CoCo 3

The Tandy Color Computer 3 Technical Field Guide

Hardware, MC6809E, GIME, MMU, Super Extended BASIC, Disk BASIC, graphics, sound, I/O, storage, assembly language and practical debugging

ROM/BASIC programming edition

Version 1.0 - September 2026

Scope This guide is deliberately centered on the stock CoCo 3 hardware, Color/Extended/Super Extended BASIC, Disk Extended Color BASIC, ROM internals and 6809E machine-language programming. OS-9 and NitrOS-9 are intentionally outside the scope of this edition.

A shareable technical reference compiled from Tandy and Motorola documentation, ROM-source reconstruction, and established CoCo hardware research.

How to use this guide

The Color Computer 3 is unusually compact from a hardware point of view and unusually deep from a programmer's point of view. A 6809E still sees a 64K logical address space, yet the GIME can map a much larger physical RAM space, generate several families of video modes, provide palette registers and scrolling controls, and add timer, keyboard, cartridge and raster-related interrupt sources. At the same time the machine keeps a large part of the original CoCo programming model alive. The result is a computer where a short BASIC program, a ROM hook, a carefully timed machine-language routine and an MMU-based memory manager can all be manipulating different layers of the same system.

This document is written as a field guide rather than a tutorial that assumes one particular project. Early chapters establish the architecture. The middle chapters document the programmable devices and ROM/BASIC environment. Later chapters concentrate on techniques, failure modes, testing and quick-reference tables. The appendices are intended to be kept open beside an assembler, debugger or real CoCo 3.

Evidence labels OFFICIAL means a behavior is documented by Tandy or Motorola. ROM means it is supported by the reconstructed/disassembled CoCo ROM sources. COMMUNITY means the behavior comes from long-standing independent GIME/hardware research and should be verified on the target machine when timing or undocumented behavior matters.

Hexadecimal values are written with a dollar sign in assembly-oriented prose ($FF90) and sometimes with BASIC notation when an example is explicitly BASIC (&HFF90). Binary bit numbers use bit 7 for the most significant bit and bit 0 for the least significant bit. Unless stated otherwise, addresses such as $8000 are CPU-visible logical addresses. Physical addresses above $FFFF refer to the GIME/MMU physical RAM map.

Contents

1. The machine in one view

Radio Shack/Tandy introduced the Color Computer 3 in 1986 as the third major generation of the 6809-based Color Computer family. It preserved a very high degree of compatibility with CoCo 1/2 software while replacing much of the earlier video, address-decode and memory-control logic with a custom multifunction chip. Tandy service literature calls this device the ACVC; the CoCo community almost universally calls it the GIME. The GIME is the defining component of the CoCo 3.

Component	Stock CoCo 3
CPU	Motorola MC68B09E / 6809E-family 8-bit CPU; externally clocked; normally about 0.895 MHz or about 1.79 MHz.
RAM	128K standard on common retail systems; designed for an official 512K expansion. The GIME MMU natively describes 64 physical 8K pages = 512K.
ROM/BASIC	32K resident BASIC environment: Color BASIC 2.0, Extended Color BASIC 2.0, and Super Extended BASIC 2.0; Disk BASIC normally comes from the disk-controller cartridge.
Video	Legacy CoCo-compatible modes plus GIME-native text and graphics, up to 640 horizontal pixels; 64-entry RGB/composite color code space with up to 16 palette entries active at once in ordinary modes.
Text	Legacy 32x16 plus high-resolution text such as 40x24 and 80x24 under BASIC; GIME hardware supports additional character-width combinations.
Graphics	GIME native families using 2, 4 or 16 colors per pixel group; BASIC exposes HSCREEN 1-4, while direct GIME programming exposes more combinations and 192/200/225-line timing choices.
Audio	No synthesizer chip. Six-bit DAC output through the PIA plus a 1-bit sound path; software generates tones and digital audio.
Input	57-key keyboard matrix; two analog joystick ports with buttons; cassette input; cartridge and interrupt inputs.
Storage	Cassette interface built in; floppy disk via controller cartridge using Disk Extended Color BASIC; 35-track 18-sector single-sided standard format.
Video outputs	RF television, composite video/audio, and analog RGB through the bottom connector.
Expansion	40-pin cartridge bus; disk controllers, ROM paks, Multi-Pak Interface and many third-party devices.

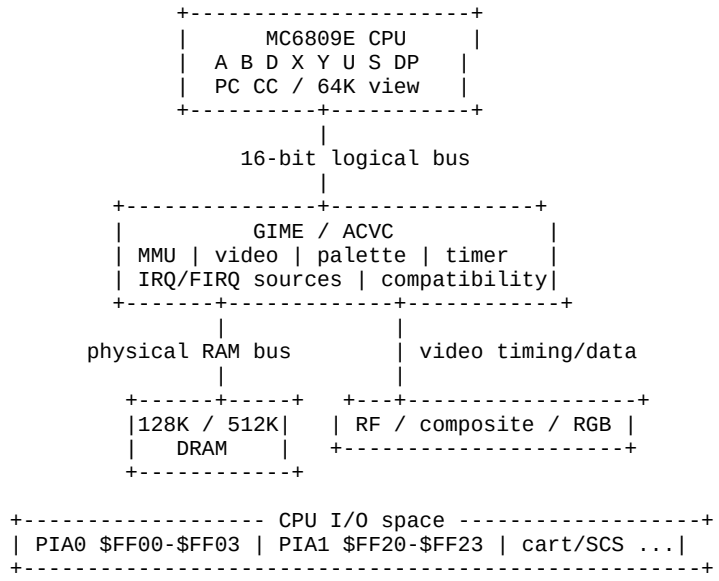
Primary references: Tandy Color Computer 3 Service Manual [S1]; CoCo 3 Extended BASIC manual [S2]; Super Extended BASIC Unravelled [S6].

1.1 Why the CoCo 3 is technically distinctive

- The CPU did not become a 16-bit-address-plus-banking processor. It remained a normal 6809E with a 16-bit address bus. The GIME remaps eight 8K logical windows behind that bus.
- There are two complete MMU task maps. Switching the task bit can replace the entire 64K logical view in a single register operation, although shared code, stack, vectors and I/O must be planned carefully.
- The video generator can fetch from physical memory independently of the CPU's current logical map. This lets the display live outside the CPU's immediately visible 64K.
- Super Extended BASIC is not a separate modern operating environment; it extends the familiar Microsoft BASIC lineage and uses the additional RAM and GIME through ROM routines, work areas and MMU mappings.
- The CoCo 3 keeps the original CoCo PIA/SAM-style programming interfaces for compatibility while also offering native GIME registers. This dual personality is the source of both its flexibility and many of its quirks.

2. System architecture

At the center is the externally clocked 6809E. The CPU provides a 16-bit address bus and an 8-bit data bus. The GIME sits between that logical bus and system memory/video timing. Two 6821-compatible PIA devices provide keyboard scanning, joystick comparison, DAC/sound, cassette and serial/printer functions. Cartridge expansion shares the CPU bus and receives chip-select and interrupt-related signals. Video data is fetched by the GIME from physical RAM according to its video-start, resolution, palette and scrolling registers.



The central mental model Always distinguish CPU logical address, physical RAM address, and video fetch address. Many CoCo 3 bugs are simply one of those three being mistaken for another.

3. The MC6809E programmer model

The 6809 is unusually pleasant for an 8-bit CPU because it was designed with high-level language support, orthogonal addressing, two stack pointers and extensive indexed addressing. The CoCo 3 uses the E-clock version, meaning the system supplies the CPU timing externally. The CPU is big-endian: for a 16-bit word, the most significant byte is stored at the lower address.

Register	Width	Purpose
A	8	Accumulator A.
B	8	Accumulator B.
D	16	A:B combined as a 16-bit accumulator; A is the high byte.
X	16	Index register; heavily used for memory traversal and addressing.
Y	16	Second index register.
U	16	User stack / general stack pointer; useful for private stacks, parameter frames and fast block techniques.
S	16	System hardware stack; interrupts and subroutine conventions commonly use it.
PC	16	Program counter.
DP	8	Direct Page register; supplies the high byte for direct addressing.
CC	8	Condition Code register: E F H I N Z V C.

3.1 Condition-code register

Bit	Name	Meaning
7	E	Entire state stacked. Set in full interrupt frames.
6	F	FIRQ mask.
5	H	Half carry, primarily useful for BCD adjustment.
4	I	IRQ mask.
3	N	Negative.
2	Z	Zero.
1	V	Signed overflow.
0	C	Carry/borrow.

3.2 Addressing modes that matter on the CoCo 3

- Direct addressing uses DP as the high address byte. Setting DP to a hot data page can save both code bytes and cycles in inner loops.
- Extended addressing supplies a full 16-bit address and is the straightforward way to touch hardware registers such as $FF90.

- Indexed addressing on X, Y, U and S is a major 6809 strength: constant offsets, accumulator offsets, auto-increment/decrement, PC-relative forms and indirect forms support compact table walkers and blitters.
- Relative branches make relocatable local control flow practical. Long branches are available when the target is farther away.
- PSHS/PULS and PSHU/PULU can transfer multiple registers in one instruction and are central to interrupt handlers and calling conventions.

3.3 IRQ, FIRQ, NMI and software interrupts

IRQ is the general maskable interrupt and normally causes a full machine state to be stacked. FIRQ is optimized for speed: in its fast form the CPU stacks only the condition code and program counter, leaving the handler responsible for preserving any additional registers it destroys. NMI is non-maskable once the CPU is armed after reset. SWI, SWI2 and SWI3 provide software interrupt vectors. On the CoCo 3, the fixed vectors at the top of the address space lead into ROM/RAM vector stubs, giving software practical hook points without rewriting the immutable vector addresses themselves.

Vector	Address
SWI3	$FFF2
SWI2	$FFF4
FIRQ	$FFF6
IRQ	$FFF8
SWI	$FFFA
NMI	$FFFC
RESET	$FFFE

Primary references: Motorola MC6809/MC6809E Programming Manual [S5]; Sock Master GIME reference [S9].

4. Clocking and performance

The CoCo 3 preserves the classic CoCo slow rate and adds a faster CPU rate. Tandy documentation specifies approximately 0.89 MHz and 1.78 MHz. Software traditionally selects speed through the compatibility speed-control addresses in the $FFD6-$FFD9 area. In common practice, writing to $FFD8 selects the slow rate and writing to $FFD9 selects the fast rate. The value written is not the point; the write strobe is.

Timing warning Changing CPU speed changes software delay loops, bit-banged serial timing, sampled audio timing and any routine synchronized by instruction count. Video timing itself is not simply doubled. Use hardware timers or raster synchronization when wall-clock timing matters.

The fast rate is normally desirable for computation, rendering, decompression and disk-side processing that tolerates it. Code that talks to old cartridges or relies on marginal bus timing may need the slow rate. A robust program explicitly establishes the speed it requires instead of assuming the previous program left the expected state.

5. Memory: logical versus physical

The 6809 can issue only 65,536 logical addresses at a time. The GIME makes more RAM useful by dividing the CPU address space into eight logical blocks of 8K and mapping each block to a selected physical 8K page. A 512K system therefore has 64 physical pages numbered $00-$3F. A 128K system physically contains the upper 16 pages, $30-$3F, corresponding to physical addresses $60000-$7FFFF.

CPU logical block	Task 0 register	Task 1 register
$0000-$1FFF	$FFA0	$FFA8
$2000-$3FFF	$FFA1	$FFA9
$4000-$5FFF	$FFA2	$FFAA
$6000-$7FFF	$FFA3	$FFAB
$8000-$9FFF	$FFA4	$FFAC
$A000-$BFFF	$FFA5	$FFAD
$C000-$DFFF	$FFA6	$FFAE
$E000-$FFFF	$FFA7	$FFAF

Each MMU register holds a six-bit page selector on stock hardware. The two maps are selected by the task bit in GIME INIT1 ($FF91 bit 0). MMU translation must also be enabled in INIT0 ($FF90 bit 6). The I/O region remains decoded as I/O rather than ordinary RAM, and ROM/constant-vector controls complicate the very top of the address space when those features are enabled.

5.1 Physical page math

```
physical_page = physical_address >> 13
page_offset   = physical_address & $1FFF
physical      = physical_page * $2000 + page_offset

Example: physical $6C123
  page   = $36
  offset = $0123
To view it at CPU $2000-$3FFF, map page $36 into $FFA1 or $FFA9,
then access logical $2123.
```

5.2 The stock 128K physical layout used by BASIC

Page	Physical range	Typical Super Extended BASIC use
$30	$60000-$61FFF	High-resolution graphics area
$31	$62000-$63FFF	High-resolution graphics area
$32	$64000-$65FFF	High-resolution graphics area
$33	$66000-$67FFF	High-resolution graphics area
$34	$68000-$69FFF	HGET/HPUT graphics buffer area
$35	$6A000-$6BFFF	Secondary stack/work area
$36	$6C000-$6DFFF	High-resolution 40/80-column text screen
$37	$6E000-$6FFFF	Generally unused by stock BASIC; valuable but do not assume third-party software leaves it free
$38	$70000-$71FFF	Normal 64K view / lower system-BASIC RAM
$39	$72000-$73FFF	Normal 64K view

Page	Physical range	Typical Super Extended BASIC use
$3A	$74000-$75FFF	Normal 64K view
$3B	$76000-$77FFF	Normal 64K view
$3C	$78000-$79FFF	RAM containing copied ROM image / upper normal view
$3D	$7A000-$7BFFF	RAM containing copied ROM image / upper normal view
$3E	$7C000-$7DFFF	RAM containing copied ROM image / upper normal view
$3F	$7E000-$7FFFF	RAM containing copied ROM image, vectors and I/O-adjacent space

Do not confuse installed RAM with readable page numbers On a 128K machine the real RAM occupies physical pages $30-$3F. Some mappings or hardware configurations can make lower selections appear mirrored. Mirroring is not extra storage. A memory manager must test installed memory and treat aliases as aliases.

Primary references: Assembly Language Programming for the CoCo 3 [S7]; Super Extended BASIC Unravelled [S6]; Sub-Etha MMU reference [S12].

5.3 Two MMU tasks: what they are and what they are not

Task 0 and Task 1 are two sets of eight page-map registers, not two CPUs, processes or protected address spaces. Switching the task bit in $FF91 changes which map the CPU uses. There is no privilege separation and no automatic context save. A task switch is useful only when software has deliberately arranged code, stack, data and vectors so execution remains valid after the mapping changes.

- If the currently executing code page changes underneath the CPU, the next instruction comes from the newly mapped page. That can be intentional, but usually it is fatal.
- If the S stack page changes, the next interrupt, RTS, PSHS or PULS can corrupt state immediately.
- If an interrupt can occur while a temporary mapping is active, its handler must either be mapping-independent or the routine must mask that interrupt around the transaction.
- Super Extended BASIC uses extra mappings for high-resolution graphics and other services. Programs that appropriate task registers must restore them exactly, not approximately.
- For library-quality code, save the page register you actually alter, make the smallest mapping change needed, perform the operation, then restore before reenabling asynchronous activity.

6. Power-up, ROM and the BASIC environment

The CoCo 3 contains 32K of internal BASIC ROM. Its startup sequence tests and initializes hardware, copies ROM code into the upper RAM area, applies CoCo 3 patches, establishes vectors and work areas, and enters the familiar Microsoft BASIC-derived environment. This copy-to-RAM architecture is important: after startup, much of what appears to be ROM behavior is executing from RAM that contains the ROM image. That enables patches and compatibility fixes without changing the physical ROM chip.

The buildable coco_roms project is especially useful to programmers because it reconstructs Color BASIC, Extended Color BASIC, Disk BASIC and Super Extended BASIC from the Unravelled lineage and verifies resulting ROM images. For debugging an entry point or internal variable, source-level inspection is often more reliable than a memory-map table copied through several generations of magazines.

ROM component	Logical role
Extended Color BASIC 2.0	Graphics, sound, extended statements/functions and compatibility layer inherited from the CoCo 1/2 environment.
Color BASIC 2.0	Core Microsoft BASIC interpreter, expression evaluation, variables, strings, console, cassette and common runtime services.
Super Extended BASIC 2.0	CoCo 3 additions: GIME graphics/text, palettes, enhanced error handling, long-memory access and related support.
Disk Extended Color BASIC 1.x	Normally supplied by disk-controller ROM and integrated at startup to add files, drives, sector I/O and disk commands.

ROM routine rule Do not call an undocumented ROM address merely because one program found it there. First determine the exact ROM/BASIC version and whether the address is a published entry point, a stable vector, or an implementation detail. CoCo 3 startup patching means the most useful call target may be a RAM vector rather than the original ROM address.

Primary references: Super Extended BASIC Unravelled [S6]; buildable CoCo ROM source [S10].

7. The GIME / ACVC

The GIME consolidates and extends functions that earlier Color Computers distributed across the SAM, VDG and address-control logic. It provides native video generation, palette selection, horizontal and vertical display controls, memory mapping, two selectable MMU maps, a programmable timer, additional interrupt sources and compatibility controls. It does not replace the PIAs; keyboard, DAC, joysticks and several low-level I/O functions remain PIA-driven.

7.1 GIME register blocks

Range	Function
$FF90-$FF97	Initialization, IRQ/FIRQ enables/status, timer, reserved registers
$FF98-$FF9F	Video mode, resolution, border, vertical scroll, display start, horizontal offset/virtual mode
$FFA0-$FFA7	MMU map / task 0
$FFA8-$FFAF	MMU map / task 1
$FFB0-$FFBF	16 palette registers
$FFC0-$FFDF	Legacy SAM-compatible write strobes for video/address/speed/ROM-RAM behavior
$FFE0-$FFFF	Reserved/vector region; CPU vectors at $FFF2-$FFFE

Real-hardware readback hazard Several GIME control registers should be treated as write-only unless a specific source documents reliable readback. Real CoCo 3 hardware can return open-bus/bleed-through values where an emulator may appear to echo the last write. Never build a read-modify-write sequence around a write-only GIME control register. Keep software shadow bytes.

8. GIME initialization and interrupt registers

8.1 $FF90 - INIT0

Bit	Name	Function
7	COCO	1 selects CoCo 1/2-compatible video interpretation; 0 selects native GIME mode.
6	MMUEN	1 enables GIME MMU translation.
5	IEN	Master enable for GIME-generated IRQ output.
4	FEN	Master enable for GIME-generated FIRQ output.
3	MC3	Constant FExx RAM / secondary-vector behavior.
2	MC2	Standard SCS behavior.
1-0	MC1:MC0	ROM map control: internal/external ROM mapping combinations.

A common native-graphics BASIC state uses $44 in INIT0: MMU enabled, standard SCS behavior, native GIME video, master GIME IRQ/FIRQ outputs disabled. That value is not a universal initialization constant. A standalone program should construct the required bit mask from the features it owns and should preserve any environment state it promises to restore.

8.2 $FF91 - INIT1

Bit	Function
7	Unused/reserved on stock CoCo 3.
6	Memory-configuration-related behavior is described by community references; do not depend on it for portable stock code without verification.
5	TINS: timer clock source. Selects the fast or slow GIME timer source.
4-1	Unused/reserved.
0	TR: selects MMU task register set 0 ($FFA0-$FFA7) or 1 ($FFA8-$FFAF).

8.3 $FF92 and $FF93 - IRQ and FIRQ sources

Bit	Source
5	GIME timer
4	Horizontal border event
3	Vertical border event
2	Serial-related event
1	Keyboard event
0	Cartridge event

The IRQ and FIRQ source registers are both enable/status mechanisms. A robust handler identifies and acknowledges the source according to the hardware semantics rather than assuming every interrupt came from the timer. If multiple sources can be enabled at once, service them deterministically and keep the handler short.

8.4 $FF94-$FF95 - 12-bit interval timer

The GIME supplies a 12-bit countdown timer. The high register contributes the upper four bits and the low register the lower eight. Writing the timer control/count registers starts or reloads the timer according to the documented sequence. INIT1 TINS selects a relatively fast source or a much slower source, making the timer useful both for high-rate service routines and longer intervals. Community investigation has documented small revision-dependent count differences, so cycle-critical code should measure the exact target GIME rather than baking in an emulator-derived correction.

Primary references: Tandy Color Computer 3 Service Manual [S1]; Sock Master GIME reference [S9].

9. Native video control: $FF98-$FF9F

9.1 $FF98 - VMODE

Bits	Function
7	Graphics/text selector for native GIME display.
6	Reserved in the normal stock programming model.
5	Burst-phase control / phase inversion function.
4	Monochrome/composite-related control.
3	50/60 Hz timing selection.
2-0	Lines-per-character-row selection in text-oriented operation.

9.2 $FF99 - VRES

VRES combines vertical line count, horizontal bytes per row and color depth/attribute interpretation. This is why it is better to think of a GIME mode as a packing format plus timing parameters rather than a single enumerated mode number.

Field	Meaning
bits 6-5 (LPF)	Lines per field: documented native choices include 192, 200 and 225-line operation, with one encoding reserved/undefined.
bits 4-2 (HRES)	Bytes fetched per display row. Encodings span 16, 20, 32, 40, 64, 80, 128 and 160 bytes per row.
bits 1-0 (CRES)	Graphics color depth (2, 4 or 16 colors) or text attribute interpretation depending on VMODE. One combination is reserved/undefined.

9.3 $FF9A - border color

In native modes, $FF9A selects the border color using the same six-bit color-code space as the palette hardware. This is a genuine programmable border, unlike the more constrained legacy behavior.

9.4 $FF9C - vertical scroll

The low four bits provide vertical smooth-scroll control. Combined with a display-start update when the fine scroll wraps, this makes hardware-assisted vertical scrolling possible without moving the entire screen bitmap each frame.

9.5 $FF9D-$FF9E - display start

The native GIME display-start registers select the physical base address from which video fetch begins. The encoding operates in small address units rather than storing a literal 19-bit byte address; the commonly documented native relationship gives 8-byte start granularity. This lets the video image live outside the CPU's current MMU view and supports page flipping and coarse scrolling.

9.6 $FF9F - horizontal offset and virtual width

$FF9F supplies horizontal offset control and a horizontal virtual-enable bit. Horizontal offset advances in two-byte units. When horizontal virtual mode is enabled, the logical row stride becomes 256 bytes, allowing the visible window to slide horizontally within a wider virtual surface. This is one of the most powerful stock GIME features for smooth scrolling engines.

Reserved registers Do not treat $FF96, $FF97 or $FF9B as general-purpose storage. Stock documentation reserves them. Some memory expansions assign new meanings to otherwise unused register bits/addresses, but that is expansion-specific behavior, not stock CoCo 3 behavior.

Primary references: Tandy Color Computer 3 Service Manual [S1]; Sock Master GIME reference [S9]; Lomont hardware reference [S8].

10. Palette and color system

The GIME provides sixteen palette registers at $FFB0-$FFBF. A screen pixel stores a palette index; the palette register then supplies a six-bit hardware color code. Thus a 16-color graphics mode can choose any 16 entries from the machine's 64-code color space, and a 4-color mode can select four of the 16 currently defined palette entries according to the pixel packing and mode rules.

Register range	Role
$FFB0-$FFBF	Palette entries 0-15; low six bits select the hardware color code. Upper bits are not part of the stock six-bit color value.

10.1 RGB encoding

For the RGB output, the six color bits represent two bits each of red, green and blue. A useful way to read the bit order is R1 G1 B1 R0 G0 B0. Each component therefore has four intensity levels, giving 4 x 4 x 4 = 64 RGB codes. Composite output is generated differently and the visible hue/intensity relationship is not identical to RGB.

10.2 Palette programming pattern

```
; Example: place hardware color code $3F in palette entry 1.
; Preserve/choose values according to the target display; $3F is just an example.
        LDA   #$3F
        STA   $FFB1
```

Composite is not RGB A six-bit value that looks ideal on RGB may not look identical on NTSC composite, and PAL machines add another conversion path. If color fidelity matters, test the palette on the actual output type being targeted.

10.3 Artifact color

NTSC composite can create apparent colors from high-frequency monochrome pixel patterns through color-subcarrier artifacting. CoCo software famously exploited this in legacy high-resolution modes, and sophisticated CoCo 3 techniques can extend the idea. These artifact colors are an analog display effect, not extra digital palette entries. Claims of a conventional hidden 256-color stock GIME register mode should therefore be separated from techniques that produce many apparent colors through composite artifacting or scanline palette changes.

11. Text modes

The CoCo 3 can operate in the original 32-column style and in GIME high-resolution text. Super Extended BASIC commonly exposes 40x24 and 80x24 displays through WIDTH, while direct hardware programming can select other byte-row/character-width combinations. In attribute text, each screen cell consumes two bytes: character code followed by an attribute byte.

Attribute bit	Meaning
7	Blink/flash
6	Underline
5-3	Foreground palette selector (selects among the foreground palette group)
2-0	Background palette selector (selects among the background palette group)

A practical consequence is that an 80-column row with attributes consumes 160 bytes. Twenty-four rows require 3,840 bytes, comfortably inside one 8K page. BASIC conventionally places its high-resolution text screen in physical page $36. Machine-language software can map that page into a convenient logical window, edit characters/attributes directly, and then restore the previous mapping.

Do not assume WIDTH is cosmetic WIDTH changes the display subsystem, work areas and sometimes memory mappings. Code that borrows the high-resolution text page or task registers must expect WIDTH, HSCREEN and related ROM services to reconfigure them.

12. Graphics modes and framebuffer math

The GIME graphics engine is best understood by three values: bytes per row, pixels packed per byte, and lines per field. In a 2-color mode one bit per pixel gives 8 pixels per byte. In a 4-color mode two bits per pixel gives 4 pixels per byte. In a 16-color mode four bits per pixel gives 2 pixels per byte. Multiplying the programmed bytes-per-row by pixels-per-byte gives the horizontal pixel count.

Bytes/row	2 colors (8 px/B)	4 colors (4 px/B)	16 colors (2 px/B)
16	128	64	32
20	160	80	40
32	256	128	64
40	320	160	80
64	512	256	128
80	640	320	160
128	1024*	512	256
160	1280*	640	320

Asterisked combinations are useful for understanding the register math but are not necessarily practical as a normal visible raster at the standard timing/window. The service-manual and community mode tables identify the combinations intended for the display generator. Common native graphics include 320x192 with 16 colors, 640x192 with 4 colors, and related 2/4/16-color modes at 128/160/256/320/512/640 horizontal resolutions. Vertical choices include 192, 200 and 225 lines at the hardware level.

12.1 BASIC HSCREEN modes

BASIC mode	Resolution / color depth
HSCREEN 1	320x192, 4 colors
HSCREEN 2	320x192, 16 colors
HSCREEN 3	640x192, 2 colors
HSCREEN 4	640x192, 4 colors
HSCREEN 0	Return to text display

Super Extended BASIC intentionally presents a smaller, manageable subset of what the GIME hardware can do directly. Code that requires 225-line screens, unusual widths, virtual-width scrolling or custom row strides generally programs the GIME itself.

12.2 Screen memory calculation

```
frame_bytes = bytes_per_row * visible_lines

Example: 320 x 192 x 16 colors
  2 pixels per byte -> 160 bytes/row
  160 * 192 = 30,720 bytes

Example: 640 x 192 x 4 colors
  4 pixels per byte -> 160 bytes/row
  160 * 192 = 30,720 bytes
```

This is why several of the headline GIME modes fit into roughly 30K: they trade color depth for horizontal resolution while keeping the same 160-byte row width.

13. Scrolling, page flipping and video memory

Because the GIME fetches video from a physical start address, scrolling does not require the CPU to copy an entire framebuffer for every movement. Vertical scrolling can combine the fine-scroll nibble in $FF9C with display-start updates. Horizontal virtual mode can treat each logical raster line as 256 bytes and move the visible window using the horizontal offset. Page flipping is performed by preparing another framebuffer in RAM and changing the video-start registers during a safe display interval.

- Use display-start changes for coarse movement in memory.
- Use $FF9C for fine vertical motion between coarse line steps.
- Use $FF9F horizontal offset with virtual-width mode for fine/coarse horizontal viewport movement.
- Write palette, video-start and mode registers during vertical blank or another known-safe interval when visible tearing or color bars matter.
- Keep the CPU's MMU mapping conceptually independent from the video start: the CPU can edit a page by mapping it somewhere convenient even while the video generator reads it from its physical location.

Framebuffer ownership BASIC owns its HSCREEN memory while BASIC graphics commands are in use. A machine-language program can borrow or repurpose those pages, but only if it also owns the transition back. Otherwise the next BASIC graphics operation may overwrite the data or remap the page.

14. Interrupts, raster timing and the GIME timer

The GIME can signal horizontal border, vertical border, timer, keyboard, serial and cartridge events through IRQ or FIRQ. This enables animation and audio systems that do not spend all their time polling. The design choice is not simply IRQ versus FIRQ: it includes which sources are enabled, whether the handler is safe under temporary MMU mappings, and how long the handler can run before it disturbs video or audio timing.

14.1 FIRQ design pattern

```
; Skeleton only - source acknowledgement and saved registers depend on the use case.
FIRQ_HANDLER:
        PSHS  A,B,X          ; FIRQ did not save these for us
        ; identify/acknowledge enabled GIME source
        ; perform bounded real-time work
        PULS  A,B,X
        RTI
```

A FIRQ handler should do the minimum time-critical work and return. Buffer management, decompression and other long operations belong in foreground code. If a handler touches an MMU register, either dedicate that mapping to the interrupt subsystem or establish and restore it atomically.

14.2 Raster effects

Horizontal/vertical border events can be used to synchronize register changes with the raster. Palette splits, split screens and other beam-raced effects are possible, but timing margins differ among RGB/composite paths, CPU speeds, GIME revisions and attached hardware. Such effects should be measured on real target hardware. Code that works in an emulator with exact deterministic scheduling can fail by a few cycles on a physical system.

15. Legacy CoCo compatibility

The CoCo 3 is deliberately bilingual: it can behave enough like a CoCo 1/2 for a large software library while also exposing its native GIME model. The legacy interface includes PIA behavior, SAM-style write addresses for video/control, old VDG-style modes and the traditional 32-column text environment. Compatibility is excellent but not perfect because the GIME emulates functions formerly performed by separate chips.

Write-address family	Legacy role
$FFC0-$FFC5	Legacy video mode control bits
$FFC6-$FFD3	Legacy display-offset/address controls
$FFD6-$FFD9	CPU speed control family
$FFDE-$FFDF	ROM/RAM mapping control family

A write-only address pair convention is inherited from the SAM: one address clears a bit and its partner sets the bit. Because these are strobes rather than ordinary storage locations, reading them is not a meaningful way to discover the current state. Maintain software state or reinitialize explicitly.

Not every obscure VDG semigraphics mode is faithfully reproduced. Programs that depend on undocumented analog details of the original VDG/SAM combination should be tested on a CoCo 3 rather than assuming compatibility from the major modes alone.

16. Super Extended BASIC

Super Extended BASIC extends the familiar command-line and program environment rather than replacing it. Its purpose is to make the CoCo 3 hardware usable without requiring every owner to become a GIME programmer. The most visible additions cover 40/80-column text, palettes, native high-resolution graphics, long-memory access and structured error/break handling.

Area	Representative CoCo 3 additions
Display	WIDTH, high-resolution text control, LOCATE, attributes and palette-related commands.
Graphics	HSCREEN, HCLS, HCOLOR, HLINE, HPAINT, HCIRCLE, HDRAW, HSET, HRESET, HPOINT, HGET, HPUT, HBUFF and related high-resolution operations.
Long memory	LPEEK and LPOKE permit BASIC to address physical memory beyond its ordinary 16-bit workspace.
Control/error handling	ON BRK GOTO, ON ERR GOTO and error-information variables such as ERLIN/ERNO.
Presentation	HPRINT and related high-resolution screen functions.

The exact syntax and side effects belong to the official CoCo 3 Extended BASIC manual. The programming lesson is that these commands are not thin wrappers around a framebuffer only: some reserve buffers, manipulate MMU mappings, switch text/graphics subsystems and alter system work areas. A hybrid BASIC/machine-language application should define which side owns each hardware resource.

16.1 LPEEK/LPOKE versus MMU mapping

LPEEK/LPOKE are convenient for occasional physical-memory access from BASIC. For high-throughput operations such as image conversion, sound buffering or bulk copies, it is normally faster for machine language to map the desired 8K physical page into a known CPU window and use ordinary indexed loads/stores. The safe pattern is map - operate - restore, with interrupts controlled if an asynchronous routine expects the old map.

17. Calling machine language from BASIC

A common CoCo style is to let BASIC handle menus, filenames and orchestration while machine language handles the inner loops. EXEC transfers control to a machine-language address; USR functions provide a mechanism for BASIC expressions to invoke machine code; ROM vectors can be called for console or interpreter services when their contracts are understood. The boundary must preserve the interpreter state and any registers/memory locations the called routine is not entitled to destroy.

- Choose a load address that does not collide with BASIC program text, variables, strings, graphics buffers, disk work areas or high-resolution screens.
- If the routine changes DP, S, interrupt masks, CPU speed, GIME mode, palette, MMU registers or PIA directions, either document that ownership or restore the state before returning.
- Do not leave the CPU executing from a page that you then remap.
- If BASIC can trigger BREAK or an error while hardware state is temporarily altered, install a cleanup path or mask the condition during the critical section.
- Use a small stable call gate when a larger body of code is banked. The gate stays in a fixed page, maps the bank, calls it, restores the page, then returns.

18. Keyboard hardware

The keyboard is a scanned matrix connected to the PIAs. Software drives keyboard column lines and reads row lines to discover pressed keys. The CoCo 3 keyboard adds keys such as ALT, CTRL, F1 and F2 while retaining the familiar arrows, BREAK/ESC, CLEAR and two SHIFT keys. BASIC and ROM routines translate the matrix into character/key codes, but low-level programs can scan the matrix directly for games, chords and non-character controls.

PIA register area	Keyboard role
$FF00-$FF03	PIA associated with keyboard matrix and joystick comparator/button functions; individual data-direction/control modes determine the exact interpretation.
$FF02 data side	Keyboard column drive lines are conventionally associated with this side of the matrix.
$FF00 data side	Keyboard row sense lines share this side with joystick/button-related functions.

PIA caution A 6821 port is not just a byte at an address. Its control register selects whether the paired address accesses a data register or a data-direction register. Blindly writing a PIA port can change pin direction or disturb another subsystem. Preserve and understand the control state.

For user interfaces, ROM/BASIC keyboard routines are convenient because they honor the current translation environment. For games or diagnostic software, matrix scanning avoids translation latency and allows simultaneous-key detection, but the routine must avoid fighting joystick/DAC scanning that uses shared PIA resources.

19. Joysticks, mouse and analog conversion

The two CoCo joystick ports are analog rather than digital direction switches. Each stick provides potentiometer positions plus a button. The machine does not have a separate general-purpose ADC chip; software uses the six-bit DAC and an analog comparator path to determine where the joystick voltage crosses the DAC reference. This is why joystick reading interacts with sound/DAC use.

A typical algorithm selects one joystick axis, drives a DAC value, samples the comparator, and searches for the transition point. BASIC hides this behind JOYSTK-style functions. Machine language can trade speed, noise tolerance and resolution differently. Because analog component tolerances and joystick potentiometers are real-world devices, emulator-perfect values should not be treated as hardware truth.

Shared-resource problem Digital audio that continually writes the DAC and joystick/mouse sampling both want the same analog subsystem. Schedule them deliberately. One common design briefly suspends DAC audio updates for a bounded joystick sample, or samples input at known positions in the audio cycle.

20. Sound

The stock CoCo 3 has no programmable sound generator comparable to a SID, AY-3-8910 or Yamaha FM chip. Its flexibility comes from software. The PIA exposes a six-bit DAC path and a one-bit sound control path, and the analog mux can route sources such as cartridge sound. BASIC SOUND and PLAY synthesize tones in software; machine-language programs can drive the DAC at audio rates for speech, samples and custom waveforms.

Register / path	Audio relevance
PIA around $FF20	Upper six data bits form the classic 6-bit DAC output path.
PIA around $FF22	Includes a 1-bit sound-related output and additional mux/control signals.
Cartridge SND signal	Expansion cartridges can inject an analog/audio signal into the CoCo audio path.

20.1 Digital audio

Six-bit unsigned PCM gives 64 amplitude levels. The practical quality depends strongly on sample rate, reconstruction filtering, volume headroom and timing jitter. An 11 kHz sample consumes about 11,000 output updates per second; at 1.79 MHz that leaves only roughly 163 CPU cycles per sample before considering interrupt and display overhead. Efficient players therefore use tight loops, timer-driven buffering, preprocessing or lower sample rates.

20.2 Audio engineering tips

- Preconvert samples to the six-bit DAC domain so the playback loop does not scale every sample.
- Avoid clipping: leave a little headroom when mixing multiple software voices.
- For interrupt-driven playback, use a ring buffer and keep the ISR to fetch-output-advance work.
- When exact pitch matters, derive timing from a hardware source rather than a BASIC delay loop.
- Test on the actual analog output path. Emulators cannot reproduce every noise, filter and loading characteristic of a physical machine.

21. Cassette interface

The built-in cassette interface stores data as audio-frequency-shift-keyed information on ordinary tape. The classic format uses tones around 1200 and 2400 Hz and an effective data rate around 1500 baud. The PIA controls cassette motor and data paths. Cassette files are block-oriented and use leaders/synchronization so the ROM routines can acquire timing from tape.

Element	Typical format role
Leader	A run of $55 bytes/tone pattern used for synchronization.
Sync	Block synchronization marker, conventionally $3C after the leader byte.
Block type	Identifies name/header, data or end-of-file block.
Length	Number of data bytes in the block.
Payload	Up to 255 bytes in a data block.
Checksum	Simple integrity check used by the cassette format.
Trailer	Finishes the block and permits resynchronization.

Machine-code cassette files include load and execution address information in their header. ASCII versus binary flags and gap handling are part of the file metadata. When implementing a PC-side cassette converter, use the official ROM format as the authority and preserve timing tolerance rather than generating theoretically perfect zero-jitter edges that some real cassette hardware may reproduce poorly.

22. Serial/printer bit-banger

The CoCo serial/printer interface is software driven through the PIA rather than backed by a UART. The ROM printer path uses conservative timing suitable for the historical printer interface. Higher serial rates are possible in machine language, but the program becomes responsible for bit timing, framing and interrupt interference. This is another area where CPU speed must be treated as part of the protocol.

Because a bit-banged transmitter occupies CPU time at every bit boundary, buffering only solves the producer side; the actual shift timing must still happen on schedule. If the same program is also producing DAC audio or raster effects, assign priorities and quantify the worst-case latency instead of hoping all timing loops coexist.

23. The cartridge port

The 40-pin cartridge connector exposes much of the CPU bus plus control, chip-select, interrupt and audio signals. It is the path used by Program Paks, disk controllers, serial interfaces, sound devices and the Multi-Pak Interface. Important signal families include A0-A15, D0-D7, read/write, E/Q timing, cartridge and spare chip selects, HALT/NMI/RESET-related control, power/ground and cartridge sound.

The SCS region is traditionally associated with the $FF40-$FF5F expansion I/O window, while cartridge ROM selection and mapping depend on CTS/ROM controls and the current GIME ROM map. Hardware design should always use the exact CoCo 3 service-manual connector table and timing diagrams rather than a CoCo 1-era pin list copied from memory.

Never hot-plug Tandy service documentation warns against inserting/removing cartridges with power applied. The edge connector carries live bus and power signals; hot-plugging can corrupt memory or damage components.

23.1 Multi-Pak considerations

A Multi-Pak Interface lets several cartridges share the expansion connector under slot-selection control. Older units and address-decode revisions can require a CoCo 3 compatibility upgrade. Low-level programs should also remember that third-party devices may decode more addresses than they should, producing register ghosting or conflicts with newer GIME registers. If an unexplained hardware failure disappears when the MPI is removed, decode/timing compatibility belongs high on the diagnostic list.

24. Disk Extended Color BASIC and the floppy format

A standard CoCo disk system adds a floppy controller cartridge and Disk Extended Color BASIC (DECB). The canonical single-sided 5.25-inch filesystem uses 35 tracks, 18 sectors per track and 256 bytes per sector. Raw capacity is 35 x 18 x 256 = 161,280 bytes. Track 17 is reserved for allocation and directory information, leaving 34 tracks for ordinary file data.

Quantity	Value
Tracks	35, numbered 0-34
Sectors/track	18, numbered 1-18
Bytes/sector	256
Raw bytes	161,280
Allocation unit	1 granule = 9 sectors = 2,304 bytes
Data granules	68, numbered 0-67
Directory track	Track 17
GAT	Track 17, sector 2; first 68 bytes describe granules
Directory	Track 17, sectors 3-11
Directory entries	72 possible 32-byte slots in the standard directory area

24.1 Directory entry

Offset	Length	Meaning
0	8	Filename, space padded
8	3	Extension, space padded
11	1	File type: traditionally 0 BASIC, 1 data, 2 machine-language/binary, 3 text-editor/text
12	1	ASCII/binary flag
13	1	First granule number
14	2	Bytes used in final sector / final-length information
16	16	Reserved/unused in the classic format

24.2 Granule allocation table

Each of the first 68 bytes of track 17 sector 2 corresponds to one data granule. $FF means free. Values below the end-marker range link to the next granule in a file chain. End markers in the $C0-and-up family encode that this is the final granule and how many sectors of it are used. Robust disk utilities must validate chains: detect out-of-range links, loops, cross-linked granules and directory entries that point at impossible starts.

24.3 Raw sector I/O

DECB provides DSKI$ and DSKO$ for 256-byte sector reads and writes from BASIC. Because BASIC strings have length constraints, the sector is conventionally split across two string variables. Direct sector access is powerful enough to write disk repair tools, custom loaders and file converters, but a single incorrect DSKO$ to track 17 can destroy the allocation table or directory. Write utilities should make a backup image first and verify drive/track/sector arguments before modifying media.

24.4 JVC .DSK images

A common host-side representation of a standard CoCo disk is a linear JVC-style .DSK image. In its simplest headerless form it is exactly 161,280 bytes, containing sectors consecutively in track order. JVC variants can prepend a small geometry header, and other formats such as DMK preserve lower-level controller information. A tool should identify the image format before calculating offsets; do not assume every file ending in .DSK is a headerless 35-track image.

Primary references: Tandy disk documentation [S4]; standard DECB disk-structure references [S15]; cocofs format documentation [S16].

25. BASIC file and binary loading concepts

DECB distinguishes tokenized BASIC, ASCII BASIC/textual data and machine-language binary files. LOADM reads a machine-language file format that can contain one or more load blocks followed by execution information. SAVE, CSAVE, LOAD, CLOAD and disk equivalents operate on different representations. A host utility that claims to preserve a CoCo file must retain not just visible bytes but the file type, ASCII/binary flag, load metadata where applicable, and exact sector/granule chain.

When moving source code between a modern PC and a CoCo disk image, line endings are a frequent source of trouble. CoCo BASIC source and text utilities often expect carriage-return line termination. A file that visually looks correct on the PC can parse incorrectly on the CoCo if LF or CRLF bytes are inserted without conversion.

26. Video outputs and display realities

The CoCo 3 can feed RF television, composite video and analog RGB. These outputs do not merely differ in sharpness. Composite combines luminance and chroma and therefore exhibits NTSC/PAL encoding artifacts that can be deliberately exploited. RGB presents component color more directly and is the reference for the GIME's two-bits-per-component interpretation. A program designed around composite artifact colors may look entirely different on RGB.

- Use RGB when you need crisp 80-column text and predictable digital palette relationships.
- Use composite testing when software intentionally depends on NTSC artifact colors or when targeting televisions/monitors commonly used with the machine.
- Do not judge a composite palette only from an emulator screenshot. The physical encoder, monitor, cable and capture device all affect the result.
- PAL CoCo 3 machines use different color/timing support circuitry; 50/60 Hz and palette expectations should not be blindly imported from NTSC code.

27. Practical 6809 optimization on the CoCo 3

Optimization on a CoCo 3 is rarely about one magic instruction. The largest gains usually come from moving work out of loops, arranging data for indexed access, reducing MMU transactions, choosing a useful direct page, and avoiding copies that hardware scrolling or page flipping can eliminate.

Technique	Why it helps
Direct page	Turns many hot 16-bit-address references into compact direct accesses.
X/Y/U indexed loops	Efficient pointer traversal; auto-increment/decrement reduces explicit arithmetic.
Precomputed tables	Trade RAM for multiplication, masks, pixel packing, trigonometry, color conversion or address calculations.
Page-aligned data	Simplifies MMU/window changes and lets a routine process whole 8K banks predictably.
Unrolled inner loops	Useful in blitters/audio when code size is cheaper than loop overhead.
Hardware scrolling	Avoids copying tens of kilobytes per frame.
Double buffering/page flip	Moves rendering off-screen and makes display update nearly constant time.
Fast CPU mode	Nearly doubles CPU throughput for code that is safe at the faster bus timing.
FIRQ for tiny real-time jobs	Reduces interrupt entry/exit overhead compared with a full IRQ frame.

27.1 Optimize measurements, not folklore

The 6809 has instruction-dependent cycle counts and indexed modes can add cycles depending on the addressing form. A loop that is shorter in bytes is not always faster. Count cycles for the exact addressing mode, then measure the routine on hardware if video contention, cartridge timing or interrupts can affect it. When benchmarking BASIC or a mixed BASIC/assembly program, make the timing window long enough to exceed the system timer resolution.

28. Safe MMU programming patterns

28.1 Single-window transaction

```
; Conceptual pattern. Choose a logical window that does not contain this code/stack.
; Save and restore the exact MMU register you modify.

        ORCC  #$50          ; example: mask IRQ+FIRQ during the map transaction
        LDA   $FFA4         ; read MMU register; mask upper bits if needed
        ANDA  #$3F
        PSHS  A
        LDA   target_page
        STA   $FFA4         ; physical page -> logical $8000-$9FFF

        ; ... access $8000-$9FFF ...

        PULS  A
        STA   $FFA4
        ANDCC #$AF          ; example only: restore masks properly in production
```

Do not copy the mask example blindly Production code should save and restore the original CC mask bits rather than assuming interrupts were enabled on entry. Likewise, MMU reads on real hardware can contain undefined upper bits; keep only documented page bits.

28.2 Better: a fixed call gate

For large overlays, keep a small dispatcher in a page that never changes. The dispatcher saves the target mapping, maps an overlay into a data/code window, JSRs into it, returns to the fixed gate, restores the old mapping and then returns to the caller. This gives every overlay the same entry/exit contract and greatly reduces the number of places that can strand the CPU in the wrong map.

28.3 Ownership table

For nontrivial software, document page ownership. A simple table that says "page $36 = text screen", "page $34 = graphics buffer", "logical $8000-$9FFF = temporary bank window", and "task 1 map is owned by graphics" prevents an entire class of intermittent corruption bugs.

29. ROM hooks, vectors and compatibility

The CoCo software ecosystem relies heavily on vectors. The fixed CPU vectors point into system code, and the system in turn uses writable RAM vector tables/jump stubs for many services. Hooking a writable vector is generally safer than patching an arbitrary instruction in the ROM image, because the vector expresses an intended indirection point. Even so, a hook must preserve the calling convention and chain to the previous handler when required.

- Record the old vector before replacing it.
- Install the hook atomically with interrupts masked if an interrupt can traverse the vector during modification.
- On removal, restore only if the vector still points to your hook; another component may have chained after you.
- Do not assume a Disk BASIC cartridge, communications cartridge or monitor leaves every vector at the stock value.
- Use the exact ROM-source listing to determine which registers, direct-page assumptions and work areas a routine uses.

30. Hardware versus emulator behavior

Modern emulators are indispensable development tools, but the CoCo 3 contains analog paths, bus timing and partially documented GIME behavior that cannot be reduced to instruction semantics alone. The most dangerous emulator discrepancy is not an obvious missing feature; it is a convenient behavior that real hardware never guaranteed.

Area	Typical trap
GIME register reads	An emulator may return the written value for a register that behaves like open bus or partially readable hardware on a real machine.
MMU reads	Upper bits may reflect bus bleed and must be masked to the documented page width.
Timing	Cycle-perfect-looking raster or serial code may have no margin for physical GIME, bus or peripheral tolerances.
Joystick/DAC	Analog comparator thresholds and noise do not behave like ideal digital arithmetic.
Composite video	Artifact color and palette appearance are analog phenomena.
Disk	An image file has no head-settle, rotational, marginal-sector or write-protect behavior unless the emulator models it.
Expansion bus	Real cartridges can have decode conflicts, propagation delay and power-loading effects absent from a virtual device.

Best verification loop Use the emulator to iterate quickly, use source/disassembly to understand software behavior, and use a physical CoCo 3 to settle hardware-specific questions. When hardware disagrees with an emulator, investigate the assumption before changing the hardware-oriented code to satisfy the emulator.

31. Common failure patterns and how to debug them

Symptom	High-value checks
Immediate crash after MMU write	Did you remap the executing code, S stack, DP data, vector page or return address? Did task 0/1 selection change?
Works once, fails on second run	A register, palette, interrupt vector, MMU page, stack, CPU speed or PIA direction was not restored. Compare entry state on run 1 and run 2.
Works in WIDTH 32, fails in WIDTH 40/80	High-resolution text uses different physical memory and ROM paths. Check page $36 ownership, task mappings, screen initialization and vector assumptions.
Graphics corrupt after BASIC command	BASIC reclaimed HSCREEN/HGET/HPUT pages or changed video/MMU registers. Define resource ownership.
Random color/sparkles	Check palette timing, display-start alignment, GIME revision, RGB/composite path, memory timing and expansion-bus devices.
Timer frequency off slightly	Check TINS source, reload semantics, GIME revision differences and whether the emulator models the same count behavior.
Joystick values change during audio	DAC/comparator subsystem is shared. Schedule sampling and audio updates.
Disk directory damaged	Verify DSKO$ track/sector, image geometry, GAT chain updates and directory write order. Work from a copy.
BREAK exits unexpectedly	BASIC BREAK is not necessarily a character returned through INKEY$. Use the language's BREAK handling mechanism or a low-level keyboard scan depending on the desired contract.
Register read-modify-write breaks hardware only	The register may be write-only/open-bus. Replace RMW with a software shadow and explicit write.

32. A disciplined CoCo 3 debugging method

1. Freeze a known-good disk/image and calculate its checksum before making changes.
2. Reduce the failure to the smallest path that still reproduces it on real hardware.
3. Record machine configuration: 128K/512K, GIME revision if known, disk controller, MPI/expansions, video output, CPU speed and BASIC/Disk BASIC versions.
4. Capture the current values you can reliably observe: MMU page registers (mask undefined bits), task selector, relevant palette values, CPU speed state, software shadow copies of write-only GIME registers, stack pointer and important vectors.
5. Change one ownership assumption at a time. Avoid simultaneously moving code, changing screen mode and rewriting an interrupt handler.
6. Use the ROM source or Unravelled listing to follow any BASIC command that precedes the crash. Determine which memory maps and work areas it changes.
7. If an emulator disagrees, construct a small hardware probe instead of expanding the full application until the behavior is understood.
8. After a fix, repeat the operation at least twice without rebooting. Re-entry exposes leaked state that a cold boot hides.
9. Regress WIDTH/text mode, graphics mode, disk access, BREAK/interrupt behavior and any other subsystem that shares the modified state.
10. Promote a revision only after a hardware pass; preserve the previous passing image as a frozen baseline.

33. Programming patterns for robust applications

33.1 Enter / operate / leave

Treat every subsystem change as a transaction. On entry, snapshot the state you are responsible for. During operation, own it explicitly. On every normal, error and BREAK path, restore it. This applies to WIDTH, palettes, MMU windows, task selection, CPU speed, interrupt masks, PIA control and vector hooks.

33.2 Separate UI state from hardware state

A BASIC variable that says "mode=2" is not proof that the GIME is still in the corresponding mode; another ROM routine may have changed the hardware. Conversely, reading a write-only register is not proof either. Keep authoritative software shadows and reassert critical hardware state at well-defined boundaries.

33.3 Prefer bounded critical sections

Masking interrupts for an entire drawing operation makes software simple but can destroy audio and input responsiveness. Instead, disable interrupts only while replacing a mapping or vector that an interrupt could traverse. Reenable them while bulk work proceeds in a stable mapping.

33.4 Build hardware diagnostics into development versions

During development, print or log the task number, relevant MMU pages, stack pointer, mode shadow, last error line and state transitions before returning to BASIC. A one-line diagnostic such as "XMEM FAIL 6 CORE 6 FREE 36" is vastly more useful on real hardware than a silent green screen because it identifies which invariant failed before the display environment was destroyed.

34. Modern cross-development without changing the target

A modern PC can make CoCo 3 development dramatically faster while the target program remains pure ROM/BASIC/6809 code. The key is to treat host tools as build and inspection aids rather than as a different runtime model.

Tool class	Use
6809 assembler/linker	LWASM/LWLINK and other 6809 assemblers can generate binaries, listings, symbol maps and cycle-aware code for the real machine.
ROM source	coco_roms provides buildable/disassembled BASIC sources for tracing interpreter behavior and symbols.
Disk-image tools	DECB-aware utilities can list, inject, extract and validate files in .DSK images without repeatedly writing physical media.
Emulators	XRoar, VCC, MAME and others are useful for rapid iteration, tracing and screenshots; final hardware-sensitive behavior still requires physical verification.
Checksums/diffs	Hash every frozen baseline and perform binary file/disk diffs so a narrow fix remains narrow.
Hex/disassembly tools	Inspect tokenized BASIC, ML load blocks, GAT/directory sectors, palette tables and binary patches.

Target discipline Do not let a host tool silently normalize line endings, rewrite a disk filesystem, alter BASIC tokenization or change binary load addresses. A reproducible build should specify every conversion step.

35. Quick-reference: important addresses

Address	Function / note
$FF00-$FF03	PIA0 - keyboard/joystick-related I/O and controls
$FF20-$FF23	PIA1 - DAC, sound, cassette, serial and related controls
$FF40-$FF5F	Expansion / SCS I/O window used by disk and cartridges
$FF90	GIME INIT0
$FF91	GIME INIT1 / MMU task / timer source
$FF92-$FF93	GIME IRQ / FIRQ enable-status registers
$FF94-$FF95	12-bit GIME timer
$FF98	GIME VMODE
$FF99	GIME VRES
$FF9A	Border color
$FF9C	Vertical scroll
$FF9D-$FF9E	Video display start
$FF9F	Horizontal offset / virtual-width control
$FFA0-$FFA7	MMU task 0 page registers
$FFA8-$FFAF	MMU task 1 page registers
$FFB0-$FFBF	Palette registers 0-15
$FFC0-$FFD3	Legacy SAM-compatible video/address strobes
$FFD8-$FFD9	Common slow / fast CPU-speed write strobes
$FFDE-$FFDF	Legacy ROM / all-RAM mapping strobes
$FFF2-$FFFE	SWI3, SWI2, FIRQ, IRQ, SWI, NMI, RESET vectors at the standard even addresses

36. Quick-reference: GIME register map

Register	Name	Key meaning
$FF90	INIT0	Compatibility/native, MMU enable, IRQ/FIRQ master, ROM mapping, constant vectors/SCS controls
$FF91	INIT1	Timer source and active MMU task; other bits reserved/implementation-related
$FF92	IRQ	IRQ source enables/status
$FF93	FIRQ	FIRQ source enables/status
$FF94	TIMER MS	Upper four timer bits / reload control sequence
$FF95	TIMER LS	Lower eight timer bits
$FF96-$FF97	Reserved	Do not use as scratch storage
$FF98	VMODE	Native text/graphics and video timing controls
$FF99	VRES	Lines/field, bytes/row, color depth/attribute mode
$FF9A	BRDR	Native border color
$FF9B	Reserved	Stock GIME: reserved; expansions may repurpose
$FF9C	VOFF	Vertical fine scroll
$FF9D-$FF9E	VSTART	Physical video-start encoding
$FF9F	HOFF/HVEN	Horizontal offset and 256-byte virtual-row mode
$FFA0-$FFAF	MMU	Two sets of eight 8K mapping registers
$FFB0-$FFBF	PALETTE	Six-bit hardware color code for each of 16 palette entries

37. Quick-reference: MMU blocks

Logical page	CPU range	Task 0	Task 1
0	$0000-$1FFF	$FFA0	$FFA8
1	$2000-$3FFF	$FFA1	$FFA9
2	$4000-$5FFF	$FFA2	$FFAA
3	$6000-$7FFF	$FFA3	$FFAB
4	$8000-$9FFF	$FFA4	$FFAC
5	$A000-$BFFF	$FFA5	$FFAD
6	$C000-$DFFF	$FFA6	$FFAE
7	$E000-$FFFF	$FFA7	$FFAF

Stock 512K physical page number range: $00-$3F. Stock 128K installed RAM: physical pages $30-$3F. Mask MMU readback to the documented six page bits before comparing page numbers.

38. Quick-reference: graphics packing

Color depth	Bits/pixel	Pixels/byte	Pixel extraction concept
2 colors	1	8	Bit selects palette index 0/1
4 colors	2	4	Two-bit field selects one of four palette indices
16 colors	4	2	High or low nibble selects one of sixteen palette indices

```
; Address of a pixel in a conventional linear bitmap:
row_base = screen_base + y * bytes_per_row

2-color:   byte = row_base + (x >> 3), bit field = 7 - (x & 7)
4-color:   byte = row_base + (x >> 2), 2-bit field depends on x & 3
16-color:  byte = row_base + (x >> 1), high/low nibble depends on x & 1
```

39. Quick-reference: standard DECB disk

Structure	Location / value
Geometry	35 tracks x 18 sectors x 256 bytes = 161,280 bytes raw
Allocation unit	9 sectors = 2,304-byte granule
Usable granules	68
Directory track	17
GAT	Track 17 sector 2, first 68 bytes
Directory entries	Track 17 sectors 3-11; 32 bytes each; 72 slots
Free granule marker	$FF
Directory filename	8 characters + 3-character extension, space padded

40. GIME and machine revision realities

CoCo 3 production spans more than one GIME revision. Community hardware research commonly distinguishes early 1986 and later 1987-era chips, with differences in timing behavior and fixes for some video/cartridge issues. The practical rule is not to build ordinary software around revision-specific undefined behavior. If a demo intentionally races the beam, depends on exact timer counts or drives unusual border/video combinations, identify the GIME revision used for testing and include a compatibility path when possible.

Memory expansions beyond 512K also exist, often by extending or repurposing GIME-adjacent behavior. Those upgrades are not part of the stock GIME contract. A shareable program should detect such hardware or require it explicitly rather than accidentally depending on a register extension that is absent on a normal 128K/512K CoCo 3.

Primary references: Community GIME references [S9], hardware programming reference [S8], and Tandy service documentation [S1].

41. Things the CoCo 3 does not have

Knowing what is absent is as useful as knowing what is present. The stock machine has no blitter, sprite engine, tilemap hardware, DMA controller, PCM sound chip, UART, real-time clock, hard-disk interface, protected-memory mode, cache or native 16/32-bit arithmetic unit. Every sprite, mixer, decompressor, filesystem extension and network protocol therefore becomes an exercise in 6809 code, GIME page/palette/video control, cartridge hardware, or some combination of them.

That apparent limitation is exactly why the platform rewards careful systems programming: the programmer can account for nearly every byte moved and nearly every hardware register that changes.

42. Glossary

Term	Definition
6809E	Externally clocked version of Motorola's MC6809 CPU used by the CoCo family.
ACVC	Tandy service-manual name for the CoCo 3 custom video/memory/control chip commonly called GIME.
Artifact color	Apparent composite-video color created by the interaction of high-frequency pixels and the color subcarrier.
DECB	Disk Extended Color BASIC, the disk extension to the Microsoft-derived CoCo BASIC environment.
Direct page	6809 addressing mechanism in which DP supplies the high byte of a one-byte address operand.
FIRQ	Fast interrupt request. The 6809 enters with a reduced automatic stack frame for low latency.
GAT	Granule Allocation Table on a DECB floppy disk.
GIME	Community name for the CoCo 3 multifunction graphics/interrupt/memory enhancement chip.
Granule	DECB allocation unit of 9 sectors = 2,304 bytes.
HSCREEN	Super Extended BASIC command family for CoCo 3 high-resolution graphics screens.
Logical address	The 16-bit address the 6809 issues, from $0000 to $FFFF.
MMU	Memory Management Unit inside the GIME; maps 8K logical windows to physical RAM pages.
Palette	One of 16 GIME registers mapping a pixel index to a six-bit hardware color code.
Physical address	Address in the GIME's larger RAM space, up to $7FFFF for 512K stock addressing.
PIA	Peripheral Interface Adapter; the two 6821-style devices handle keyboard, DAC, cassette, joystick and related I/O.
RSDOS	Common community shorthand for the ROM/Disk BASIC environment. This guide prefers BASIC/DECB where precision matters.
SCS	Spare chip select used by cartridge expansion I/O.
Super Extended BASIC	CoCo 3 BASIC extensions for GIME graphics/text, palettes, long memory and other new facilities.
Task map	One of two sets of eight GIME MMU registers. It is a memory map, not an operating-system process.
VDG/SAM	Legacy CoCo video/address-control components whose behavior is substantially emulated/replaced by the GIME on CoCo 3.

43. Source hierarchy and bibliography

The CoCo community has accumulated decades of excellent reverse engineering, but not every web page deserves the same evidentiary weight. For hardware behavior, this guide uses the following hierarchy: official Tandy/Motorola documentation; ROM source/disassembly that can be checked against shipped binaries; long-standing specialist hardware research; then emulator or anecdotal behavior. When a timing-sensitive community finding conflicts with a physical machine, reproduce the experiment on the target hardware.

S1. Tandy / Radio Shack, Color Computer 3 Service Manual. Primary source for hardware block diagrams, connectors, PIAs, GIME registers, video modes and service specifications. https://colorcomputerarchive.com/repo/Documents/Manuals/Hardware/Color%20Computer%203%20Service%20Manual%20(Tandy).pdf

S2. Tandy / Radio Shack, Color Computer 3 Extended BASIC. Primary user/programmer documentation for Super Extended BASIC commands and display operation. https://colorcomputerarchive.com/repo/Documents/Manuals/Hardware/Color%20Computer%203%20Extended%20Basic%20(Tandy).pdf

S3. Tandy / Radio Shack, Color Computer 3 BASIC Quick Reference Manual. https://colorcomputerarchive.com/repo/Documents/Manuals/Hardware/Color%20Computer%203%20BASIC%20Quick%20Reference%20Manual%20(Tandy).pdf

S4. Tandy / Radio Shack, Color Computer Disk System documentation and programming references, available through the Color Computer Archive manuals collection. https://colorcomputerarchive.com/repo/Documents/Manuals/Hardware/

S5. Motorola, MC6809-MC6809E 8-Bit Microprocessor Programming Manual (M6809PM/AD), 1981/1983. https://github.com/M6809-Docs/m6809pm

S6. Spectral Associates, Super Extended BASIC Unravelled / Unravelled Series. Detailed CoCo 3 ROM, memory map and BASIC internals. https://colorcomputerarchive.com/repo/Documents/Books/Unravelled%20Series/

S7. Laurence A. Tepolt, Assembly Language Programming for the CoCo 3 (1987). Strong explanation of GIME/MMU programming and 6809 techniques. https://colorcomputerarchive.com/repo/Documents/Books/Assembly%20Language%20Programming%20for%20the%20CoCo3%20%281987%29%20%28Laurence%20A%20Tepolt%29.pdf

S8. Chris Lomont, Color Computer 1/2/3 Hardware Programming. Broad hardware-register reference compiled from multiple technical sources. https://www.lomont.org/software/misc/coco/Lomont_CoCoHardware_2.pdf

S9. John Kowalski (Sock Master), GIME Register Reference; mirror maintained by the CoCo community. Essential independent register/timing notes. https://www.6809.org.uk/twilight/sock/gime.html

S10. tomctomc/coco_roms, buildable 6809 assembly source for Color BASIC, Extended BASIC, Disk BASIC and Super Extended BASIC, with ROM hash verification. https://github.com/tomctomc/coco_roms

S11. CoCopedia, Color Computer 3 and related hardware/software articles. Useful community encyclopedia and cross-reference. https://www.cocopedia.com/wiki/Color_Computer_3

S12. Allen C. Huffman / Sub-Etha Software, Understanding and Using the CoCo 3 MMU. Clear physical-page and BASIC-allocation reference. https://subethasoftware.com/2020/04/19/understanding-and-using-the-coco-3-mmu/

S13. Computer Archeology CoCo section. Annotated code/disassembly resources and links to primary material. https://computerarcheology.com/CoCo/

S14. Tandy/TRS-80 hardware documentation archive. Schematics, service manuals and peripheral information. https://tandy-trs80.com/hardware-docs/

S15. Color Computer Archive - Books collection, including the Unravelled series, memory maps, CoCo 3 programming books and period technical references. https://colorcomputerarchive.com/repo/Documents/Books/

S16. cocofs documentation/source. Useful independent description and tooling for the classic 35-track DECB disk-image structure. https://github.com/thorpej/cocofs

S17. Motorola, MC6821 Peripheral Interface Adapter data sheet / programming model. Primary reference for PIA data-direction, control and interrupt semantics. https://colorcomputerarchive.com/repo/Documents/Manuals/Hardware/

S18. Radio Shack-compatible floppy controller memory map as documented in CoCo disk-controller references; standard CoCo mapping uses $FF40 control and $FF48-$FF4B FDC registers. https://colorcomputerarchive.com/repo/Documents/Manuals/Hardware/

S19. LWTOOLS manual, DECB Binaries. Defines the LOADM/CLOADM five-byte load preamble, repeated sections and five-byte execution postamble. https://www.lwtools.ca/manual/manual.html

S20. Sub-Etha Software, Color BASIC program-line and variable-table investigations, cross-checked against the Unravelled/ROM sources. https://subethasoftware.com/tag/color-basic/

44. Final programming checklist

Before calling a CoCo 3 program "hardware safe," answer these questions explicitly:

- Which CPU speed does it require, and does it restore the prior speed?
- Which GIME registers does it write, and does it maintain shadow copies for write-only state?
- Which task map is active at every point where an MMU register is changed?
- Could the code or S stack be remapped by its own bank operation?
- Which physical pages does it own, and can BASIC WIDTH/HSCREEN/HGET/HPUT reuse them?
- Can IRQ/FIRQ occur while a temporary map is active? If yes, is the handler safe in that map?
- Does it preserve/restore vectors, PIA modes, palette, border, video start and text/graphics state?
- If it uses audio and analog input, how are DAC and joystick operations scheduled?
- If it writes disk sectors, does it validate the target format, directory chain and backup image first?
- Has it been run twice without rebooting?
- Has it been tested on the intended physical RAM size and real video output?
- If emulator and hardware disagree, has the hardware behavior been isolated with a small probe?

The CoCo 3 rewards this discipline. Once logical memory, physical memory, video memory and shared I/O ownership are kept distinct, the machine becomes remarkably deterministic. Its apparent constraints become useful boundaries: eight MMU windows, sixteen palettes, explicit interrupt sources, simple PIAs and a CPU whose entire state can be reasoned about. That transparency is what makes the CoCo 3 an unusually good machine for learning - and practicing - real systems programming.

45. Appendix: PIA bit-level reference

The two 6821-style PIAs are the CoCo 3 interface to the keyboard matrix, joystick comparator, cassette, serial bit-banger, six-bit DAC and several legacy compatibility signals. Each PIA has two data ports and two control registers. The data-register address can become a data-direction register when the corresponding control-register DDR-select bit is clear, so code that assumes every read or write is ordinary port data can silently reconfigure the interface.

Address	Primary CoCo 3 use	Important data bits / notes
$FF00	PIA0 port A	Keyboard row inputs on bits 0-6; joystick comparator on bit 7. Keyboard inputs are active-low.
$FF01	PIA0 port A control	Controls the port-A data/DDR selection and CA interrupt/edge behavior. Preserve bits you do not own.
$FF02	PIA0 port B	Keyboard column drive. Software selects one or more columns and reads the row result through $FF00.
$FF03	PIA0 port B control	Controls the port-B data/DDR selection and CB interrupt/edge behavior; also participates in joystick multiplexing on classic CoCo hardware.
$FF20	PIA1 port A	bit 0 cassette input; bit 1 serial output; bits 2-7 six-bit DAC value.
$FF21	PIA1 port A control	Cassette motor/CA control and data-versus-DDR selection. The cassette motor is controlled through the PIA control path.
$FF22	PIA1 port B	bit 0 serial input; bit 1 one-bit sound; remaining legacy video/RAM-sense functions are compatibility-era signals.
$FF23	PIA1 port B control	Includes six-bit sound/DAC routing control and data-versus-DDR selection; cartridge interrupt behavior also shares this PIA side.

PIA discipline A robust program snapshots the PIA control state it intends to change, changes only the required bits, and restores the prior state. Changing a DDR-select bit at the wrong time can make a write alter port direction instead of output data.

PIA0 appears at $FF00-$FF03 and is mirrored through the $FF00-$FF1F region; PIA1 appears at $FF20-$FF23 and is mirrored through $FF20-$FF3F. Use the canonical addresses in new code. Mirrors can be useful when analyzing old software but provide no advantage for normal programming.

Primary references: Tandy CoCo 3 service documentation [S1], Motorola 6821 documentation [S17], and Lomont hardware reference [S8].

46. Appendix: physical connector pinouts

The following pinouts use the connector numbering in Tandy documentation. Always verify connector orientation before wiring a cable; drawings viewed from the outside of the computer and drawings viewed from the solder side are mirror images.

46.1 Analog RGB connector

Pin	Signal	Use
1	GND	Ground
2	GND	Ground
3	R	Red analog video
4	G	Green analog video
5	B	Blue analog video
6	KEY	No pin; polarizing key
7	AUDIO	Audio output
8	HSYNC	Horizontal sync
9	VSYNC	Vertical sync
10	N/C	No connection

46.2 RS-232 / serial-printer DIN

Pin	General RS-232 use	BASIC printer use
1	CD / status input	Not used
2	RS-232 IN	Printer READY/BUSY status input
3	Ground	Ground
4	RS-232 OUT	Serial data to printer

The built-in BASIC printer routines are software timed rather than UART driven. Tandy documents a conventional printer assumption of 600 baud, 8 data bits, no parity and two stop bits; custom software can use different timing and reinterpret the input signals.

46.3 Cassette DIN

Pin	Signal
1	Remote-control relay
2	Signal ground
3	Remote-control relay
4	Input from recorder EAR/output
5	Output to recorder MIC/AUX input

46.4 CoCo 3 joystick DIN

Pin	Signal
1	Horizontal analog/comparator input
2	Vertical analog/comparator input
3	Ground
4	Fire button 1; open high, closed low
5	Current-limited +5 V supply
6	Fire button 2 (CoCo 3 addition)

Hardware warning Do not use connector supply pins as general-purpose power rails for arbitrary loads, and do not hot-plug cartridge hardware. For electrical limits, connector orientation and protection circuitry, consult the service manual or the schematic for the exact board revision.

Primary references: Tandy CoCo 3 Service Manual [S1], CoCopedia connector references [S11], and Tandy Color Computer hardware manuals [S14].

47. Appendix: Radio Shack floppy-controller registers

The classic Radio Shack CoCo disk interface occupies the cartridge SCS window. A common source of confusion is that some third-party hardware can switch between CoCo and Dragon address schemes. The standard CoCo mapping places the drive-control latch at $FF40 and the FDC command/status block at $FF48-$FF4B.

Address	Function	Notes
$FF40	Drive/control latch	Write-only on the standard interface; software normally keeps a shadow copy. Mirrors commonly occupy $FF41-$FF47.
$FF48	FDC command (write) / status (read)	WD179x-family command/status register.
$FF49	FDC track register	Current/target track value used by the controller.
$FF4A	FDC sector register	Sector number used for sector commands.
$FF4B	FDC data register	Byte data path for read/write operations.

47.1 $FF40 drive/control latch

Bit	Function
0	Select drive 0
1	Select drive 1
2	Select drive 2
3	Drive motor enable
4	Write precompensation control
5	Density control
6	Select drive 3
7	HALT control used by disk I/O timing

Controller variants The exact FDC silicon varies among Radio Shack and compatible controllers (for example WD1793-family and compatible parts), and third-party controllers may add registers. Program to the controller you actually target; do not infer hardware solely from a .DSK image.

Primary references: Disk BASIC Unravelled [S6], Tandy disk-controller/service documentation [S4], and Radio Shack-compatible controller mapping [S18].

48. Appendix: DECB LOADM/CLOADM binary format

Disk Extended Color BASIC machine-language files are not necessarily raw memory dumps. The DECB binary format can contain multiple independently addressed load blocks followed by one execution record. This is why a correctly generated LOADM file can place noncontiguous code/data sections with one command.

Record	Bytes	Meaning
Load preamble	00 LL LL AA AA	00 marker; 16-bit big-endian byte count; 16-bit big-endian load address. Exactly that many data bytes follow.
Additional load block	00 LL LL AA AA + data	May repeat, allowing multiple loadable sections.
Postamble	FF 00 00 EE EE	FF marker; two zero bytes; 16-bit big-endian execution address.

```
00 00 03 30 00 12 34 39 FF 00 00 30 00
|  |-----| |-----| -------- |  |---| |---|
|   length   load    data    |  zeros  exec
load block                   postamble
```

The example above loads three bytes at $3000 and supplies $3000 as the execution address. LOADM can load the file without immediately executing it; EXEC or a LOADM form that uses the execution address can subsequently transfer control according to BASIC behavior. CLOADM uses the same conceptual binary structure when machine-language data is transported by cassette.

Primary references: LWTOOLS DECB binary-format documentation [S19] and Tandy Disk BASIC documentation [S4].

49. Appendix: BASIC program and variable storage

Understanding BASIC's in-memory representation is invaluable when writing machine-language extensions, editors, tracers or memory inspectors. The interpreter stores a tokenized linked sequence of program lines followed by variable and string-related areas. Direct modification is powerful but unsafe unless the program is stopped at a stable interpreter point and the relevant pointers are updated consistently.

49.1 Tokenized program lines

Field	Size	Meaning
Next-line pointer	2 bytes	Address of the next tokenized line in memory.
Line number	2 bytes	Big-endian BASIC line number.
Tokenized text	variable	Keywords are represented by token bytes; literals and punctuation remain encoded as expected by the interpreter.
Terminator	1 byte	$00 ends the line.

Thus every program line carries five bytes of structural overhead in addition to its tokenized contents. In the classic Microsoft-derived CoCo BASIC workspace, locations 25/26 contain the program-start pointer and 27/28 contain VARTAB, the start of the simple-variable table.

49.2 Simple variables

A simple variable entry occupies seven bytes: two bytes encode the significant variable name/type and five bytes hold the numeric value or string descriptor. Numeric variables use the interpreter's five-byte floating-point representation. For a string, the descriptor includes the string length and a pointer to the character data; the maximum BASIC string length is 255 bytes.

Do not confuse storage and accumulator formats The Microsoft BASIC floating-point package uses internal working/accumulator state that is not identical to the five-byte value stored in a normal numeric variable. Copying arbitrary FP workspace bytes into a variable is not a valid general conversion.

Primary references: Unravelled BASIC references [S6], buildable ROM source [S10], and Sub-Etha BASIC memory investigations [S20].

50. Appendix: safe RAM-size and MMU probing

The GIME exposes a six-bit physical-page field, but installed RAM may be smaller than the 512K addressable stock maximum. A 128K CoCo 3 physically provides only sixteen 8K pages, conventionally visible as pages $30-$3F in the BASIC environment. Reading an MMU register therefore tells you what page number is programmed, not whether independent RAM actually exists at that page.

A destructive RAM-size test is easy to write and easy to get wrong: aliasing can make a write to a supposedly absent page appear somewhere else, and the test itself can overwrite BASIC, video RAM, stacks or ROM-shadow workspace. A safe diagnostic reserves two known scratch locations, records their contents, disables or controls interrupts, changes one MMU window at a time, writes a pattern, checks for aliasing against a known installed page, and restores every byte and mapping before returning.

- Never probe through the logical window containing the running test code, hardware stack or live interrupt handler.
- Do not use screen or BASIC-owned long-memory pages as scratch without first taking ownership of them.
- Keep a software copy of the exact MMU register value you replace and restore it even on an error path.
- Mask readback to documented page bits; do not interpret undefined high bits as installed-memory information.
- Prefer a user-declared 128K/512K requirement when reliability matters more than automatic detection.

CoCo 3 RAM detection fact Unlike earlier CoCo RAM-size conventions, software should not expect a simple dedicated PIA flag that reliably distinguishes a stock 128K CoCo 3 from a 512K expansion. Detection must be designed around actual memory behavior or an explicit configuration.

Primary references: Super Extended BASIC Unravelled / memory maps [S6], GIME references [S8, S9], and CoCo 3 service documentation [S1].

51. Appendix: timing, interrupts and benchmarking

Cycle counts alone are necessary but not sufficient for CoCo 3 performance work. The 6809E may run at the normal or fast clock, memory/peripheral accesses can impose hardware-specific timing effects, interrupts steal deterministic chunks of execution time, and video/raster synchronization changes when a loop begins. For that reason benchmark design is part of systems programming, not an afterthought.

51.1 Interrupt entry costs and state

IRQ stacks the full normal machine context selected by the 6809 interrupt mechanism; FIRQ is designed for low latency and automatically stacks a reduced state. A FIRQ handler that needs additional registers must save them explicitly. The handler must also acknowledge the hardware source correctly before returning or it may immediately re-enter.

51.2 GIME interval timer

The GIME interval timer is programmed through $FF94-$FF95 and can generate IRQ or FIRQ through the corresponding enable register. Community hardware measurements distinguish a fast timer source on the order of one GIME clock period and a much slower source on the order of a horizontal-line interval. Exact timing-sensitive code should measure the target GIME/revision and video environment rather than treating emulator timing as laboratory instrumentation.

51.3 Benchmark rules

- Warm up or explicitly initialize the state being measured; first-run initialization should not accidentally become part of every sample.
- Measure enough iterations that the elapsed interval spans many timer ticks. A zero-tick result means the measurement window is too short, not that the routine is infinitely fast.
- Record CPU speed and interrupt state with every result.
- If video is active, state whether the benchmark includes waits for vertical/horizontal events.
- Compare identical machine state before and after an optimization; especially check MMU mappings and BASIC text/graphics state.
- On real hardware, repeat samples and report dispersion rather than only the single fastest number.

For cycle-critical 6809 code, an assembler listing with instruction cycle counts is useful for local reasoning, but the final authority is a timed experiment on the intended CoCo 3 hardware configuration.

Primary references: Motorola 6809 programming documentation [S5], GIME timing research [S9], and hardware programming reference [S8].