have more than one disk drive. Whenever you select a new disk drive by writing into the Control register, you may want to update the controller's Track register with the position of the new drive.

Data Register

Command/Status Register

Writing into this register gives a command to the disk controller chip. Reading from it tells you the status of the command's execution. In effect, the two registers share the same address; the Command register is write-only, and the Status register is read-only.

Command bits

There are four types of disk commands.

Type I - Restore, Seek, Step, Step In, and Step Out.
Type II - Read Sector and Write Sector.
Type III - Read Track, Write Track, and Read Address.
Type IV - Force Interrupt.

Status bits

Bits in the error code are defined as follows. (This info comes from the 1793 data sheet.) The 1793 data sheet is more terse than the 1771 data sheet which was mistakenly referenced before, so I've filled in some missing descriptions in italics, like this; some from the 1771 data sheet and others that just seem obvious. Take those additions with a grain of salt, although I *think* they are right.

Note that some of the bits have different meanings based on which type of command caused them to be set. The status word is defined only for commands of type I, II, and III. The status of a type IV command, Force Interrupt, depends on the command that was interrupted.

Signals are named from the perspective of the 1793 disk controller chip, so "input" means input to that chip from either the computer or the disk drive, and "output" means an output from the 1793 to one of those. Signal names preceded by an asterisk "*" indicate that the signal is active-low, or inverted, so that "0" means true and "1" means false.

7 Not Ready

This bit, when set, indicates that the disk drive is not ready. When reset it indicates that the drive is ready. This bit is an inverted copy of the READY input from the disk drive and logically ORed with the *MR (Master Reset) input signal. Type II and III commands will not execute unless the drive is ready.

6 Write Protect

Type I commands:

When set, indicates that Write Protect is activated. This bit is an inverted copy of the *WRPT (Write Protect) input from the disk drive.

Type II/III commands:

On Read Sector: not used. On Read Track: not used. On any Write command, this bit indicates that the diskette was write protected so the write failed. This bit is reset when updated.

5 Head Loaded/Record Type/Write Fault

Type I commands: Head Loaded

This indicates that the head is loaded and engaged, and is a logical AND of the HLD (Head Loaded) and HLT (Head Load Timing) input signals from the disk drive.

Type II/III commands: Record Type/Write Fault

On Read Sector: it indicates the record-type code from the data-field address mark. 0 = Data Mark, 1 = Deleted Data Mark. On any Write: It indicates a write fault. This bit is reset when updated.

4 Seek Error/Record Not Found

Type I commands: Seek Error

When set, the desired track was not verified. This bit is reset to 0 when updated.

Type II/III commands: Record Not Found

When set, it indicates that the desired track, sector, or side were not found. This bit is reset when updated.

3 CRC error

Type I commands:

CRC error encountered in ID field during track verification.

Type II/III commands:

If Bit 4 is set, an error was found in one or more ID fields; otherwise, it indicates an error in data field. (Each sector is written as an ID field followed by a data field; each field contains a CRC, which is a kind of checksum used as a sanity-check when reading.) This bit is reset when updated.

2 Track 00/Lost Data

Type I commands: Track 00

When set, indicates Read/Write head is positioned to Track 0. This bit is an inverted copy of the *TR00 (Track 00) input from the disk drive.

Type II/III commands: Lost Data

When set, it indicates that the computer did not respond to the DRQ (Data Request) output in one byte time and therefore that data was lost. This bit is reset to zero when updated.

1 Index/Data Request
Type I commands: Index
When set, indicates index mark detected from drive. This bit is an inverted copy of the *IP (Index Pulse) input signal from the disk drive.
Type II/III commands: Data Request
This bit is a copy of the DRQ output. When set, it indicates that the DR (Data Register) is full on a Read operation or the DR is empty on a Write operation. This bit is reset to zero when updated.

0 Busy
When set, command is under execution. When reset, no command is under execution.

DUP - merge and remove

TODO - clean up, unify with hardware reference

In short:
$FF40 Control register
$FF48 Command/Status register
$FF49 Track register
$FF4A Sector register
$FF4B Data register

Write a command into the command register, and read the status in the status register. For reads and writes you need to read/write data to/from the data register. You must do this at the proper speed or the command will fail.

Writing a 0 into the control register turns off the drive motor.

The control register is write only, so Disk Basic keeps a copy of what is written there. If you modify it, you should keep this in mind.

The Track and Sector registers hold current track and sector numbers, reflecting register the current position of the head. Use the Seek command to position the head to the Track you want. Then write the Sector register to tell the controller which sector you want.

Command/Status
Writing into register $FF48 gives a command to the disk controller chip. Reading from it tells you the status of the command's execution.

There are four types of disk commands.
Type I - Restore, Seek, Step, Step In, and Step Out.
Type II - Read Sector and Write Sector.
Type III - Read Track, Write Track, and Read Address.
Type IV - Force Interrupt.

Status bits in the error code are defined as follows, from the 1793 data sheet, and have meaning dependent on the command type. Type IV status codes depend on what command was interrupted.

Bit 7 - Not Ready
  0 - drive ready
  1 - drive not ready.
  Type II and III will not execute unless the drive is ready.

Bit 6 - Write Protect
  Type I   : 0 - not write protected, 1 - write protected;
  Type II/III : Not used on Read Sector or Track. On Write, same as Type I.
    This bit is reset when updated.

Bit 5 - Head Loaded/Record Type/Write Fault
  Type I commands: Head Loaded 1 - head loaded and engaged
  Type II/III commands: Record Type/Write Fault
    Read : indicates the record-type code from the data-field address
      mark. 0 = Data Mark, 1 = Deleted Data Mark.
    Write: indicates a write fault. This bit is reset when updated.

Bit 4 Seek Error/Record Not Found
  Type I : Seek Error - 0 = verified, 1 = track not verified. Reset to 0
    when updated.
  Type II/III : Record Not Found - 0 - ok, 1 - track, sector, or side not
    found. Reset when updated

Bit 3 CRC error (Cyclic Redundancy Check)
  Type I commands: 0 - CRC ok, 1 - CRC failed
  Type II/III commands: If bit 4 set, indicates an error in 1+ ID fields,
    else error in Data field This bit is reset when updated.

Bit 2 Track 00/Lost Data
  Type I commands: Track 00 - 0 = ?, 1 = Read/Write head positioned at
    Track 0.
  Type II/III commands: Lost Data 1 - Computer did not respond to DRQ
    (Data Request) in time and lost data. Bit reset to 0 on update.

Bit 1 Index/Data Request
  Type I commands: Index - 0 - ?, 1 - index mark detected from drive.
  Type II/III commands: Data Request., copy of DRQ output. 1 - DR(Data
    Register) is full on a read or empty on write, reset to 0 when
    updated.

Bit 0 Busy

0 - not busy
1 - Command being processed

TODO

Serial I/O

Software

The 4-pin DIN connector on the CoCo back is a serial port. This must be operated from software; a loop reads and writes bits to this port as needed.

Set baud rate (values in decimal):

```
    POKE 150,180    [300 bps]
    POKE 150,88     [600 bps]
    POKE 150,41     [1200 bps]
    POKE 150,18     [2400 bps]
    POKE 150,7      [4800 bps]
    POKE 150,1      [9600 bps]
```

Others have used assembly routines to support much faster rates.

Hardware

The Color Computer has a four-pin DIN connector on its back panel for its serial port. There is very little internal hardware dedicated to supporting this, so most of the work of sending and receiving bits is done in software; the CPU goes into a loop either setting the output bit or reading the input bit.

As you might imagine, doing both at the same time can be tricky, since you don't have any guarantees about when the first bit of a byte will arrive; it might be while you are right in the middle of sending a byte. The result is a limited baud rate; the CPU can only do so many bits per second.

Here is a drawing of that connector, including a pinout and showing how the pins are numbered. The drawing is of the back-panel connector, looking at it from the back of the machine. So if you are looking at the pins of the connector at the end of a cable, it is backwards.

```
       ------        Pin#  RS-232 signal           Printer signal
     /  \__/  \      ----  ---------------------   ---------------
    /          \
   / 4      1   \     1    CD - carrier detect     ignored
      o      o
  |              |    2    RS232IN - input data    printer status
  |              |                                 (high == ready)
  |  3      2    |
      o      o        3    GND - ground            ground
   \            /
    \          /      4    RS232OUT - output data  data to printer
     \        /
       ------
```

Note that pins of connector are not always used the same way! This is possible because most everything is done in software; pins 1 and 2 are required by the hardware to be inputs, but it is up to the program to decide how to use those inputs. While any sort of communications program should use the RS-232 pin-out, the built-in BASIC printer routines use pin 2 as "printer status" and completely ignore pin 1. So the cable you wire up for a printer has to be different from the one you wire up for a modem.

TODO

Cartridge Info

Color Computer 1, 2, & 3 Cartridge Connector Definitions (* are LOW (0 volts) to activate)

Pin	Signal Name	Description
1	N.C.	(-12 VDC on CoCo 1 and 2)
2	N.C.	(+12 VDC on CoCo 1 and 2)
3	HALT*	Halt input to the CPU
4	NMI*	Non-Maskable Interrupt to the CPU
5	RESET*	Main Reset and Power-up Clear
6	E CLOCK	Main CPU Clock
7	Q CLOCK	Clock which leads E by 90 degrees
8	CART*	Rom-Pak Detection Interrupt
9	+5 VDC	+5 Volts DC (300 mA)
10	DATA 0	CPU Data Bus - Bit 0
11	DATA 1	CPU Data Bus - Bit 1
12	DATA 2	CPU Data Bus - Bit 2
13	DATA 3	CPU Data Bus - Bit 3
14	DATA 4	CPU Data Bus - Bit 4
15	DATA 5	CPU Data Bus - Bit 5
16	DATA 6	CPU Data Bus - Bit 6
17	DATA 7	CPU Data Bus - Bit 7
18	R/W*	CPU Read/Write Signa
19	ADDR 0	CPU Address Bus - Bit 0
20	ADDR 1	CPU Address Bus - Bit 1
21	ADDR 2	CPU Address Bus - Bit 2
22	ADDR 3	CPU Address Bus - Bit 3
23	ADDR 4	CPU Address Bus - Bit 4
24	ADDR 5	CPU Address Bus - Bit 5
25	ADDR 6	CPU Address Bus - Bit 6

26	ADDR 7	CPU Address Bus - Bit 7
27	ADDR 8	CPU Address Bus - Bit 8
28	ADDR 9	CPU Address Bus - Bit 9
29	ADDR 10	CPU Address Bus - Bit 10
30	ADDR 11	CPU Address Bus - Bit 11
31	ADDR 12	CPU Address Bus - Bit 12
32	CTS*	Cartridge (ROM) Select Signal
33	GROUND	Signal Ground
34	GROUND	Signal Ground
35	SND	Cartridge Sound Input
36	SCS*	Spare Cartridge (DISK) Select Signal
37	ADDR 13	CPU Address Bus - Bit 13
38	ADDR 14	CPU Address Bus - Bit 14
39	ADDR 15	CPU Address Bus - Bit 15
40	SLENB*	Input to Disable Internal Devices

By covering pin 8 on the cartridge, ROM-packs could be inserted without them starting up. It is EXTREMELY DANGEROUS to insert a ROM-Pack with the CoCo switched on. You might cook your CoCo.

Some signals:

HALT*	This signal allows the data and address buses to be placed in the tri-state mode so an external processor may access RAM and ROM
NMI*	This is the non-maskable interrupt input to the CPU.
RESET*	This is master system reset and power-up clear signal.
E & Q	These are the two clock signals for the 6809E CPU
CART*	This is an interrupt input to one of the PIA'S. It is used to detect the presence of a Cartridge
CTS*	This is the select signal to the Cartridge. The address space C000 (Hex) through FFEF (Hex) is selected.
SND	This signal is connected directly to the sound multiplexer, to allow a sound source in the cartridge.
SCS*	This is a spare divide select signal from U11. It selects the address space FF40 (Hex) through FF5F (Hex).
SLENB*	This signal disables the internal device selection. This allows decoded but unused sections of memory to be used by the Cartridge hardware.

TODO – boot sequence? Addresses used?

Basic, Extended Basic, and Disk Basic Summary

TODO – shrink, more comments, 4 column wide table?

Color Basic (non-Extended) Summary

Statements

AUDIO ON	
AUDIO OFF	
CLEAR n,h	Reserve n bytes for strings, and use only up to address h for BASIC
CLOAD	
CLOAD name	
CLOSE d	
CLS c	
CONT	
CSAVE name	
CSAVE name,A	
DATA	
DIM	
END	
EXEC	
EXEC address	
FOR .. TO .. STEP .. / NEXT	
GOSUB linenumber	
GOTO linenumber	
IF .. THEN .. ELSE ..	
INPUT	
INPUT #-1	
LIST	
LLIST	
MOTOR ON	
MOTOR OFF	
NEW	
ON .. GOSUB ..	
ON .. GOTO ..	
OPEN m,#d,filename	
POKE addr,value	Save value at address addr, where 0 <= addr <= 65535, and 0 <= value <= 255
PRINT	
PRINT #-1	
PRINT #-2	
PRINT TAB	
PRINT @location	
READ	
REM	
RESET (x,y)	
RESTORE	
RETURN	

RUN
SET (x,y,c)
SKIPF
SKIPF name
SOUND tone,duration
STOP

Functions
ABS(num)
ASC(str)
CHR$(charcode)
EOF(f)
INKEY$
INSTR(first,str,substr)
INT(num)
JOYSTK(j) Reads joystick value j: 0=left_horiz 1=left_vert 2=right_horiz 3=right_vert
LEFT$(str,length)
LEN(str)
MEM
MID$(str,first,len)
POINT(x,y)
RIGHT$(str,length)
SGN(num)
SIN(num)
STR$(num)
USR(num) Calls the machine-language routine whose address is stored at addresses 275 and 276
VAL(str)
VARPTR(var)

Operators
^ Exponentiation
-,+ Unary negative, positive
*,/ Multiplication, division
+,- Addition and concatenation, subtraction
<,>,=,<=,>=,<> Relational tests
NOT, AND, OR Logical operators

Error messages
Abbrev. Explanation
/0 Division by zero
AO File already open
BS Bad subscript - out of range
CN Cannot continue
DD Redimensioned array
DN Device number error
DS Direct statement in file

FC Illegal function call
FD Bad file data
FM Bad file mode
ID Illegal direct
IE Input past end of file
I/O Input/Output error
LS String too long
NF NEXT without FOR
NO File not open
OD Out of data
OM Out of memory
OS Out of string space
OV Overflow
RG RETURN without GOSUB
SN Syntax error
ST String formula too complex
TM Type mismatch
UL Undefined line number

Extended Color Basic Summary

Statements
AUDIO ON
AUDIO OFF
CIRCLE(x,y),r,c,hw,start,end
CLEAR n,h Reserve n bytes for strings, and use only up to address h for BASIC
CLOAD
CLOAD name
CLOADM
CLOADM name
CLOADM name,offset
CLOSE d
CLS c
COLOR (fg,bg)
CONT
CSAVE name
CSAVE name,A
CSAVEM name,a1,a2,ax
DATA
DEF FN
DEFUSERn = addr
DEL
DIM
DLOAD

DRAW string
EDIT linenumber
END
EXEC
EXEC address
FOR .. TO .. STEP / NEXT
GET (start)-(end),dest,G
GOSUB linenumber
GOTO linenumber
IF .. THEN .. ELSE
INPUT
INPUT #-1
LET
LIST
LLIST
LINE (x1,y1)-(x2,y2),PSET,BF
LINE (x1,y1)-(x2,y2),PRESET,BF
LINE INPUT
MOTOR ON
MOTOR OFF
NEW
ON .. GOSUB
ON .. GOTO
OPEN m,#d,filename
PAINT (x,y),c,b
PCLEAR n
PCLS c
PCOPY
PLAY string
PMODE mode,startpage
POKE addr,value   Save value at address addr, where 0 <= addr <= 65535, and 0 <= value <= 255
PRESET (x,y)
PRINT
PRINT #-1
PRINT #-2
PRINT TAB
PRINT USING
PRINT @location
PSET (x,y,c)
PUT (start)-(end),source,action
READ
REM
RENUM newline,startline,increment
RESET (x,y)
RESTORE
RETURN

RUN
SCREEN screentype,colorset
SET (x,y,c)
SKIPF
SKIPF name
SOUND tone,duration
STOP
TROFF
TRON

Functions

ABS(num)
ASC(str)
ATN(num)
CHR$(charcode)
COS(num)
EOF(f)
EXP(num)
FIX(num)
HEX$(num)
INKEY$
INSTR(first,str,substr)
INT(num)
JOYSTK(j) Reads joystick value j: 0=left_horiz 1=left_vert 2=right_horiz 3=right_vert
LEFT$(str,length)
LEN(str)
LOG(num)
MEM
MID$(str,first,len)
PEEK(address)
POINT(x,y)
POS(dev)
PPOINT(x,y)
RIGHT$(str,length)
SGN(num)
SIN(num)
STRING$(length,charcode)
STRING$(length,str)
STR$(num)
SQR(num)
TAN(num)
TIMER
USRn(num) Calls the machine-language subroutine whose address was defined by DEFUSRn, where 0 <= n <= 9

VAL(str)
VARPTR(var)

Operators

^	Exponentiation
-,+	Unary negative, positive
*,/	Multiplication, division
+,-	Addition and concatenation, subtraction
<,>,=,<=,>=,<>	Relational tests
NOT, AND, OR	Logical operators

Error messages

Abbrev.	Explanation
/0	Division by zero
AO	File already open
BS	Bad subscript - out of range
CN	Cannot continue
DD	Redimensioned array
DN	Device number error
DS	Direct statement in file
FC	Illegal function call
FD	Bad file data
FM	Bad file mode
ID	Illegal direct
IE	Input past end of file
I/O	Input/Output error
LS	String too long
NF	NEXT without FOR
NO	File not open
OD	Out of data
OM	Out of memory
OS	Out of string space
OV	Overflow
RG	RETURN without GOSUB
SN	Syntax error
ST	String formula too complex
TM	Type mismatch
UL	Undefined line number

Disk Basic Summary

In addition to the capabilities of Extended Color BASIC, Color Disk BASIC adds the following.

Disk management commands

BACKUP n TO m	Copy all files from one disk to another

```
  BACKUP n               BACKUP a disk using only a single disk drive
  COPY file1 TO file2    Make a duplicate of a file
  DIR n                  List the files that are on the disk
  DRIVE n                Use drive n as the default
  DSKINIn                Initialize (format) a disk
  KILL file              Delete a file from the disk
  LOAD file              Load a program
  LOAD file,R            Load a program and start it running
  LOADM file,offset      Load a machine-code program, shifting by offset
  MERGE file             Load an ASCII program without clearing the old one
  MERGE file,R           Merge a program and start it running
  RENAME file1 TO file2  Change the name of a file
  RUN file               Load a program and start it running
  RUN file,R             Load and run program, leaving files open
  SAVE file              Save a program
  SAVE file,A            Save a program in ASCII format
  SAVEM file,a1,a2,ax    Save a machine-code program, from a1 to a2, exec at ax
  VERIFY ON              Double-check all writes to the disk
  VERIFY OFF             Don't double-check

Programming commands
  FILES max_f,size       Reserve buffers for open files
  FREE(n)                Returns the number of free granules (2304 bytes each)
  UNLOAD n               Close all open files on drive n
  DSKI$ n,t,s,v1$,v2$    Read track t sector s into v1$ and v2$
  DSKO$ n,t,s,v1$,v2$    Write track t sector s from v1$ and v2$
  OPEN "I",f,file        Open a file for sequential input (ie: INPUT)
  OPEN "O",f,file        Open a file for sequential output (ie: PRINT/WRITE)
  OPEN "D",f,file,len    Open a file for direct access; (ie: GET/PUT); record length len is optional
  CLOSE #f               Close a file

Sequential file commands
  EOF(f)                 Returns true if file f has been read to the end
  INPUT #f, var,...      Read variables from a file
  LINE INPUT #f,var$     Read an entire line from a file into a string variable
  WRITE #f,values        Write values to file, with commas, strings in quotes,...
  PRINT #f,values        Write values to file, just as PRINT would display them
  PRINT #f,USING f$;values Formatted printing; many options for f$

Direct-access file commands
  FIELD #f, size AS v$,... Give variable names to parts of the file buffer
  RSET v$ = value$       Fill in a named part of the file buffer, right-justified
  LSET v$ = value$       Fill in ..., left justified
  PUT #f,r               Write the buffer to record r
  GET #f,r               Read record r into the buffer
  CVN(var$)              Make a number out of a binary string
```

MKN$(num)	Make a binary string out of a number
LOC(f)	Return the current record number in the buffer
LOF(f)	Return the highest record number in the file

In all cases, f is a file number, n and m are drive numbers, file is a filename, and dollar signs signify variables that must be string-variables. Note that filenames must be either string variables or string constants in quotes. Upper-case words are keywords, lower-case words are supplied by the user.

Special file numbers are -2=printer -1=cassette and 0=screen

The name can be up to eight characters long, and cannot include a dot, slash, colon, or zero. The extension can be up to three characters long, and also cannot include those four characters. The drive number is a single digit, from zero up to the highest drive on your system.

Examples of legal filenames:

```
PROGRAM.BAS  -- filename=PROGRAM,  extension=BAS, no drivenum
PROGRAM/BAS  -- filename=PROGRAM,  extension=BAS, no drivenum
FOO.BAR:0    -- filename=FOO,      extension=BAR, drivenum=0
FOO:1        -- filename=FOO,      no extension,  drivenum=1
FRED         -- filename=FRED      no extension,  no drivenum
```

There is one documented subroutine in the Disk BASIC ROM that you can use to access the disk. Its address is stored at $C004 and $C005, so you jump to it using indirection: JSR [$C004] .

Before calling that, you should load the X register with the address of a data structure that describes what you want to do. The examples in the manual always load this address from locations $C006 and $C007. I have not tried using this, so I don't know it will work if you put your structure anyplace else. This data structure is seven bytes long:

1 byte	op code (0 - 3)
1 byte	drive number (0 - 3)
1 byte	track number (0 - 34)
1 byte	sector number (1 - 18)
2 bytes	address of 128-byte data buffer
1 byte	error code

Op codes are either 0 (restore to track 0), 1 (no op), 2 (read sector), or 3 (write sector).

Bits in the error code seem to come straight from the chip in the disk controller. See that for more details.

The disk control routine modifies the contents of only the condition code register.

ROM Routines

Color Basic Info

To detect which version of the Color BASIC ROM you have between 1.0, 1.1, and 1.2, check location $A155, which holds $30, $31, and $32 respectively.

Extended Color Basic Info

To detect which version of the Extended Color BASIC ROM you have between 1.0 and 1.1, check location $80FF, which holds $30 and $31 respectively.

Disk Color Basic Info

TODO – get info from unraveled series. Also get disk ROM versions and how to detect.

Rom Routines

Here are the approved routines that could be called from assembly language.

ROM subroutines

BLKIN [$A006]
Reads a Block from Cassette
Must immediately follow CSRDON.
CSRDON, CBUFAD contains the Buffer address.
BLKTYP, located at 7C, contains the block type:
0 = File Header
1 = Data
FF = End of File
BLKLEN, located at 7D, contains the number of data bytes in the block (0-255).
Z = 1, A=CSRERR=0 (if NO Errors.)
Z = 0, A=CSRERR=1 (if a checksum error occurs)
Z = 0, A=CSRERR=2 (if a memory error occurs)
(Z is a flag in the Condition Code CC Register)
(CSRERR = 81)
Unless a memory error occurs, X=CBUFAD + BLKLEN. If a memory error occurs, X points to beyond the bad address. Interrupts are masked. U and Y are preserved, all other registers are modified.

BLKOUT [$A008]
Writes a Block to Cassette
Call Subroutine WRTLDR

CBUFAD, located at 7E, contains the buffer address
BLKTYP, located at 7C, contains the block type
BLKLEN, located at 7D, contains the number of data bytes
Interrupts are masked.
X = CBUFAD + BLKLEN.
All Registers are Modified

WRTLDR [$A00C]
Turns the cassette On and writes a Leader
Entry: None
Return: None

CHROUT [$A002]
Outputs a Character to Device
On Entry, the character to be output is in A
Output device is determined by the contents of 6F (DEVNUM) (0 = Screen, -2 = Printer)
All registers except CC are preserved

CSRDON [$A004]
starts the cassette and gets into bit sync for reading
Entry: None
FIRQ and IRQ are masked.
U and Y are preserved, all others are modified

GIVABF [$B4F4]
Passes parameter to BASIC
D = parameter
USR variable = parameter

INTCNV [$B3ED]
Passes parameter from BASIC
USR argument = parameter
D = parameter

JOYIN [$A00A]
Samples all 4 joystick pots. Values are stored in POTVAL through POTVAL + 3
Left Joystick Up / Down $15A Right / Left $15B
Right Joystick Up / Down $15C Right / Left $15D
Entry conditions: None
Registers used: Y is preserved. All others are modified

POLCAT [$A000]
Polls Keyboard for a character
None
Return:
If Key is pressed

Z = 0 and A register contains ASCII value
If no key is pressed
Z = 1 and A register contains 00
Registers: A and CC are modified

There are many more that can be called depending on the ROM installed, and these must be used very carefully.

Color Computer Hardware Register Reference ($FF00-$FFFF)

Here is the usage of the hardware registers $FF00-$FFFF.

PIA Reference ($FF00-$FF3F)

For PIA details see the section on the PIA.

PIA0 ($FF00-$FF1F)

$FF00 (65280)	PIA 0 side A data register - PIA0AD	CoCo 1/2/3
Bit 7	Joystick Comparison Input	
Bit 6	Keyboard Row 7	
Bit 5	Row 6	
Bit 4	Row 5	
Bit 3	Row 4 & Left Joystick Switch 2	
Bit 2	Row 3 & Right Joystick Switch 2	
Bit 1	Row 2 & Left Joystick Switch 1	
Bit 0	Row 1 & Right Joystick Switch 1	
(1) Todo - keyboard matrix - note		

$FF01 (65281)	PIA 0 side A control reg - PIA0AC	CoCo 1/2/3
Bit 7	HSYNC Flag	
Bit 6	Unused	
Bit 5	1	
Bit 4	1	
Bit 3	Select Line LSB of MUX	
Bit 2	DATA DIRECTION TOGGLE 0 = $FF00 sets data direction 1 = normal	
Bit 1	IRQ POLARITY 0 = flag set on falling edge 1=set on rising edge	
Bit 0	HSYNC IRQ 0 = disabled 1 = enabled	

$FF02 (65282)	PIA 0 side B data register - PIA0BD	CoCo 1/2/3
Bit 7	KEYBOARD COLUMN 8	
Bit 6	7 / RAM SIZE OUTPUT	
Bit 5	6	
Bit 4	5	

Bit 3	4
Bit 2	3
Bit 1	2
Bit 0	KEYBOARD COLUMN 1
|(1) Todo - keyboard matrix - note	

$FF03 (65283)	PIA 0 side B control reg - PIA0BC CoCo 1/2/3
Bit 7	VSYNC FLAG
Bit 6	N/A
Bit 5	1
Bit 4	1
Bit 3	SELECT LINE MSB of MUX
Bit 2	DATA DIRECTION TOGGLE 0 = $FF02 sets data direction 1=normal
Bit 1	IRQ POLARITY 0=flag set on falling edge 1=set on rising edge
Bit 0	VSYNC IRQ 0=disabled 1=enabled

Note: $FF00-$FF03 are repeated through addresses $FF04 to $FF1F. Thus $FF1E is an alias for $FF02. Similarly, $FF20-$FF23 are repeated through $FF24-$FF3F.

PIA1 ($FF20-$FF3F)

$FF20 (65312)	PIA 1 side A data register - PIA1AD CoCo 1/2/3
Bit 7	6 BIT DAC MSB
Bit 6	
Bit 5	
Bit 4	
Bit 3	
Bit 2	6 BIT DAC LSB
Bit 1	RS-232C DATA OUTPUT
Bit 0	CASSETTE DATA INPUT
$FF21 (65313)	**PIA 1 side A control reg - PIA1AC CoCo 1/2/3**
Bit 7	CD FIRQ FLAG
Bit 6	N/A
Bit 5	1
Bit 4	1
Bit 3	CASSETTE MOTOR CONTROL 0=OFF 1=ON
Bit 2	DATA DIRECTION CONTROL 0=$FF20 data direction 1=normal
Bit 1	FIRQ POLARITY 0=falling 1=rising
Bit 0	CD FIRQ (RS-232C) 0=FIRQ disabled 1=enabled

$FF22 (65314)	PIA 1 side B data register - PIA1BD CoCo 1/2/3
Bit 7	VDG CONTROL A/G : Alphanum = 0, graphics =1
Bit 6	" GM2

Bit 5	" GM1 & invert
Bit 4	VDG CONTROL GM0 & shift toggle
Bit 3	RGB Monitor sensing (INPUT) CSS - Color Set Select 0,1
Bit 2	RAM SIZE INPUT
Bit 1	SINGLE BIT SOUND OUTPUT
Bit 0	RS-232C DATA INPUT
(1) VDG sets graphics modes for CoCo 1/2 and CoCo 3 in compatibility mode. To set a mode, use these bits and the registers $FFC0-$FFC5. See the section under $FFC0-$FFC5 for details and text/graphics mode settings.	

$FF23 (65315)	PIA 1 side B control reg - PIA1BC CoCo 1/2/3
Bit 7	CART FIRQ FLAG
Bit 6	N/A
Bit 5	1
Bit 4	1
Bit 3	SOUND ENABLE
Bit 2	DATA DIRECTION CONTROL 0 = $FF22 data direction 1 = normal
Bit 1	FIRQ POLARITY 0 = falling 1 = rising
Bit 0	CART FIRQ 0 = FIRQ disabled 1 = enabled

Note: $FF00-$FF03 are repeated through addresses $FF04 to $FF1F. Thus $FF1E is an alias for $FF02. Similarly, $FF20-$FF23 are repeated through $FF24-$FF3F.

Disk Controller Reference

Disk Controller ($FF40)

$FF40 (65344)	Disk Controller DSKREG CoCo 1/2/3
Bit 7	halt flag 0 = disabled 1 = enabled
Bit 6	drive select 3
Bit 5	density flag 0 = single 1 = double
Bit 4	write precompensation 0 = no precomp 1 = precomp
Bit 3	drive motor enable 0 = motors off 1 = motors on
Bit 2	drive select 2
Bit 1	drive select 1
Bit 0	drive select 0

1. This is a write only register
2. Write precomp should be on for tracks over 22.
3. Disk communication is done through $FF48-$FF4B as follows

Reg	Read operation	Write operation
$FF48	Status	Command
$FF49	Track	Track
$FF4A	Sector	Sector
$FF4B	Data	Data

4. See $FF48 for the list of commands.

DSKREG Copies ($FF41-$FF47) (65345-65351)

$FF41-$FF47 (65345-65351)	DSKREG IMAGES	CoCo 1/2/3
1) Copies of disk registers?		

Status/Command ($FF48)

$FF48 (65352)	Floppy Disk Controller STATUS/COMMAND REGISTER FDCREG	CoCo 1/2/3
Bits 7 - 0	Status/Command register for disk controller	

(1) Write sends a command, then read to get status

COMMANDS	TYPE	COMMAND CODE
RESTORE	I	$03
SEEK	I	$17
STEP	I	$23
STEP IN	I	$43
STEP OUT	I	$53
READ SECTOR	II	$80
WRITE SECTOR	II	$A0
READ ADDRESS	III	$C0
READ TRACK	III	$E4
WRITE TRACK	III	$F4
FORCE INTERRUPT	IV	$D0

(2) Read obtains status resulting from a command. See Status explained elsewhere

(3) Commands

Bit

7 6 5 4 3 2 1 0	Command
0 0 0 0 x x x x	Restore to track 0
0 0 0 1 x x x x	Seek
0 0 1 x x x x x	Step
0 1 0 x x x x x	Step in
0 1 1 x x x x x	Step out

Bits:

4: 0:No update of track reg
   1:Update track register

3: 0:Unload head at start
   1:Load head at start

2: 0:No verify of track no
   1:Verify track no. on disc

1-0:Read as 2-bit stepping rate:
00 = 6ms
01 = 12ms
10 = 20ms
11 = 30ms

1 0 0 x x x x 0 Read sector
1 0 1 x x x x x Write sector
1 1 0 0 0 x x 0 Read address
1 1 1 0 0 x x 0 Read track
1 1 1 1 0 x x 0 Write track

Bits:
4: 0:Read/write 1 sector
1:Read all sectors till the end of a track.
3: Interpretation of 2 bit sector length field in sector header
0: Field is interpreted as
00 = 256 bytes/sector
01 = 512 bytes/sector
10 = 1024 bytes/sector
11 = 128 bytes/sector
1: Field is interpreted as
00 = 128 bytes/sector
01 = 256 bytes/sector
10 = 512 bytes/sector
11 = 1024 bytes/sector (set to 1 on Dragon)
2: 0:No head loading delay
1:Head loading delay of 30ms prior to read/writes.
1: 0:Set side select o/p to 0
1:Set side select o/p to 1
0: 0:Write Data Address Mark
1:Write Deleted Data

Address mark

1 1 0 1 x x x x Force Interrupt
Generate an interrupt & terminate the current operation on:
Bits set:
0 - Drive status transition Not-Ready to Ready
1 - Drive status transition Ready to Not-Ready
2 - Index pulse
3 - Immediate interrupt

Bits clear:
No interrupt occurs, all operations terminated. ($D0)

Status (read), when set: Status bits may have different meanings depending on the command being performed. 0 - Drive busy 1 - Data Request (Data Read/Data Written) OR Index Pulse 2 - Lost Data/Track 00 3 - CRC error 4 - Record Not Found/Seek Err 5 - Data Address Mark 0: Data Address Mark read 1: Deleted Data Address Mark read OR Head Loaded 6 - Write Protect 7 - Not Ready

Track $FF49

$FF49 (65353)	FDC Track Register	CoCo 1/2/3
Bits 7 - 0	Disk Controller Track Register	
(1) Track is 0-34 decimal (2) Do not write directly, but use SEEK command		

Sector $FF4A

$FF4A(65354)	FDC Sector Register	CoCo 1/2/3
Bits 7-0	Disk Controller Sector Register	
(1) Sector is 1-18 decimal (2) Can write directly		

Data $FF4B

$FF4B(65355)	FDC Data Register	CoCo 1/2/3
Bits 7 - 0	Disk Controller Data Register	
(1) Read or write data bytes from/to the disk controller (2) Must do so at the exact needed rate or there will be errors		

Other Disks $FF50-$FF5F

$FF50-$FF5F (65360-65375)	Other Disks	CoCo 1/2/3
Bit 7		
Bit 6		
Bit 5		
Bit 4		

Bit 3	
Bit 2	
Bit 1	
Bit 0	

(1) TODO - Tandy, Disto mini controller, mirror of drive controller
(2) TODO - Disto mini expansion bus $FF50-$FF57?
(3) TODO - Glenside IDE controller default address $FF50-$FF58
The Glenside IDE board memory map:
$FFx0 - 1st 8 bits of DATA register
$FFx1 - Error (read) / Features (Write) register
$FFx2 - Sector count register
$FFx3 - Sector # register
$FFx4 - Cylinder low byte
$FFx5 - Cylinder high byte
$FFx6 - Device/head register
$FFx7 - Status (read) / Command (Write) register
$FFx8 - 2nd 8 bits of DATA register (latch)
Please note, that if you are using ATAPI, most of these change (which is why the current driver will not handle ATAPI, except for detecting it's presence). (L. Curtis Boyle)

Miscellaneous Hardware

$FF60 (65376)-$FF62 (65378) X-Pad interface

$FF60-$FF62 (65376-65378)	X-Pad interface	CoCo 1/2/3
$FF60	X COORDINATE FOR X-PAD, 0-255	
$FF61	Y COORDINATE FOR X-PAD, 0-191	
$FF62	STATUS REGISTER FOR X-PAD	

(1) Upper left on the X-Pad is (0,0)
(2) Coords wrap around on the X-Pad margins
(3) Reading the x coord causes the y value and status to lock at that time so the values stay in sync for reading.
(4) Status is 4 bits:
Bit 0 - Pen down
Bit 1 - Pen within 1" of surface
Bit 2 - Pen in X-Margin
Bit 3 - Pen in Y-Margin

$FF60 (65376)-$FF67 (65383) CoCo Max A/D Module

$FF60-$FF67 (65376-65383)	CoCo Max A/D Module	CoCo 1/2/3
Bit 7		

TODO
It is unfortunately simplistic to say that the addresses are as you say. The first time an address is

accessed (read), it sets up an A/D conversion cycle for the channel as you specify above. THEN the next access is normally a read which reads the value converted from the previous read access. By doing a read on the next channel, you set up the A/D conversion cycle for the channel read, but read the previous channel's data. Here's another way to look at it.

Access(read)	address	data retrieved
1	$FF60	Whatever channel was set up last
2	$FF61	Data from channel #0 (X pos)
3	$FF62	Data from channel #1 (Y pos)
4	$FF63	Data from channel #2 (pen switch)
5	any	Data from channel #3 (not used in CCMax)
6	Ad-nausium..	

(Nosko S.)

$FF60 (65376)-$FF7F (65407) TC^3 SCSI

$FF60-$FF7F (65376-65407)	TC^3 SCSI	CoCo 1/2/3
TODO TC^3 SCSI interface uses two addresses anywhere in this range		

$FF63 (65379)-$FF67 (65383) Unused

$FF63-$FF67 (65379-65383)	Unused	CoCo 1/2/3
Unused		

$FF68 (65384)-$FF6B (65387) RS-232 PROGRAM PAK Interface

$FF68-$FF6B (65384-65387)	RS-232 PROGRAM PAK Interface	CoCo 1/2/3
$FF68	READ/WRITE DATA REGISTER	
$FF69	STATUS REGISTER	
$FF6A	COMMAND REGISTER	
$FF6B	CONTROL REGISTER	

(1) Based on Synertek 6551 ASCI chip.
(2) Write to STATUS causes a soft reset, read obtain status, an 8-bit field:

Status bit meaning	to clear:
Bit 0 - Parity error = 1, no error = 0	self clearing
Bit 1 - Framing error	self clearing
Bit 2 - Overrun	self clearing
Bit 3 - Receive data register full = 1	read receive data reg
Bit 4 - Transmit data register empty = 1	write transmit data reg
Bit 5 - NOT(DCD) 1 = high	
Bit 6 - NOT(DSR) 1 = high	
Bit 7 - IRQ 1 = interrupt	read status reg

(3) Command register:
Bits 765: parity:

```
      xx0 - parity disabled
      001 - odd parity rec and trans
      011 - even parity rec and trans
      101 - mark parity bit transmitted, parity check disabled
      111 - space parity bit transmitted, parity check disabled
   Bit  4  : echo = 1 (bits 2 and 3 must be 0)
   Bits 32 : transmitter controls
            trans interrupt   NOT(RTS) level   Transmitter
      00 -  disabled          high             off
      01 -  enabled           low              on
      10 -  disabled          low              on
      11 -  disabled          low              trans brk
   Bit  1  : receiver interrupt enable 0 - IRQ enabled with bit 3 of status
                                       1 - IRQ disabled
   Bit  0  : data terminal ready 0 - disable receiver and interrupts
                                 1 - enable receiver and interrupts
(4) Control register:
   Bit    7: stop bit 0 = 1 stop bit, 1 = 2 stop bits (TODO - more)
   Bits  65: data length 00=8, 01=7, 10=6, 11 = 5
   Bit    4: receiver clock source 0 = external, 1=baud rate generator
   Bits 3-0: baud rate generator
     0000 -   16x external clock
     0001 -    50 baud
     0010 -    75 baud
     0011 -   109.92 baud
     0100 -   134.58 baud
     0101 -   150 baud
     0110 -   300 baud
     0111 -   600 baud
     1000 -  1200 baud
     1001 -  1800 baud
     1010 -  2400 baud
     1011 -  3600 baud
     1100 -  4800 baud
     1101 -  7200 baud
     1110 -  9600 baud
     1111 - 19200 baud
```

$FF6C (65388)-$FF6F (65391) Direct Connect Modem Pak

$FF6C-$FF6F (65388-65391)	Direct Connect Modem Pak	CoCo 1/2/3
$FF6C	READ/WRITE DATA REGISTER	
$FF6D	STATUS REGISTER	
$FF6E	COMMAND REGISTER	

$FF6F	CONTROL REGISTER
(1) Same control information as the RS-232 Program Pak	

$FF70 (65392)	LR-Tech SASI controller	CoCo 1/2/3
alternate address of LR-Tech SASI controller		

$FF70 (65392),$FF72 (65394) Musica stereo pack

$FF70,$FF72 (65392,65394)	Musica stereo pack	CoCo 1/2/3
Musica stereo pack - the two stereo channels $FF72 - (*NOT* $FF71) / (Nosko S.)		

$FF70 (65392)-$FF72 (65394) Laser light show D/A

$FF70-$FF72 (65392-65394)	laser light show D/A	CoCo 1/2/3
$FF70	X	
$FF71	Y	
$FF72	Z (intensity)	
laser light show D/A converters (Nosko S.)		

$FF70 (65392)-$FF74 (65396) SPEECH SYSTEMS SUPERVOICE

$FF70-$FF74 (65392-65396)	SPEECH SYSTEMS SUPERVOICE	CoCo 1/2/3
(VOTRAX SC-02) new model (1 MHZ clock) (these can be modified to be 2MHz) (Rodney V Hamilton) TODO		

$FF70 (65392)-$FF74 (65396) Burke & Burke CYBERVOICE

$FF70-$FF74 (65392-65396)	Burke & Burke CYBERVOICE	CoCo 1/2/3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		
Bit 1		
Bit 0		
(VOTRAX SC-02) 2MHz clock, OS9L2/CoCo 3 compatible (Rodney V Hamilton) TODO		

$FF70 (65392)-$FF78 (65400) Glenside IDE controller

$FF70-$FF78 (65392-65400)	Glenside IDE controller	CoCo 1/2/3
$FFx0	1st 8 bits of DATA register	
$FFx1	Error (read) / Features (Write) register	
$FFx2	Sector count register	
$FFx3	Sector # register	
$FFx4	Cylinder low byte	
$FFx5	Cylinder high byte	
$FFx6	Device/head register	
$FFx7	Status (read) / Command (Write) register	
$FFx8	2nd 8 bits of DATA register (latch)	
The Glenside IDE board memory map , alternate address.		

$FF74 (65396)		CoCo 1/2/3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		
Bit 1		
Bit 0		
TODO default address of LR-Tech SASI controller		

$FF74-$FF77 (65396-65399)	Disto SCII	CoCo 1/2/3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		
Bit 1		
Bit 0		
TODO Disto SCII haltless controller additional addresses		

$FF70-$FF79 (65392-65401)	Unused	CoCo 1/2/3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		

Bit 1	
Bit 0	

$FF7A (65392)-$FF7B (65404) Orchestra-90

$FF7A-$FF7B (65392-65404)	Orchestra-90 CoCo 1/2/3
$FF7A	Left Channel
$FF7B	Right Channel
(1) TODO - detail	

$FF7C (65404)	Unused CoCo 1/2/3

$FF7D (65405)-$FF7E (65406) SOUND/SPEECH CARTRIDGE

$FF7D-$FF7E (65405-65406)	SOUND/SPEECH CARTRIDGE CoCo 1/2/3
$FF7D	SOUND/SPEECH CARTRIDGE RESET
$FF7E	SOUND/SPEECH CARTRIDGE READ/WRITE

(1) Based on a SPO256-AL2 Speech Processor and a AY3-8913 Programmable Sound Generator. Also has a PIC7040 with internal 4K ROM, 2K RAM
(2) Set RESET bit 0 to 1 then to 0 to do a reset.
(3) Read from $FF7E bit 7 set indicates previous byte written not yet processed. Bit 6 set indicates chip is currently talking. bit 5 set indicates sound currently playing.

4 . No more info known - TODO

$FF7F(65407) MULTI-PAK PROGRAMMING REGISTER

$FF7F (65407)	MULTI-PAK PROGRAMMING REGISTER CoCo 1/2/3
Bit 7	(2)
Bit 6	(2)
Bits 5-4	Number of active CTS slot (ROM)
Bit 3	(2)
Bit 2	(2)
Bits 1-0	Number of active SCS slot (FDC)

1. Poke value 0 for slot 1, 17 for slot 2, 34 slot 3, 51 slot 4
2. All set means value given is select switch setting binary 00rr00ii bits 5-4: number of active CTS slot (ROM $C000-$DFFF) & CART* select bits 1-0: number of active SCS slot (I/O $FF40-$FF5F)

$FF80-$FFBF (65408-65471)	Unused in CoCo 1/2 CoCo 1/2
(1) $FF90-$FFBF are used in CoCo3 for the GIME chip, elsewhere in this document.	

$FF80-$FF84 (65408-65412)	SPEECH SYSTEMS SUPERVOICE	CoCo 1/2/3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		
Bit 1		
Bit 0		
(VOTRAX SC-02) old model (1 MHZ clock) (these can be modified to be CYBERVOICE address/speed compatible) (Rodney V Hamilton) TODO		

$FF80-$FF8F (65408-65424)	Unused in CoCo 3	CoCo 3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		
Bit 1		
Bit 0		
(1) $FF90-$FFBF are used in CoCo3 for the GIME chip, elsewhere in this doc		

CoCo 3 GIME Hardware Reference

$FF90 (65424) Initialization Register 0 - INIT0

$FF90 (65424)	Initialization Register 0 - INIT0	CoCo 3
Bit 7	CoCo Bit 1 = Color Computer 1/2 Compatible, 0 = CoCo3	
Bit 6	M/P 1 = MMU enabled	
Bit 5	IEN 1 = GIME IRQ output enabled to CPU, 0 = disabled	
Bit 4	FEN 1 = GIME FIRQ output enabled to CPU, 0 = disabled	
Bit 3	MC3 1 = Vector RAM at FEXX enabled, 0 = disabled	
Bit 2	MC2 1 = Standard SCS (DISK) (0=expand 1=normal)	
Bit 1	MC1 ROM Map - see note (1)	
Bit 0	MC0 " "	
(1) MC1 BIT MC0 BIT ROM MAP (VECTORS EXCLUDED)		

0	**X**	**16K INTERNAL, 16K EXTERNAL**
1	**0**	**32K INTERNAL**
1	**1**	**32K EXTERNAL (EXCEPT INTERRUPT VECTORS)**

(2) SCS is Spare Chip Select
(3) To get CoCo 1/2: CoCo bit set, MMU disabled, Video address from SAM, RGB/Comp Palettes => CC2.
(4) To use CoCo 3 graphics, the COCO bit must be set to zero. When using CoCo 1/2 resolutions, the bit is set to 1. RSDOS typically sets the INIT0 register to 196 in CoCo 2 resolutions and 68 when using CoCo 3 graphics modes.

$FF91 (65425) Initialization Register 1 - INIT1

$FF91 (65425)	Initialization Register 1 - INIT1 CoCo 3
Bit 7	Unused
Bit 6	Memory type 1=256K, 0=64K chips
Bit 5	TINS Timer INput clock source 1=279.365 nsec, 0=63.695 usec
Bits 4-1	Unused
Bit 0	MMU Task Register select 0=enable $FFA0-$FFA7 1=enable $FFA8-$FFAF

(1) TINS=1 is a 279.365 ns (3.58 MHZ) clock, not a 70ns clock as published some places. TINS = 0 is default
(2) The TINS bit selects the clock speed of the countdown timer. The 279 ns clock is useful for interrupt driven sound routines while the 63 us (15.87 kHZ) clock is used for a slower timer.
(3) The task register selects which set of MMU bank registers to assign to the CPU's 64K workspace. The task bit is generally set to zero in DECB.

$FF92 (65426) Interrupt Request Enable Register – IRQENR

$FF92 (65426)	Interrupt Request Enable Register - IRQENR CoCo 3
Bits 7-6	Unused
Bit 5	TMR 1=Enable timer IRQ, 0 = disable
Bit 4	HBORD 1=Enable Horizontal border Sync IRQ, 0 = disable
Bit 3	VBORD 1=Enable Vertical border Sync IRQ, 0 = disable
Bit 2	EI2 1=Enable RS232 Serial data IRQ, 0 = disable
Bit 1	EI1 1=Enable Keyboard IRQ, 0 = disable
Bit 0	EI0 1=Enable Cartridge IRQ, 0 = disable

(1) THIS REGISTER WORKS THE SAME AS FIRQENR EXCEPT THAT IT GENERATES IRQ INTERRUPTS.

(2) SEE NOTES FOLLOWING $FF93 FIRQENR FOR MORE INTERRUPT INFORMATION.

$FF93 (65427) Fast Interrupt Request Enable Reg - FIRQENR

$FF93 (65427)	Fast Interrupt Request Enable Reg - FIRQENR CoCo 3
Bits 7-6	Unused
Bit 5	TMR 1=Enable timer FIRQ, 0 = disable

Bit 4	HBORD 1=Enable Horizontal border Sync FIRQ, 0 = disable
Bit 3	VBORD 1=Enable Vertical border Sync FIRQ, 0 = disable
Bit 2	EI2 1=Enable RS232 Serial data FIRQ, 0 = disable
Bit 1	EI1 1=Enable Keyboard FIRQ, 0 = disable
Bit 0	EI0 1=Enable Cartridge FIRQ, 0 = disable

(1) TMR: FIRQ interrupt generated whenever 12 bit timer counts down to zero.
(2) HBORD: Horiz border FIRQ interrupt generated on falling edge of HSYNC.
(3) VBORD: Vert border FIRQ interrupt generated on falling edge of VSYNC.
(4) EI2: Serial FIRQ interrupt generated on falling edge of the signal on PIN 4 of the serial port.
(5) EI1: Keyboard FIRQ interrupt generated whenever a zero appears on any one of PA0-PA6 on the PIA0.
(6) EI0: Cartridge FIRQ interrupt generated on the falling edge of the signal on PIN 8 of the cartridge port.
(7) Reading from the register tells you which interrupts came in and acknowledges and resets the interrupt source.
(8) Here's a table of the interrupt vectors and where they end up going. You can't change the $FFxx vectors, but you can change the $FExx and $01xx vectors which contain jmps/lbras to the interrupt routine.

Be sure to disable the interrupt you are setting before changing values.

Interrupt -> CPU reads -> points to -> jumps to this routine

```
SWI3      $FFF2     $FEEE     $0100
SWI2      $FFF4     $FEF1     $0103
FIRQ      $FFF6     $FEF4     $010F
IRQ       $FFF8     $FEF7     $010C
SWI       $FFFA     $FEFA     $0106
NMI       $FFFC     $FEFD     $0109
RESET     $FFFE     $8C1B
```

This is in order of increasing precedence. Thus an IRQ firing while a FIRQ is being serviced will interrupt the FIRQ. Conversely, a FIRQ never interrupts an IRQ.

Note that the equivalent interrupt output enable bit must be set in $FF90

(9) You can also read these regs to see if there is a LOW on an interrupt input pin. If you have both the IRQ and FIRQ for the same device enabled, you read a 1 bit on both regs if that input is low. For example, if you set $FF02=0 and $FF92=2, then as long as a key is held down, you will read back bit 1 as Set.

$FF94-$FF95 TIMERMSB/TIMERLSB

$FF94 (65428)	Timer register MSB - TIMERMSB	CoCo 3
Bits 7-4	Unused	
Bits 3-0	TMRH - Timer Bits 8-11 - write here to start timer	

$FF95 (65429)	Timer register LSB - TIMERLSB	CoCo 3
Bits 7-0	TIMRL - Timer Bits 0-7	

(1) The 12-bit timer can be loaded with any number from 0-4095. The timer resets and restarts counting down as soon as a number is written to $FF94. Writing to $FF95 does not restart the timer, but the value does save. Reading from either register does not restart the timer.

When the timer reaches zero, it automatically restarts and triggers an interrupt (if enabled). The timer also controls the rate of blinking text. Storing a zero to both registers stops the timer from operating. Lastly, the timer works slightly differently on the 1986 and 1987 versions of the GIME. Neither can actually run a clock count of 1. That is, if you store a 1 into the timer register, the 1986 GIME actually processes this as a '3' and the 1987 GIME processes it as a '2'. All other values stored are affected the same way: nnn+2 for 1986 GIME and nnn+1 for 1987 GIME. (2) Must turn timer interrupt enable off/on again to reset timer IRQ/FIRQ. (3) Storing a $00 at $FF94 seems to stop the timer. Also, apparently each time it passes thru zero, the $FF92/93 bit is set without having to re-enable that Interrupt Request.

$FF96-$FF97 - Unused

$FF96-$FF97 (65430-65431)	Unused	CoCo 3
Bit 7		
Bit 6		
Bit 5		
Bit 4		
Bit 3		
Bit 2		
Bit 1		
Bit 0		
Both registers unused		

$FF98 (65432) Video mode register - VMODE

$FF98 (65432)	Video mode register - VMODE	CoCo 3
Bit 7	BP 0=alphanumeric (text modes), 1=bit plane (graphics modes)	
Bit 6	Unused	
Bit 5	DESCEN 1= extra DESCender ENable(text), swap artifact colors (in gr mode)	
Bit 4	MOCH MOnoCHrome (composite video output) (1=mono), 0 = color	
Bit 3	H50 1=50hz vs 0=60hz bit	
Bit 2 -0	LPR210 - Number of lines/char row	

(1) LPR210 is Lines Per Row:

```
    000 - 1 line/row        100 - 9
    001 - 2 (CoCo1&2)       101 - 10 (Reserved?)
    010 - 3 (CoCo1&2)       110 - 11 (12?(CoCo1&2?))
    011 - 8                 111 - (12?) Infinite*
```

(2) Bit 5 is the artifact color shift bit. Change it to flip Pmode 4 colors.
A One is what is put there if you hold down the F1 key on reset.
POKE &HFF98,&H13 from Basic if colors artifact the wrong way for you.

*Mostly useless, but it does generate a graphics mode where the whole screen is filled with the same line of graphics - like a 320x1 resolution. This can be used for a very fast oscilloscope type display where the program only updates data in one scan line over time and as the screen refreshes, you get a screen full of samples. Sockmaster used it in his Boink bouncing ball demo to take manual control of the vertical resolution of the screen to make the ball appear that it's going up and down (without actually scrolling the whole screen up and down).

$FF99 (65433) Video resolution register - VRES

$FF99 (65433)	Video resolution register - VRES	CoCo 3
Bit 7	Unused(?)	
Bits 6-5	LPF10 – Lines per field	
Bits 4 -2	HRES210 – Horizontal resolution	
Bit 1-0	CO01 – Color bits	

(1) BITS 6-5: LINES PER FIELD LPF:
00 -> 192 SCAN LINES ON SCREEN
01 -> 200 SCAN LINES ON SCREEN
10 -> *ZERO/INFINITE LINES ON SCREEN (UNDEFINED)
11 -> 225 SCAN LINES ON SCREEN

(2) Bits 4-2: Horizontal resolution HR
Graphics modes:
000=16 bytes per row
001=20 bytes per row
010=32 bytes per row
011=40 bytes per row
100=64 bytes per row
101=80 bytes per row
110=128 bytes per row
111=160 bytes per row
Text modes (HR1 - don't care for text):
0x0=32 characters per row
0x1=40 characters per row
1x0=64 characters per row
1x1=80 characters per row

(3) Bits 1-0 CRES Color Resolution
Graphics modes:
00=2 colors (8 pixels per byte)
01=4 colors (4 pixels per byte)
10=16 colors (2 pixels per byte)
11=Undefined (would have been 256 colors!?)
Text modes:
x0=No color attributes
x1=Color attributes enabled

*The zero/infinite scanlines setting will either set the screen to display nothing but border (zero lines) or graphics going all the way up and down out of the screen, never retriggering. It all depends on when you set the register. If you set it while the video raster was drawing the vertical border you get zero lines, and if you set it while video was drawing graphics you get infinite lines. Mostly useless, but it should be possible to coax a vertical overscan mode using this with some tricky timing.

Old SAM modes work if CC Bit set. HR and CRES are Don't Care in SAM mode. Note the correspondence of HR2 HR0 to the text mode's bytes/line.

Commonly used graphics modes:

Width	Colors	HR210	C010	
640	4	111	01	
640	2	101	00	
512	4	110	01	
512	2	100	00	
320	16	111	10	
320	4	101	01	
320	2	011	00	
256	16	110	10	
256	4	100	01	
256	2	010	00	
160	16	101	10	
160	4	011	01	*
160	2	001	00	*
128	16	100	10	*
128	4	010	01	*
128	2	000	00	*

* - not supported. Other combos also possible but not supported.

(4) HiRes text always two bytes per character; even byte 6 bit character, odd byte attribute. Characters from 128 ASCII, no graphic chars.
Format is:
Bit 7 1 = Blink
Bit 6 1 = Underline
Bits 5-3 Foreground Palette 0-7 from $FFB0-$FFB7
Bits 2-0 Background Palette 0-7 from $FFB8-$FFBF

(5) Due to a design error in the GIME, the "200-line" mode only displays 199 lines of active video on the screen. If you do the BASIC pokes for 25 lines on the WIDTH 40 and WIDTH 80 screens, you will see the blinking underscore cursor disappear at the bottom line. If the graphic screens are poked for 200 lines, the bottom-most line will be #198, not #199. Try it and see.
(Rodney V Hamilton)

TODO – check 200 line error

$FF9A (65434) Border color register - BRDR

$FF9A (65434)	Border color register - BRDR	CoCo 3
Bits 7-6	Unused	
Bits 5-0	Border palette color, same format as $FFB0-$FFBF	
(1) This controls the color of the border around the screen. The color bits work the same as the palette registers. This register only controls the border color of CoCo 3 video modes and does not affect Coco 1/2 modes. (2) See $FFB0-$FFBF for color definition. (3) Format depends on Composite or RGB monitor.		

$FF9B (65435)	Disto 2 Meg Upgrade bank	CoCo 3
Bits 7-2		
Bits 1-0	VBANK Used by Disto 2 Meg upgrades to switch between 512K banks	

$FF9C (65436) Vertical scroll register - VSC

$FF9C (65436)	Vertical scroll register - VSC	CoCo 3
Bits7-4	Unused	
Bit 3-0	VSC Vertical smooth scroll 3=MSB <-> LSB=0 vals 0=16 (?)	
The vertical scroll register is used to allow smooth scrolling in text modes. Consecutive numbers scroll the screen upwards one scan line at a time in video modes where more than one scan line makes up a row of text (typically 8 lines per character row) or graphics (double height + graphics).		

TODO – check 0=16 in this case

$FF9D-$FF9E Vertical offset register

$FF9D (65437)	Vertical offset register MSB	CoCo 3
Bits 7-0	Y15-Y8 MSB Start of video in GIME RAM (video location * 2048)	
$FF9E (65438)	**Vertical offset register LSB**	**CoCo 3**
Bits 7-0	Y7-Y0 LSB Start of video in GIME RAM (video location * 8)	
$FF9D VERTICAL OFFSET V SCROLL MUST BE $0F $FF9D Screen start address Bits 18-11 $FF9E Screen Start Address Register 0 (bits 10-3) $FF9E V OFFSET #2 WORD = ADDRESS/8 EX. $C000 = $60000/8 BIT 7 | BIT 0 LSB $FF9E Screen start address Bits 10-3 DDDDDDDDEEEEEEEE000 $FF9E (65438) Vertical offset register LSB		

Y15-Y0 is used to set the video mode to start in any GIME memory location in 512K by steps of 8 bytes. On a 128K machine, the memory range is $60000-$7FFFF. There is a bug in some versions of the GIME that causes the computer to crash when you set odd numbered values in $FF9E in some resolutions, so it's safest to limit positioning to steps of 16 bytes. Fortunately, you can use $FF9F to make up for it and get steps as small as 2 bytes.

$FF9F (65439) Horizontal offset register

$FF9F (65439)	Horizontal offset register - TODO - CoCo 3
Bit 7	HVEN 1=Horizontal virtual screen enable (256 bytes per row) 0=Normal horizontal display
Bits 6-0	0-127 byte offset from $FF9D/$FF9E

(1) If Bit 7 set & in Text mode there are 128 chars (only 80 seen)/line. This allows an offset to be specified into a virtual 128-char/line screen, useful for horizontal hardware scrolling on wide text or spreadsheets.

(2) If you set Bit 7 and you're in graphics mode, you can scroll across a 128-byte picture. To use this, of course, you'd have to write your own graphics routines. On my machine, though, an offset of more than about 5 crashes.

```
Bit 7
Bits 6-0 X6-X0 Horizontal offset address (video location *2)
```

(3) You can combine the horizontal and vertical offsets to get a higher definition video position: Y15-Y4,X6-X0 which gives you 19 bit positioning by steps of 2 bytes. Otherwise, you can use this register to do scrolling effects. The virtual screen mode allows you to set up a 256 byte wide graphics or text screen, showing only part of it at a time and allowing you to scroll it vertically (horizontally TODO ?).

$FFA0-$FFAF (65440-65455) MMU bank registers (tasks 0 and 1)

$FFA0-$FFA7 (65440-65447)	MMU bank registers (task 0) CoCo 3
$FFA8-$FFAF (65448-65455)	MMU bank registers (task 1) CoCo 3
$FFA0/8	Page $0000-$1FFF
$FFA1/9	Page $2000-$3FFF
$FFA2/A	Page $4000-$5FFF
$FFA3/B	Page $6000-$7FFF
$FFA4/C	Page $8000-$9FFF
$FFA5/D	Page $A000-$BFFF
$FFA6/E	Page $C000-$DFFF
$FFA7/F	Page $E000-$FFFF (or $E000-$FDFF - see (TODO 1))

1. The MMU registers select 8K pages from the GIME addressable space $0-$7FFFFF into

CPU addressable space $0-$FFFF in 8K blocks.

2. The pages are numbered by the top 6 bits of the address, and are $30-$3F for a 128K machine, and $00-$3F for a 512K machine.
3. In a 128K machine pages $0-$2F are copies of pages $30-$3F.
4. The registers to set the various 8K blocks, and power-up contents:

MMU Register:				CPU:
Task0	Task1	Logical Address / Block#		Default page
$FFA0	$FFA8	$0000 - $1FFF	0	$38
$FFA1	$FFA9	$2000 - $3FFF	1	$39
$FFA2	$FFAA	$4000 - $5FFF	2	$3A
$FFA3	$FFAB	$6000 - $7FFF	3	$3B
$FFA4	$FFAC	$8000 - $9FFF	4	$3C
$FFA5	$FFAD	$A000 - $BFFF	5	$3D
$FFA6	$FFAE	$C000 - $DFFF	6	$3E
$FFA7	$FFAF	$E000 - $FDFF	7	$3F

5. $FF91 Bit 0 selects task 0 (bit = 0) or task 1 (bit = 1).Task 0 uses MMU pages from $FFA0-$FFA7 and Task 1 uses MMU pages from $FFA8-$FFAF.
6. $FE00-$FFFF can be held constant at $7Fexx.
7. If you don't know it is safe not to, you should turn off interrupts before swapping MMU blocks. Be very careful when swapping out ROM or low system RAM.
8. These registers can be read, but the top two bits must be masked out since they might contain garbage.
9. See the section on memory mapping and memory maps for more details TODO.
10. Here is the GIME address view and default page usage:

Page	GIME Address	CPU Address*	Standard Page Contents
$00-2F	$00000-$5FFFF		512K upgrade RAM, not in 128K
$30	$60000-$61FFF		Hi-Res page #1
$31	$62000-$63FFF		Hi-Res page #2
$32	$64000-$65FFF		Hi-Res page #3
$33	$66000-$67FFF		Hi-Res page #4
$34	$68000-$69FFF		HGET/HPUT buffer
$35	$6A000-$6BFFF		Secondary Stack
$36	$6C000-$6DFFF		Hi-Res text screen RAM
$37	$6E000-$6FFFF		unused
$38	$70000-$71FFF	$0000-$1FFF	Basic memory
$39	$72000-$73FFF	$2000-$3FFF	Basic memory
$3A	$74000-$75FFF	$4000-$5FFF	Basic memory
$3B	$76000-$77FFF	$6000-$7FFF	Basic memory
$3C	$78000-$79FFF	$8000-$9FFF	Extended Basic Interpreter
$3D	$7A000-$7BFFF	$A000-$BFFF	Color Basic Interpreter
$3E	$7C000-$7DFFF	$C000-$DFFF	Disk Basic Interpreter
$3F	$7E000-$7FFFF	$E000-$FFFF	Super Basic, GIME regs, I/O, Interrupts

$FFB0-$FFBF (65456-65471) Color palette registers

$FFB0-$FFBF (65456-65471)	Color palette registers -TODO	CoCo 3
$FFB0-$FFBF	Palette entries 0-15	

```
RGB Mode: Bits 7-6 Unused
          Bit 5 = High order Red           R1
          Bit 4 = High order Green         G1
          Bit 3 = High order Blue          B1
          Bit 2 = Low order Red            R0
          Bit 1 = Low order Green          G0
          Bit 0 = Low order Blue           B0
Composite mode:
          Bits 7-6 Unused
          Bits 5-4 = 4 intensity levels   I1 I0
          Bits 3-0 = 16 colors            P3 P2 P1 P0
```

Todo - RGB/Composite bit?, names? Of the 16 composite colors?

(1) These 16 registers set the 16 colors used in the system.
(2) Their format depends on the RGB/Composite bit setting in TODO
(3) They can be read, but the top two (or three) bits must be masked off for correctness.
(4) Both reading and writing to the palette registers causes a small glitch on the screen, which can be avoided by changing the palettes while the video retrace is in the vertical or horizontal border.
(5) The BORDER register uses the same format, and also depends on the RGB/COMPOSITE setting TODO
(6) $FFB0-$FFB7 are also used for the text mode character background colors, and $FFB8-$FFBF TODO
(7) Default values:

Here are the default RGB palette values on power up:

$FFB0	GREEN	$12	$FFB8	BLACK	$00
$FFB1	YELLOW	$36	$FFB9	GREEN	$12
$FFB2	BLUE	$09	$FFBA	BLACK	$00
$FFB3	RED	$24	$FFBB	BUFF	$3F
$FFB4	BUFF	$3F	$FFBC	BLACK	$00
$FFB5	CYAN	$1B	$FFBD	GREEN	$12
$FFB6	MAGENTA	$2D	$FFBE	BLACK	$00
$FFB7	ORANGE	$26	$FFBF	ORANGE	$26

Here are the default Composite palette values on power up:

$FFB0	GREEN	$12	$FFB8	BLACK	$00
$FFB1	YELLOW	$24	$FFB9	GREEN	$12
$FFB2	BLUE	$0B	$FFBA	BLACK	$00
$FFB3	RED	$07	$FFBB	BUFF	$3F
$FFB4	BUFF	$3F	$FFBC	BLACK	$00
$FFB5	CYAN	$1F	$FFBD	GREEN	$12

$FFB6 MAGENTA	$09	$FFBE BLACK	$00
$FFB7 ORANGE	$26	$FFBF ORANGE	$26

TODO – merge default colors from the color section.

SAM registers $FFC0-$FFDF

$FFC0 (65472)-$FFC5 (65477) SAM Video Display - SAM_Vx

$FFC0-$FFC5 (65472-65477)	SAM Video Display - SAM_Vx	CoCo 1/2/3
$FFC0/1	SAM_V0, or V0CLR/V0SET	
$FFC2/3	SAM_V1, or V1CLR/V1SET	
$FFC4/5	SAM_V2, or V2CLR/V1SET	

(1) This allows setting video modes on the CoCo 1 and 2
(2) SAM_Vx are three pairs of addresses (V0-V2), and poking any value to EVEN addresses sets bit Vx off (0) in Video Display Generator (VDG) circuitry. Poking a value to ODD addresses sets bit on (1) in VDG circuit.
(3) These registers work with $FF22 for setting modes, and should match up
(4) Default screen mode is semigraphic-4
(5) Mode correspondence between the SAM and the VDG:

```
      Mode                  VDG Settings     SAM
                          A/G  GM2 GM1 GM0  V2/V1/V0  Desc.       RAM used
                                                      x,y,clrs    in hex(dec)
 Internal alphanumeric     0    X   X   0    0 0 0    32x16 ( 5x7 pixel ch)
 External alphanumeric     0    X   X   1    0 0 0    32x16 (8x12 pixel ch)
 Semigraphic-4             0    X   X   0    0 0 0    32x16 ch, 64x32 pixels
 Semigraphic-6             0    X   X   1    0 0 0    64x48 pixels
 Full graphic 1-C          1    0   0   0    0 0 1    64x64x4     $400(1024)
 Full graphic 1-R          1    0   0   1    0 0 1    128x64x2    $400(1024)
 Full graphic 2-C          1    0   1   0    0 1 0    128x64x4    $800(2048)
 Full graphic 2-R          1    0   1   1    0 1 1    128x96x2    $600(1536)
 Full graphic 3-C          1    1   0   0    1 0 0    128x96x4    $C00(3072)
 Full graphic 3-R          1    1   0   1    1 0 1    128x192x2   $C00(3072)
 Full graphic 6-C          1    1   1   0    1 1 0    128x192x4   $1800(6144)
 Full graphic 6-R          1    1   1   1    1 1 0    256x192x2   $1800(6144)
 Direct memory access      X    X   X   X    1 1 1
```

(6) Notes:
- The graphic modes with -C are 4 color, -R is 2 color.
- 2 color mode - 8 pixels per byte (each bit denotes on/off)
  4 color mode - 4 pixels per byte (each 2 bits denotes color)
- CSS (in FF22) is the color select bit:
  Color set 0: 0 = black, 1 = green for -R modes
  00 = green, 01 = yellow for -C modes
  10 = blue, 11 = red for -C modes
  Color set 1: 0 = black, 1 = buff for -R modes

00 = buff, 01 = cyan, for -C modes
10 = magenta, 11 = orange for -C modes

In semigraphic-4 mode, each byte is a char or 4 pixels:
bit 7 = 0 -> text char in following 7 bits
bit 7 = 1 -> graphic: 3 bit color code, then 4 bits for 4 quads of color
colors 000-cyan, yellow, blue, red, buff, cyan, magenta, orange=111
quad bits orientation UL, UR, LL, LR

In semigraphic-6 mode, each byte is 6 pixels:
bit 7-6 = C1-C0 color from 4 color sets above
bit 5-0 = 6 pixels in 2x3 block, each on/off
TODO - orientation

Example: To set 6-C color set 0, lda #$E0, sta in $FF22, $FFC3, $FFC5
To return to text mode, clra, sta in $FF22, $FFC2, $FFC4
(7) In the CoCo 3, The SAM is mostly CoCo 1/2 compatible Write-Only registers

$FFC6 (65478)-$FFD3 (65491) SAM Page Select Reg-SAM_Fx

$FFC6-$FFD3 (65478-65491)	SAM Page Select Reg-SAM_Fx	CoCo 1/2/3
$FFC6/7	SAM_F0, or F0CLR/F0SET	
$FFC8/9	SAM_F1, or F1CLR/F1SET	
$FFCA/B	SAM_F2, or F2CLR/F2SET	
$FFCC/D	SAM_F3, or F3CLR/F3SET	
$FFCE/F	SAM_F4, or F4CLR/F4SET	
$FFD0/1	SAM_F5, or F5CLR/F5SET	
$FFD2/3	SAM_F6, or F6CLR/F6SET	

(1) These registers denote the start of the image in RAM to display in CoCo 1 and 2 text and graphics modes. The value in $F0-$F6 times 512 is the start of video RAM.
(2) SAM_Fx are seven pairs of addresses ($F0-$F6), and poking any value to EVEN addresses sets bit Fx off (0) in Video Display Generator (VDG) circuitry. Poking value to ODD addresses sets bit on (1) in VDG circuit.

$FFD4 (65492)-$FFD5 (65493) SAM Page Select Reg-SAMPAG

$FFD4-$FFD5 (65492-65493)	SAM Page Select Reg-SAMPAG	CoCo 1/2/3
$FFD4	Any write sets page #1 P1 control bit to 0, 0 = normal	
$FFD5	Any write sets page #1 P1 control bit to 1	

(1) page register MPU addresses $0000-$7FFF, apply page #1 if P1 = 1
TODO – meaning?

$FFD6 (65494)-$FFD9 (65497) Clock Speed R0/R1 - SAM_R0/1

$FFD6-$FFD9 (65494-65497)	Clock Speed R0/R1 - SAM_R0/1	CoCo 1/2/3
$FFD6	SAM_R0 - Any write sets R0 control bit to 0	
$FFD7	| - Any write sets R0 control bit to 1	
$FFD8	SAM_R1 - Any write sets R1 control bit to 0	
$FFD9	| - Any write sets R1 control bit to 1	

(1) R1-R0: 00-0.89 MHZ only, 01-0.89/1.78 MHZ <== both transparent refresh
10-1.78 MHZ only, 11-1.78 MHZ TODO – meaning?

(2) May not work on early CoCo1 (and 2?), but works on all CoCo 3's (true?)
(3) 0.89 Mhz: no address-dependent speed , default setting?
(4) Speedup only for ROM accesses?
(5) These are commonly used as follows:
Slow poke: $FFD8 write selects 0.89 Mhz CPU clock
Fast poke: $FFD9 write selects 1.78 Mhz CPU clock

(6) Switching the SAM into 1.8MHz operation gives the CPU the time ordinarily used by the VDG and refresh, so the display shows garbage, so this mode is seldom used. The SAM in Address Dependent mode, where ROM reads (since they do not use the DRAM) occur at 1.8MHz but regular RAM access occurs at .89MHz, runs the BASIC interpreter from ROM twice as fast, nearly doubling BASIC program performance.

$FFDA (65498)-$FFDD (65501) Memory size M0/M1 - SAM_M0/1

$FFDA-$FFDD (65498-65501)	Memory size M0/M1 - SAM_M0/1	CoCo 1/2/3
$FFD6	SAM_M0 - Any write sets M0 control bit to 0	
$FFD7	- Any write sets M0 control bit to 1	
$FFD8	SAM_M1 - Any write sets M1 control bit to 0	
$FFD9	- Any write sets M1 control bit to 1	

(1) M1-M0: 00 - 4K, 01 - 16K
10 - 64K (all 3 dynamic), 11 = 64K static
(2) Todo - is this right? Or Dragon only?

$FFDE/$FFDF (65502/65503) ROM/RAM map type - SAM_TYP

$FFDE-$FFDF (65502-65503)	ROM/RAM map type - SAM_TYP	CoCo 1/2/3
$FFDE	Any write switches system ROMs into memory map (ROM mode)	
$FFDF	Any write selects all-RAM mode (RAM mode)	

(1) RAM accesses use MMU translations in CoCo 3

(2) Default mode 0 - ROM Mode CoCo 1/2, Default mode 1 - RAM Mode CoCo 3 (3) These registers are often called TY=0 and TY=1

Interrupt Vectors

$FFE0-$FFF1 (65504/65522) Reserved

$FFE0-$FFF1 (65504-65522)	Reserved	CoCo 1/2/3
|(1) Reserved for future enhancements :)		

$FFF2-$FFFF (65523/65535) Interrupt vectors

$FFF2-$FFFF (65523-65535)	Interrupt vectors	CoCo 1/2/3
$FFF2/3	SWI3 points to $FEEE	
$FFF4/5	SWI2 points to $FEF1	
$FFF6/7	FIRQ points to $FEF4	
$FFF8/9	IRQ points to $FEF7	
$FFFA/B	SWI points to $FEFA	
$FFFC/D	NMI points to $FEFD	
$FFFE/F	RESET points to $8C1B	

(1) **WHEN AN INTERRUPT OF THE GIVEN TYPE OCCURS, THE VECTOR IS LOADED INTO THE PROGRAM COUNTER, WHICH POINTS TO THE ADDRESS GIVEN ABOVE. YOU CAN SET YOUR OWN INTERRUPT ROUTINES BY REPLACING THE $FEXX VALUES WITH YOUR OWN LBRA XXXX VALUES**

(2) Turn off interrupts before setting a new value.
(3) Restore what was there to restore the system
(4) See also the section on interrupts in this document.

CoCo 3 Detailed Memory Map

This memory map also has a lot of useful information for the CoCo 1 and CoCo 2. This section also contains some information on CoCo clones: Dragon 32 & 64.

Format conventions:
 $xxxx references a hexadecimal CPU memory address
 0xab or 0xabcd are C style hexadecimal constants
 %TITLE% shows a 'standard' assembler reference
 UPPERCASE words typically refer to Basic keywords or Assembler mnemonics
 (0x1234) Numbers in brackets refer to the default value at power-up

Abbreviations:
 CoCo refers to the Tandy CoCo only
 D32 only applicable to Dragon 32
 D64 only applicable to Dragon 64
 DOS refers to a generic DragonDos compatible unless stated otherwise
 lsb least significant byte
 msb most significant byte
 ptr pointer (or address of)
 w/o without

```
0000        BREAK message flag - if negative print BREAK
0001        String delimiting char (0x22 '"')
0002        Another delimiting char (0x22 '"')
0003        General counter byte
0004        Count of IFs looking for ELSE
0005        DIM flag
0006        %VALTYP% Variable type flag (0x00 numeric, Non-0x00 string)
0007        Garbage collection flag
0008        Subscript allowed flag
0009        INPUT/READ flag
000a        Arithmetic use
000b:000c   String ptr first free temporary
000d:000e   String ptr last free temporary
000f-0018   Temporary results
0019:001a   Start address of BASIC program ($1e01, $2401 with DOS)
001b:001c   Start address of simple variables
001d:001e   Start address of array variables
001f:0020   End of storage, Start of unused mem after BASIC program
0021:0022   Top of stack, growing down ($7e36)
0023:0024   Top of free string space ($7ffe)
0025:0026   Temp Ptr to string in string space
0027:0028   Top of Ram available to BASIC - returned by DOS HIMEM ($7ffe)
0029:002a   Last/CONT line number
002b:002c   Temp/Input line number store
002d:002e   Ptr to next statement to be executed
002f:0030   Direct mode command text pointer
0031:0032   Current DATA statement line number
0033:0034   Ptr to next item in current DATA statement
0035:0036   Ptr to keyboard input buffer
```

```
0037:0038   Ptr to variable last in use
0037:0038   ASCII codes of last variable used
0039:003a   VARPTR address of last variable used
003b-004e   Evaluation variables
0041:0042   High end destination addr for block
0043:0044   High end origin addr
0045:0046   Low end destination addr for block
0047:0048   Low end origin addr
004f-0054   Floating Point Accumulator Num 1
004f        Exponent
0050-0053   Mantissa
0050:0051   16 bit values in FAC stored here
0052:0053   VARPTR of variables is stored here
0054        Mantissa Sign (0x00 positive, 0xff negative)
0055        Temp sign of FAC
0056        String variable length
0057-005b   String Descriptor temporaries
005c-0061   Floating Point Accumulator Num 2
0062        Sign comparison
0062-0067   Misc use
0063        CoCo - Extended precision byte
0068:0069   Current Line number (0xffff in direct mode)
006a-006e   Device Params used in PRINT
006a        Device Comma field width (VDU - 0x10)
006b        Device Last comma field
006c        Device Current column num (VDU - 0x00-0x1f)
006d        Device Line width - num chars per line (VDU 0x20)
006e        Cassette I/O in progress flag - 0xff on input or output occurring
006f        %DEVNUM% Current device number
                    0x00 VDU screen
                    0x01-0x04 DOS - DosPlus only - drive number.
                    0xfd serial port (Dragon 64 only)
                    0xfe printer
                    0xff tape
0070        Cassette EOF flag - non-zero if EOF - used by EOF(-1)
0071        Restart flag - if not 0x55 cold start on reset, see $0072
0072:0073   Restart vector - Following a reset if $0072 pts to a NOP opcode &

                $0071 is 0x55 then a warm start is performed to this vector
                else a cold start. (0xb44f) (DOS SuperDosE6 $c706)
0074:0075   Physical end of Ram minus 1 (0x7ffe)
0076:0077   Unused
0078        Cassette status
                0x00 closed
                0x01 input
                0x02 output
0079        Cassette I/O - Buffer size - bytes in block
007a:007b   Header buffer addr - ptr to filename block
007c        %BLKTYP% Cassette block type
                0x00 filename
                0x01 data
                0xff EOF block
007d        %DBLEN% Cassette block length, number bytes read/to write
007e:007f   %DBADR% Cassette I/O Buffer address
            Contains 1 + End address of last program loaded
0080        Cassette I/O - block checksum used internally
0081        Cassette I/O - error code
```

```
                0x00 none
                0x01 CRC (checksum) error
                0x02 attempt to load into ROM
0082        Cassette I/O - Pulse width counter
0083        Cassette I/O - Sync bits counter
0084        Cassette I/O - Bit phase flag
0085        Last sine wave value for output to DAC
0086        Data for low res SET/RESET, POINT routines
0087        ASCII code of last key pressed (cleared by Break check)
0088:0089   Current VDU cursor addr (typ 0x0400-0x05ff)
008a:008b   Gen purpose 16bit scratch pad / 16bit zero (0x0000)
008a:008b   CoCo - Motor on delay
008c        Sound pitch frequency
008d:008e   Gen purpose countdown (?sound timer)
008f        Cursor flash counter (0x20)
0090:0091   Cassette leader byte count - number of 0x55 bytes written as sync
                leader (D32 - 0x0080, D64 - 0x0100)
0092        Minimum cycle width of 1200Hz (0x12)
0092:0093   CoCo - Cassette leader byte count
0093        Minimum pulse width of 1200Hz (0x0a)
0094        Maximum pulse width of 1200Hz (0x12)
0095:0096   Motor on delay (0xda5c = approx 0.5s)
0095:0096   CoCo - Serial Baud rate constant (0x0057 = 600 baud)
0097:0098   Keyboard scan debounce delay constant (0x045e)
0097:0098   CoCo - Serial Line Printer End of Line delay (0x0001)
0099        Printer comma field width (0x10 = 16)
009a        Printer last comma field (0x74 = 116) (CoCo 0x70 = 112)
009b        Printer line width dflt (0x84 = 132)
009c        Printer head column posn == POS(-2),
                Updated by LPOUT ($800f) routine
009d:009e   EXEC default entry address
                (D32 - $8b8d = ?FC ERROR; D64 - $bf49 = Boot 64k mode)
009f-00aa    %CHRGET% Self modifying routine to read next char
009f:00a0       INC <$A7
00a1:00a2       BNE $00A5
00a3:00a4       INC <$A6
00a5-00a7       LDA >xxxx
00a6:00a7       Ptr to next character to read
00a8-00aa       JMP $BB26
00ab-00ae   Used by RND
00af        TRON/TROFF trace flag - non zero for TRON
00b0:00b1   Ptr to start of USR table ($0134; DOS - $0683)
00b2        Current foreground colour (0x03)
00b3        Current background colour (0x00)
00b4        Temp/active colour in use
00b5        Byte value for current colour - ie bit pattern
00b6        Graphics PMODE number in use (0x00)
00b7:00b8   Ptr to last byte+1 of current graphics mode ($0c00 w/o Dos)
00b9        Number of bytes per line in current PMODE (0x10)
00ba:00bb   Ptr to first byte of current graphics mode ($0600)
00bc        Msb of start of graphics pages (0x06 or 0x0c with Dos)
00bd:00be   Current X cursor position (not user available ?)
00bf:00c0   Current Y cursor position (not user available ?)
00c1        Colour set currently in use (0x08 if colorset 1)
00c2        Plot/Unplot flag: 0x00 reset, non zero set
00c3:00c4   Current horizontal pixel number
00c5:00c6   Current vertical pixel number
```

```
00c7:00c8   Current X cursor coord (0x0080)
00c9:00ca   Current Y cursor coord (0x0060)
00cb:00cc   CIRCLE command X coood as if drawn in PMODE 4
00cd:00ce   CIRCLE command Y coord as if drawn in PMODE 4
00cf:00d0   CIRCLE radius as if drawn in PMODE 4
00cf:00d0   RENUM increment value
00d1:00d2   RENUM start line
00d3:00d4   CLOADM 2's complement load offset
00d5:00d6   RENUM new start line
00d7        EDIT line length (not user available)
00d7        PLAY -
00d8        PLAY - bytes left in string
00d9:00da   PLAY - ptr to current char in string
00d8-00dd   Graphics use ?
00de        PLAY: Current octave in use (0-4) (0x02)
00df:00e0   PLAY: Volume data for volume setting (D32 - 0xba42) (D64 - 0xb844)
00e1        PLAY: Current note length (0x04)
00e2        PLAY: Current tempo (0x02)
00e3:00e4   PLAY: Music duration count
00e5        PLAY: Music dotted note flag
00e6-00ff   D32 - Unused in Dragon 32 w/o DOS
00e6        CoCo - baud rate constant
00e7        Coco - Input timeout constant
00e8        Current angle used in DRAW (??)
00e9        Current scale used in DRAW (??)
00ea-00f6   DOS - Used by DragonDos
00f8        DOS - sector currently seeking {SuperDos Rom}
0100-0102   SWI3 Secondary vector (Uninitialised)
0103-0105   SWI2 Secondary vector (Uninitialised)
0106-0108   SWI Secondary vector (Uninitialised)
0109-010b   NMI Secondary vector (Uninitialised)
                (CoCo DOS JMP $d7ae; SuperDos E6 JMP $c71e)
010c-010e   IRQ Secondary vector - JMP $9d3d
                (CoCo JMP $a9b3 or $894c (extended); CoCo DOS JMP $d7bc;
                SuperDos E6 JMP $c727)
010f-0111   FIRQ Secondary vector - JMP $b469
                (CoCo JMP $a0f6; SuperDos E6 JMP $c7da)
0112:0113   TIMER value
0114        Unused
0115-0119   Random number seeds (0x80, 0x4f, 0xc7, 0x52, 0x59)
011a-011f   D32 - Unused
011a        D64 - %FLAG64% checked on Reset from 64K mode if 0x55 then
                checksum at $011b is checked against current contents of RAM,
                if the same then a warm start is performed (64 mode) else a
                cold start (32 mode)
011a        CoCo - Caps lock, 0x00 lower, non-0x00 upper
011b:011c   D64 - %CSUM64% 16bit sum of words of BASIC Rom-in-ram in 64K mode
                from $c000 to $feff
011b:011c   CoCo - Keyboard Delay constant
011d-011f   CoCo - JMP $8489 ?
011d        D64 - %LSTKEY% Last key code return by keybd poll routine
011e        D64 - %CNTDWN% Auto repeat countdown
011f        D64 - %REPDLY% Auto repeat inter-repeat delay value (0x05)
0120        %STUB0% Stub 0 - Number of reserved words (0x4e)
0121:0122   Stub 0 - Ptr to reserved words table ($8033)
0123:0124   Stub 0 - Ptr to reserved words dispatch table ($8154)
0125        Stub 0 - Number of functions (0x22)
```

```
0126:0127   Stub 0 - Ptr to reserved function words table ($81ca)
0128:0129   Stub 0 - Ptr to function words dispatch table ($8250)
012a        %STUB1% Stub 1 - Number of reserved words (0x00)
                (DOS 0x1a)
012b:012c   Stub 1 - Ptr to reserved words table (0x0000)
                (DOS $ded4; SuperDosE6 $deda)
012d:012e   Stub 1 - Ptr to reserved words token processing routine
                ($89b4; DOS $c64c; SuperDosE6 $c670)
012f        Stub 1 - Number of functions (0x00)
                (DOS 0x07)
0130:0131   Stub 1 - Ptr to function table (0x0000)
                (DOS $debb; SuperDosE6 $dec1)
0132:0133   Stub 1 - Ptr to function token processing routine
                ($89b4; DOS $c667; SuperDosE6 $c68b)
0134        %STUB2% Stub 2 - acts as a stub terminator under DOS
0134-0147   USR address table, relocated by DOS (10 x 2 bytes) ($8b8d)
0148        Auto line feed flag on buffer full - setting this to 0x00 causes
                a EOL sequence to be sent to printer when buffer reaches
                length in $009b (0xff)
0149        Alpha Lock flag    - 0x00 Lower case, 0xff Upper case (0xff)
014a-0150   Line Printer End of line termination sequence
014a        Number of bytes in EOL sequence 1-6 (0x01)
014b        EOL chr 1 (0x0d CR)
014c        EOL chr 2 (0x0a LF)
014d        EOL chr 3 (D64 - 0x00; D32 - 0x20 ' ')
014e        EOL chr 4 (D64 - 0x00; D32 - 0x44 'D' Duncan)
014f        EOL chr 5 (D64 - 0x00; D32 - 0x4e 'N' N.)
0150        EOL chr 6 (D64 - 0x00; D32 - 0x4f 'S' Smeed)
0151-0159   Keyboard matrix state table
0152-0159   CoCo - Keyboard roll-over table
015a-015d   %POTVAL% Joystick values (0-63)
015a        Right Joystick, x value == JOYSTK(0)
015b        Right Joystick, y value == JOYSTK(1)
015c        Left Joystick, x value == JOYSTK(2)
015d        Left Joystick, y value == JOYSTK(3)
015e-01a8   RAM hooks - each is called from ROM with a JSR before carrying out
                the specified function
015e-0160   Device Open (DOS JMP $d902; SuperDosE6 $d8f4)
0161-0163   Verify Device Number (DOS SuperDosE6 JMP $d8ec)
0164-0166   Device Init (DOS SuperDosE6 JMP $c29c)
0167-0169   Output char in A to DEVN (DOS JMP $d8fa; SuperDosE6 $d90b)
0167        Setting to 0xff disables keyboard ?!?
                Setting to 0x39 (RTS) allows use of SCREEN 0,1 etc. ??
016a-016c   Input char from DEVN to A (DOS SuperDosE6 JMP $c29c)
016d-016f   Input from DEVN using INPUT (DOS SuperDosE6 JMP $c29c)
0170-0172   Output to DEVN using PRINT (DOS SuperDosE6 JMP $c29c)
0173-0175   Close all files (DOS SuperDosE6 JMP $c29c)
0176-0178   Close file(DOS JMP $d917; SuperDosE6 $d6f5)
0179-017b   Command Interpreter - interpret token in A as command
                (DOS SuperDosE6 JMP $c29c)
017c-017e   Re-request input from keyboard (DOS JMP $d960; SuperDosE6 $d954)
017f-0181   Check keys - scan for BREAK, SHIFT+'@'
                (DOS SuperDosE6 JMP $c29c)
017f        Setting this to 0x9e disables LIST/DIR
0182-0184   Line input from DEVN using LINE INPUT
                (DOS JMP $d720; SuperDosE6 $dac5)
0185-0187   Close BASIC file read in and goto Command mode
```

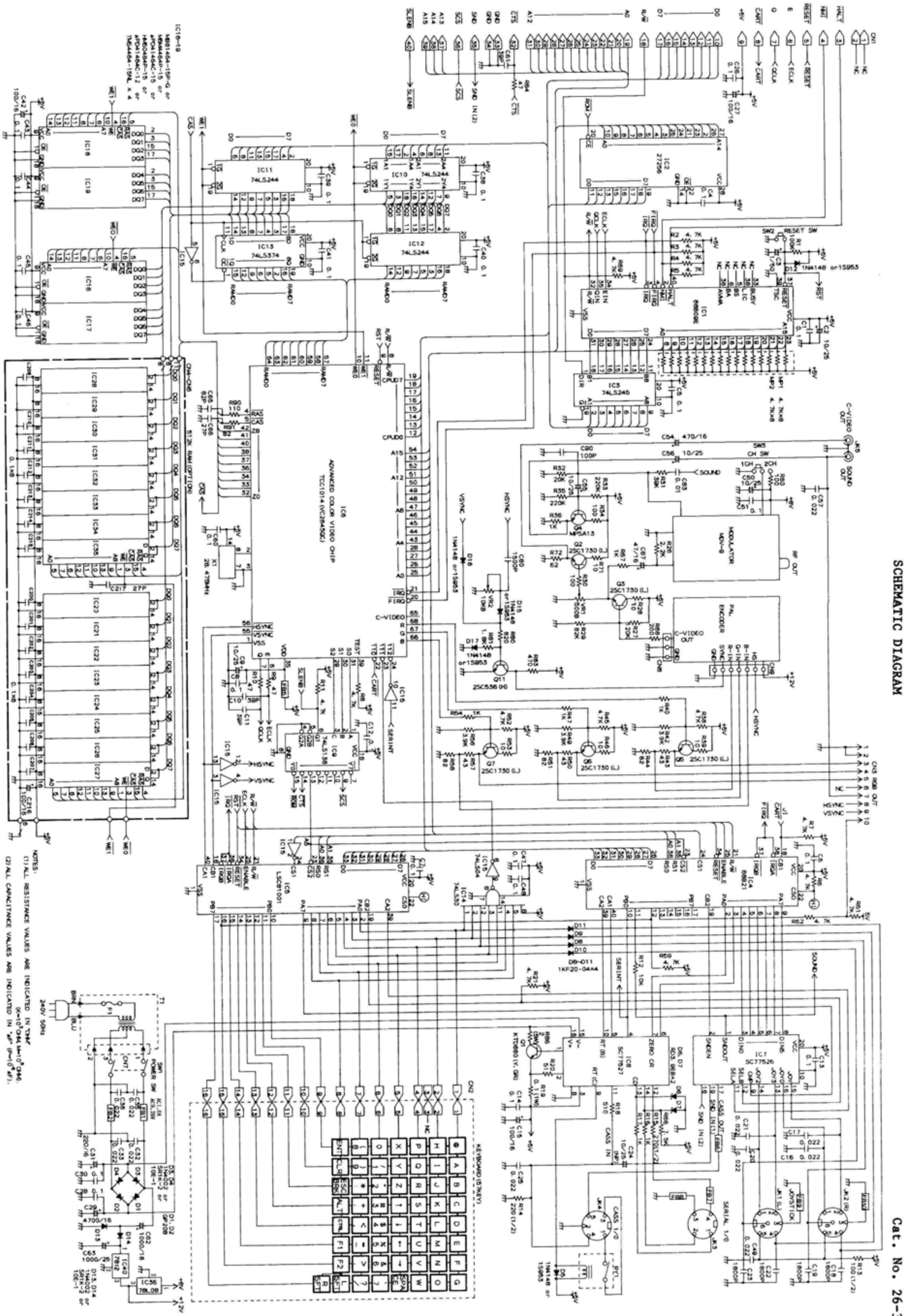
```
                (DOS SuperDosE6 JMP $c29c)
0188-018a   Check EOF on DEVN (DOS JMP $dd4d; SuperDosE6 $dd54)
018b-018d   Evaluate expression (DOS SuperDosE6 JMP $c29c)
018e-0190   User error trap, called from $8344
                (DOS SuperDosE6 JMP $c29c)
0191-0193   System error trap, called from $8344
                (DOS JMP $c69e; SuperDosE6 $c6c5)
0194-0196   Run Link - used by DOS to RUN filename
                (DOS JMP $d490; SuperDosE6 $d4b7)
0197-0199   Reset Basic Memory, editing or entering BASIC lines
019a-019c   Get next command - reading in next command to be executed
019d-019f   Assign string variable
01a0-01a2   Screen access - CLS, GET, PUT
01a3-01a5   Tokenise line
01a6-01a8   De-Tokenise line

01a9-01d0   String buffer area
01d1        Cassette filename length in range 0-8
01d2-01d9   Cassette filename to search for or write out
01da-02d8   Cassette I/O default data buffer - 255 bytes
01da-0268   D64 - 64K mode bootstrap routine is copied here to run
01da-01e1   Cassette buffer - filename of file read
01e2        Cassette buffer - filetype
                0x00 BASIC program
                0x01 Data
                0x02 Machine code
01e3        Cassette buffer - ASCII flag
                0x00 Binary
                0xff ASCII flag
01e4        Cassette buffer - gap flag
                0x00 Continous
                0xff Gapped file
01e5:01e6   Cassette buffer - Entry (Exec) addr of m/c file
01e7:01e8   Cassette buffer - Load address for ungapped m/c file
02d9-02dc   BASIC line input buffer preamble
02dd-03d8   BASIC line input buffer - used for de-/tokenising data
02dd-03dc   CoCo - 255 byte keyboard buffer
02e1-033b   CoCo - 90 byte screen buffer
03d9-03ea   Buffer space
03eb-03fc   Unused
03fd-03ff   D32 - Unused in Dragon 32
03fd:03fe   D64 - Printer end of line delay in milliseconds (0x0000)
03ff        D64 - %PRNSEL% selects default printer port
                0x00 Parallel, non-0x00 Serial (0x00)
0400-05ff   Default Text screen
0600-1dff   Available graphics pages w/o DOS
0600-0bff   DOS - workspace area see also $00ea-$00f6
0600-0dff   CoCo DOS workspace area (no more info)
0c00-23ff   DOS - Available graphics pages
8000-bfff   BASIC ROM in 32K mode
8000-9fff   CoCo - Extended Color BASIC ROM
a000-bfff   CoCo - Color BASIC ROM
bff0-bfff   These addresses mapped from ROM to $fff0-$ffff by the SAM
c000-dfff   DOS - Dos ROM
c000-feff   DOS - Cumana DOS ROM only
c000-feff   Available address range to cartridge expansion port 32K mode
c000-feff   D64 - 64K mode - copy of BASIC ROM 2 exists in RAM here
```

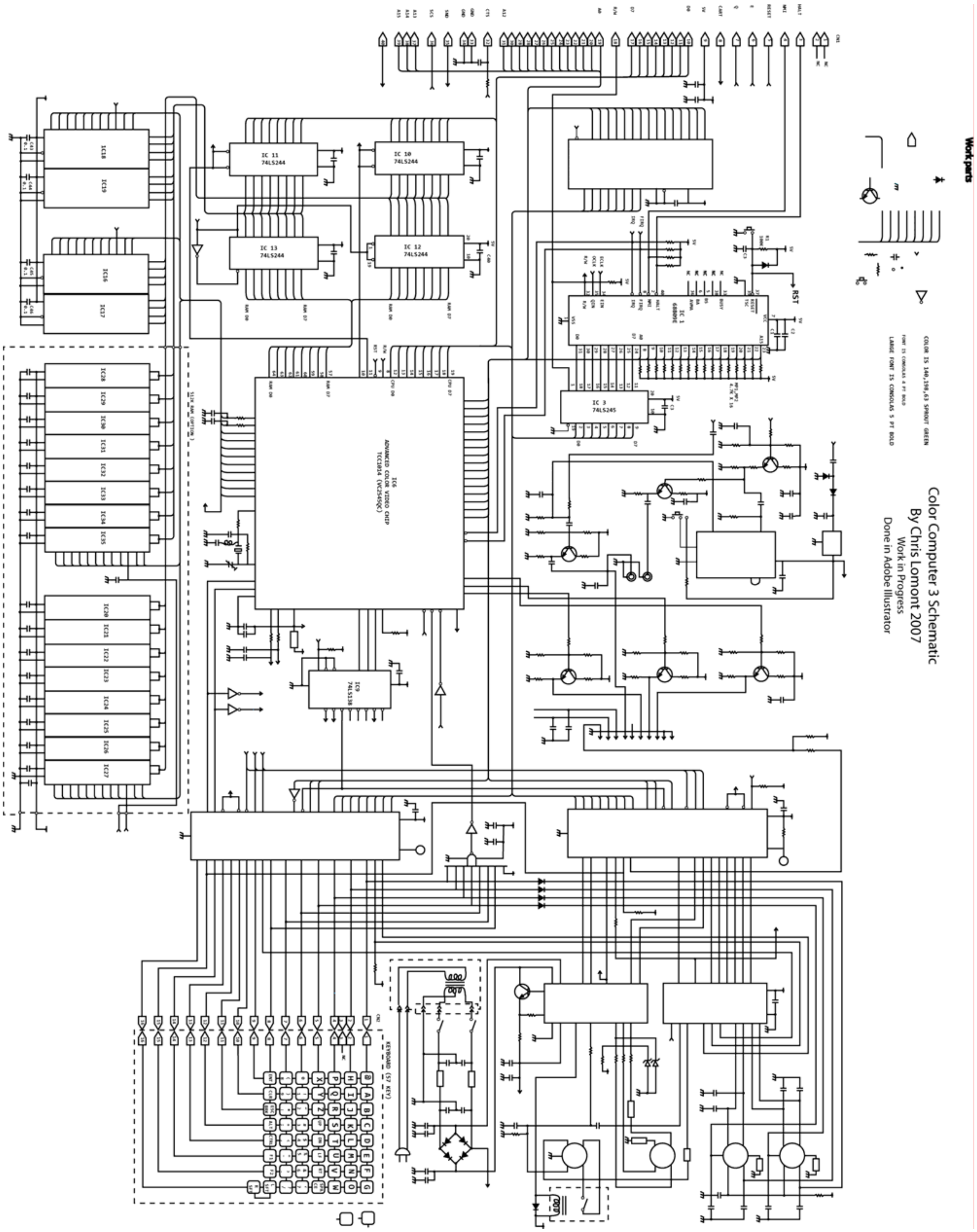
```
ff00        PIA 0 A side Data reg.
ff01        PIA 0 A side Control reg.
ff02        PIA 0 B side Data reg.
ff03        PIA 0 B side Control reg.
ff04        D64 - ACIA serial port read/write data reg.
ff05        D64 - ACIA serial port status (R)/ reset (W) reg.
ff06        D64 - ACIA serial port command reg.
ff07        D64 - ACIA serial port control reg.
ff20        PIA 1 A side Data reg.
ff21        PIA 1 A side Control reg.
ff22        PIA 1 B side Data reg.
ff23        PIA 1 B side Control reg.
ff40        DOS - Disk Controller command/status reg.
ff41        DOS - Disk Controller track reg.
ff42        DOS - Disk Controller sector reg.
ff43        DOS - Disk Controller data reg.
ff48        DOS - Disk Controller hardware control reg.
ffc0-ffdf   SAM (Synchronous Address Multiplexer) register bits - use even
                address to clear, odd address to set
ffc0-ffc5   SAM VDG Mode registers V0-V2
ffc0/ffc1   SAM VDG Reg V0
ffc2/ffc3   SAM VDG Reg V1
ffc3/ffc5   SAM VDG Reg V2
ffc6-ffd3   SAM Display offset in 512 byte pages F0-F6
ffc6/ffc7   SAM Display Offset bit F0
ffc8/ffc9   SAM Display Offset bit F1
ffca/ffcb   SAM Display Offset bit F2
ffcc/ffcd   SAM Display Offset bit F3
ffce/ffcf   SAM Display Offset bit F4
ffd0/ffc1   SAM Display Offset bit F5
ffd2/ffc3   SAM Display Offset bit F6
ffd4/ffd5   SAM Page #1 bit - in D64 maps upper 32K Ram to $0000 to $7fff
ffd6-ffd9   SAM MPU Rate R0-R1
ffd6/ffd7   SAM MPU Rate bit R0
ffd8/ffd9   SAM MPU Rate bit R1
ffda-ffdd   SAM Memory Size select M0-M1
ffda/ffdb   SAM Memory Size select bit M0
ffdc/ffdd   SAM Memory Size select bit M1
ffde/ffdf   SAM Map Type - in D64 switches in upper 32K RAM $8000-$feff
ffec-ffef   PC-Dragon - Used by Burgin's emulator to provide enhanced services
fff0-ffff   6809 interrupt vectors mapped from $bff0-$bfff by SAM
fff0:fff1   Reserved ($0000; D64 64K mode 0x3634 '64')
fff2:fff3   SWI3     ($0100)
fff4:fff5   SWI2     ($0103)
fff6:fff7   FIRQ     ($010f)
fff8:fff9   IRQ      ($010c)
fffa:fffb   SWI      ($0106)
fffc:fffd   NMI      ($0109)
fffe:ffff   RESET    ($b3b4; D64 64K mode $c000 - never accessed)
```

TODO – other memory maps, the disk stuff mismatches earlier stuff, change all addresses to $ hex and uppercase.

Schematics

Here are two CoCo 3 schematics – one PAL and one NTSC. One is still being drawn by me and is not quite finished, but between the two you can get a lot of information about the CoCo 3.


SCHEMATIC DIAGRAM
Cat. No. 26-3



TODO – insert two kinds here?! Explain types?

Bibliography

1. Self experimentation :)
2. Sockmaster's webpage (John Kowalski, http://www.axess.com/twilight/sock/)
3. Notes from Kevin K. Darling, help from Greg Law, Dennis W., and Marsha.
4. Notes from Mike Pepe.
5. Notes from Graham E. Kinns.
6. PC CoCo Emulator (coco2-13.zip) by Jeff Vavasour.
7. "The Dragon Notebook", Ray Smith, NDUG.
8. "Inside the Dragon", Duncan Smeed & Ian Sommerville, Addison-Wesley,1983.
9. "TRS-80 Color Computer Tech Ref Manual", Tandy Corp, 1981.
10. WD2797 Floppy Disc Driver Controller Data Sheet (RS #6991).
11. Dragon Disc Controller Circuit Diagram, ex Dragon Data Ltd, now NDUG.
12. Dragon 32/64 Upgrade Manual, R. Hall, NDUG, 1985.
13. "Inside the 32", Dave Barnish, p13, Jan 1987.
14. "BREAKing the '64", Martyn Armitage, p8-9, Feb 1988.
15. "Firmware - Part 1", Brian Cadge, p19, Sep 1985.
16. "Dragon Answers", Brian Cadge, p31, Sep 1985.
17. Assembly Language Graphics for the TRS-80 Color Computer, Don & Kurt Inman, 1983, Reston Publishing Company, ISBN 0-8359-0318-4.
18. TRS-80 Color Computer Assembly Language, William Barden, Jr., Radio Shack, 1983.
19. "What's Inside Radio Shack's Color Computer?", Byte Magazine, 1981, http://www.byte.com/art/9603/sec5/art4.htm
20. Notes from http://www.cs.unc.edu/~yakowenk/coco.html
21. "Assembly Language Programming for the Color Computer", Laurence A Tepolt, Tepco, 1985.
22. "Assembly Language Programming for the CoCo3", Laurence A Tepolt, Tepco, 1987.
23. The Unraveled series: "Color Basic Unraveled II", "Extended Basic Unraveled II", "Super Extended Basic Unraveled II", "Disk Basic Unraveled II", Walter K. Zydhek, Spectral Associates, 1999.
24. Info from http://www.trs-80.com/

Glossary

DOS – Disk Operating System
RS-DOS – Radio Shack Disk Operating System
TODO – Need glossary

Index

G

TODO – index all

TODO

A list of things to do in future versions
TODO

- Crosslink many more items
- Finish glossary, index
- Add schematics, perhaps part spec sheets?
- Major check on consistent layout, etc
- Check my asm and CoCo books for more info
- Need lots of content filled in, verified, corrected.
- See how prints, make 1, 2, and 4 page versions
- Final proof pass

END OF FILE