RomWBW

System Guide

Version 3.6

Updated 20 Aug 2025

RetroBrew Computers Group
www.retrobrewcomputers.org

Wayne Warthen
wwarthen@gmail.com

Contents

Chapter 1

Overview

The objective of RomWBW is to provide firmware, operating systems, and applications targeting the Z80 family of CPUs. The firmware, in the form of a ROM module, acts as the hardware interface layer with a well-defined API (the HBIOS). The associated operating systems and applications are adapted to the HBIOS API, not specific hardware.

The HBIOS is modular and configurable. New hardware interfaces can be added in the form of straightforward driver modules. Within certain constraints, new hardware platforms can be supported by simply adjusting values in a build configuration file.

RomWBW is geared toward hardware being developed in modern retro-computing hobbyist communities, not as replacement software for legacy hardware. As a result, RomWBW requires at least 128KB of bank switched RAM.

The CP/M family of operating systems has been adapted to run under RomWBW including CP/M 2.2, Z-System, CP/M 3, and several other variants.

RomWBW firmware (ROM) includes:

- System startup code (bootstrap) and bootloader
- A basic system/debug monitor
- HBIOS (Hardware BIOS) with support for most typical hardware components used in Z80 family computers
- Diagnostics and customizable debugging information.
- ROM-hosted operating systems (both CP/M 2.2 and Z-System)

- A ROM disk containing the standard OS applications and a RAM disk for working storage.

It is appropriate to note that much of the code and components that make up a complete RomWBW package are derived from pre-existing work. Most notably, the embedded operating systems are simply ROM-based copies of generic CP/M or ZSDOS. Much of the hardware support code was originally produced by other members of the RetroBrew Computers Community.

The remainder of this document focuses on RomWBW HBIOS which is the fundamental basis of RomWBW.

Chapter 2

Background

The Z80 CPU architecture has a limited, 64K address range. In general, this address space must accommodate a running application, disk operating system, and hardware support code.

Modern retro-computing Z80 CPU platforms provide a physical address space that is much larger than the CPU address space (typically 512K or 1MB of physical RAM). This additional memory can be made available to the CPU using a technique called bank switching. To achieve this, the physical memory is divided up into chunks (banks) of 32K each. A designated area of the CPU's 64K address space is then reserved to "map" any of the physical memory chunks. You can think of this as a window that can be adjusted to view portions of the physical memory in 32K blocks. In the case of RomWBW, the lower 32K of the CPU address space is used for this purpose (the window). The upper 32K of CPU address space is assigned a fixed 32K area of physical memory that never changes. The lower 32K can be "mapped" on the fly to any of the 32K banks of physical memory at a time. The primary constraint is that the CPU cannot be executing code in the lower 32K of CPU address space at the time that a bank switch is performed.

By utilizing the pages of physical RAM for specific purposes and swapping in the correct page when needed, it is possible to utilize substantially more than 64K of RAM. Because the retro-computing community has now produced a very large variety of hardware, it has become extremely important to implement a bank switched solution to accommodate the maximum range of hardware devices and desired functionality.

Chapter 3

General Design Strategy

The design goal is to locate as much of the hardware dependent code as possible out of normal 64KB CP/M address space and into a bank switched area of memory. A very small code shim (proxy) is located in the top 512 bytes of CPU memory. This proxy is responsible for redirecting all hardware BIOS (HBIOS) calls by swapping the “driver code” bank of physical RAM into the lower 32K and completing the request. The operating system is unaware this has occurred. As control is returned to the operating system, the lower 32KB of memory is switched back to the original memory bank.

The HBIOS functions are invoked simply by placing function parameters in Z80 registers and calling an address within the HBIOS proxy. Additionally, HBIOS implements a complete hardware interrupt management framework. When a hardware interrupt occurs, control vectors through the HBIOS proxy which saves the machine state, selects the HBIOS driver bank into memory, and transfers control to the registered driver’s interrupt handler. Upon completion of interrupt processing, control returns via the HBIOS proxy, machine state is restored, and normal processing resumes. The interrupt management framework supports Z80 interrupt modes 1, 2, and 3 (Z280).

HBIOS is completely agnostic with respect to the operating system (it does not know or care what operating system is using it). The operating system makes simple calls to HBIOS to access any desired hardware functions. Since the HBIOS proxy occupies only 512 bytes at the top of memory, the vast majority of the CPU memory is available to the operating system and the running application. As far as the operating system is concerned, all of the hardware driver code has been magically implemented inside of the small 512 byte area at the top of the CPU address space.

Unlike some other Z80 bank switching schemes, there is no attempt to build bank switching into the operating system itself. This is intentional so as to ensure that any operating system

can easily be adapted without requiring invasive modifications to the operating system itself. This also keeps the complexity of memory management completely away from the operating system and applications.

There are some operating systems that have built-in support for bank switching (e.g., CP/M 3). These operating systems are allowed to make use of the bank switched memory and are compatible with HBIOS. However, it is necessary that the customization of these operating systems take into account the banks of memory used by HBIOS and not attempt to use those specific banks.

Note that all code and data are located in RAM memory during normal execution. While it is possible to use ROM memory to run code, it would require that more upper memory be reserved for data storage. It is simpler and more memory efficient to keep everything in RAM. At startup (boot) all required code is copied to RAM (shadowed) for subsequent execution.

Chapter 4

Runtime Memory Layout

RomWBW divides the standard 64KB Z80 address space into 2 sections. The lower 32KB is the “banked” area. This is the area that will contain any of the 32KB chunks of physical RAM based on which bank is currently selected. The upper 32KB is “fixed”. This area of memory is never swapped out and is used to contain software and operating systems that must remain in the Z80 address space.

Throughout this document, this mechanism of selecting banks of memory into the lower 32K is referred to as memory management. Achieving this functionality requires some type of hardware which is generally referred to as the system’s Memory Management Unit (MMU). RomWBW supports a variety of MMUs – but they all perform the same function of swapping in/out banks of memory in the lower 32K of CPU address space.

Figure 4.1 depicts the memory layout for a system running the CP/M operating system. Applications residing in TPA invoke BDOS services of CP/M, BDOS invokes the custom CBIOS APIs, and finally CBIOS invokes HBIOS functions as needed by calling into the HBIOS proxy. The HBIOS proxy swaps in the HBIOS bank as needed to perform the requested function.

Additional banks of RAM are used to create a virtual disk drive.

4.1 Bank Id

RomWBW utilizes a specific assignment of memory banks for dedicated purposes. A numeric Bank Id is used to refer to the memory banks. The Bank Id is a single byte. In general, the Bank Id simply refers to each of the 32K banks in sequential order. In other words, Bank Id 0 is the first physical 32K, Bank Id 1 is the second, etc. However, the high order bit of the Bank Id has a special meaning. If it is 0, it indicates a ROM bank is being referred to. If it is 1, it indicates a

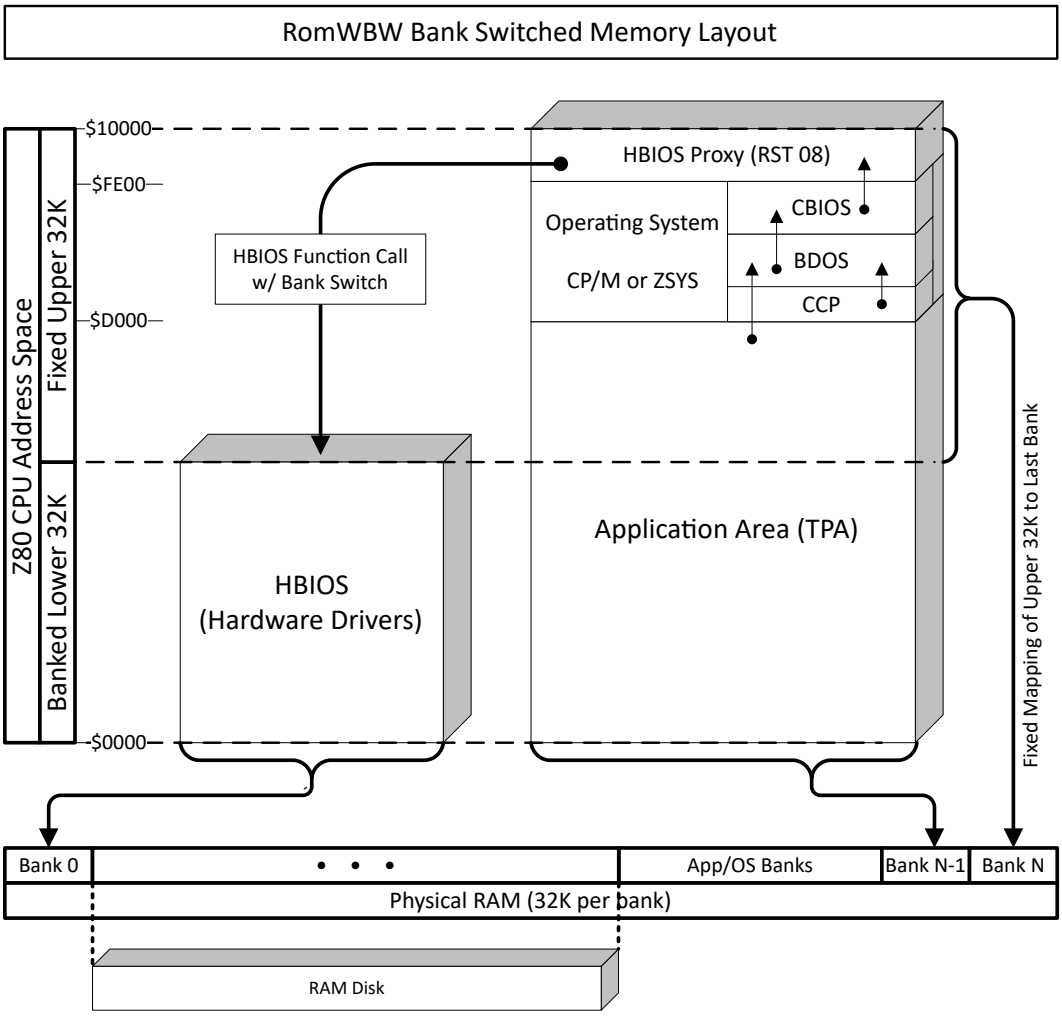

Figure 4.1: Bank Switched Memory Layout

RAM bank is being referred to.

For example, let's say we have a typical system with 512KB of ROM and 512KB of RAM. The following table demonstrates how Bank Ids represent areas of physical memory.

Physical Memory	Type	Physical Bank	Bank Id
0x000000-0x007FFF	ROM	0	0x00
0x008000-0x00FFFF	ROM	1	0x01
0x010000-0x07FFFF	ROM	2-15	0x02-0x0F
0x080000-0x087FFF	RAM	16	0x80
0x088000-0x08FFFF	RAM	17	0x81
0x090000-0x0FFFFF	RAM	18-31	0x82-0x8F

Note that Bank Id 0x00 is **always** the first bank of ROM and 0x80 is **always** the first bank of RAM. If there were more banks of physical ROM, they would be assigned Bank Ids starting with 0x10. Likewise, additional bank of physical RAM would be assigned Bank Ids starting with 0x90.

The Bank Id is used in all RomWBW API functions when referring to the mapping of banks to the lower 32K bank area of the processor. In this way, all RomWBW functions can refer to a generic Bank Id without needing to understand how a specific hardware platform accesses the physical memory areas. A single routine within the HBIOS is implemented for each memory manager that maps Bank Ids to physical memory.

4.2 Bank Assignments

RomWBW requires dedicated banks of memory for specific purposes. It uses Bank Ids via an algorithm to make these assignments. The following table describes the way the banks are assigned. The Typical column shows the specific values that would be assigned for a common system with 512KB of ROM and 512KB of RAM (nROM=16, nRAM=16).

Bank Id	Identity	Typical	Purpose
0x00	BID_BOOT	0x00	Boot Bank (HBIOS image)
0x01	BID_IMG0	0x01	Boot Loader, Monitor, ROM OSes, ROM Apps
0x02	BID_IMG1	0x02	ROM Apps
0x03	BID_IMG2	0x03	<Reserved>
0x04	BID_ROMD0	0x04	First ROM Disk Bank

Bank Id	Identity	Typical	Purpose
nROM - 1		0x0F	Last ROM Disk Bank
0x80	BID_BIOS	0x80	HBIOS (working copy)
0x81	BID_RAMD0	0x81	First RAM Disk Bank
0x80 + nRAM - 8		0x88	Last RAM Disk Bank
0x80 + nRAM - 7	BID_APP0	0x89	First Application Bank
0x80 + nRAM - 5		0x8B	Last Application Bank
0x80 + nRAM - 4	BID_BUF	0x8C	OS Disk Buffers
0x80 + nRAM - 3	BID_AUX	0x8D	OS Code Bank
0x80 + nRAM - 2	BID_USR	0x8E	User Bank (CP/M TPA)
0x80 + nRAM - 1	BID_COM	0x8F	Common Bank

In this table, nROM and nRAM refer to the number of corresponding ROM and RAM banks in the the system.

The contents of the banks referred to above are described in more detail below:

Boot Bank: The Boot Bank receives control when a system is first powered on. It contains a ROM (read-only) copy of the HBIOS. At boot, it does minimal hardware initialization, then copies itself to the HBIOS bank in RAM, then resumes execution from the RAM bank.

Boot Loader: The application that handles loading of ROM or Disk based applications including operating systems. It copies itself to a RAM bank at the start of it's execution.

Monitor: The application that implements the basic system monitor functions. It copies itself to a RAM bank at the start of it's execution.

ROM OSes: Code images of CP/M 2.2 and Z-System which are copied to RAM and executed when a ROM-based operating system is selected in the Boot Loader.

ROM Applications: Various ROM-based application images such as BASIC, FORTH, etc. They can be selected in the Boot Loader. The Boot Loader will copy the application image to a RAM bank, then transfer control to it.

ROM Disk: A sequential series of banks assigned to provide the system ROM Disk contents.

HBIOS: This bank hosts the running copy of the RomWBW HBIOS.

RAM Disk: A sequential series of banks assigned to provide the system RAM Disk.

Application Bank: A sequential series of banks that are available for use by applications that wish to utilize banked memory.

OS Disk Buffers: This bank is used by CP/M 3 and ZPM3 for disk buffer storage.

OS Code Bank: This bank is used by CP/M 3 and ZPM3 as an alternate bank for code. This allows these operating systems to make additional TPA space available for applications.

User Bank: This is the default bank for applications to use. This includes the traditional TPA space for CP/M.

Common Bank: This bank is mapped to the upper 32K of the processors memory space. It is a fixed mapping that is never changed in normal RomWBW operation hence the name "Common".

4.3 Memory Managers

The following hardware memory managers are supported by RomWBW. The operation of these memory managers is not documented here – please refer to the documentation of your hardware provider for that.

Z2: Memory memory manager introduced by Sergey Kiselv in the Zeta 2 SBC. Popular in many RCBus systems.

Z180: Memory manager built into the Z180 CPU

Z280: Memory manager built into the Z280 CPU

ZRC: Memory manager onboard the ZRC series of computers by Bill Shen.

SBC: Memory manager onboard the N8VEM SBC series of computers by Andrew Lynch.

MBC: Memory manager onboard the Nhyodyne computer system by Andrew Lynch.

N8: Memory manager onboard the N8 SBC computer by Andrew Lynch.

EZ512: Memory manager onboard the EaZy80-512 Z80 CPU Module by Bill Shen.

RPH: Memory manager onboard the Rhyophyre computer system by Andrew Lynch.

The memory manager used is determined by the configuration choices that are part of a RomWBW build process. A given ROM can only have a single memory manager – it is not selected dynamically.

The configuration variable `MEMMGR` sets the memory mannager used by the ROM build. It must be set to one of the above memory manager types. For example, for the Z2 memory manager, `MEMMGR` should be set to `MM_Z2`.

Note that the term memory manager (MM) and memory management unit (MMU) are used interchangeably in the documentation and code.

Chapter 5

Disk Layout

5.1 Floppy Disk Layout

RomWBVW generally handles floppy disks in the same physical formats as MS-DOS. However, the filesystem will normally be CP/M. The following table lists the floppy disk formats used by RomWBW. In all cases, the sector size is 512 bytes.

HBIOS Media ID	Capacity	Tracks	Heads	Sectors
MID_FD720	720KB	80	2	9
MID_FD144	1440KB	80	2	18
MID_FD360	360KB	40	2	9
MID_FD120	1200KB	80	2	15
MID_FD111	1155KB	77	2	15

5.2 Hard Disk Layout

RomWBW supports the use of PC MBR hard disk partitioning (see https://en.wikipedia.org/wiki/Disk_partitioning). When accessing a hard disk device, HBIOS will look for a partition with type id 0x2E and will use that partition exclusively for all storage. If a hard disk does not have a valid partition table with a partition of type 0x2E, the HBIOS will treat the hard disk as dedicated storage and will store data starting at the first sector of the disk.

The use of a partition of type 0x2E is preferred for RomWBW and is referred to as a “Modern” disk layout. If there is no RomWBW partition on the disk, then the disk is designated as having a “Classic” disk layout.

When a disk uses a RomWBW partition (type 0x2E) for storage (Modern layout), the CP/M filesystems on that disk will utilize a format with 1,024 directory entries per filesystem. If there is no RomWBW partition, the CP/M filesystems will have 512 directory entries per filesystem. As a result, the Modern disk layout with a RomWBW partition is also referred to as the “hd1k” layout indicating 1024 directory entries. Similarly, the Classic disk layout (no partition of type 0x2E) is also referred to as the “hd512” layout indicating 512 directory entries.

The layout type of any hard disk is simply dictated by the existence of a RomWBW partition. This also means that if you add or remove a partition table entry of type 0x2E on existing hard disk media, you will lose access to any pre-existing CP/M data on the disk. If used, partitioning should be done before putting any data on the disk.

WARNING: You **can not** mix the two hard disk layouts on one hard disk device. You can use different layouts on different hard disk devices in a single system though.

Regardless of whether a disk is Modern or Classic, RomWBW supports the concept of CP/M filesystem slices. In general, CP/M filesystems are limited to 8MB. Since current disk media is dramatically larger than this, RomWBW implements a mechanism to put many (up to 256) CP/M filesystems on a single disk. Each such filesystem is called a slice referring to the idea that the disk has been sliced into many independent CP/M filesystems. RomWBW allows the disk slices to be mapped to the limited (16) drive letters of CP/M. The mapping can be modified on-the-fly on a running system as desired.

If the case of a Modern disk layout (with a RomWBW partition), the slices are contained within the defined partition area and the number of slices is dictated by the size of the partition. In the case of a Classic disk layout (no RomWBW partition), the slices are located at the start of the disk (first sector). In either case, the slices are just sequential areas of space on the hard disk.

RomWBW accesses all hard disks using Logical Block Addressing (pure sector offset). When necessary, RomWBW simulates the following disk geometry for operating systems:

- Sector = 512 Bytes
- Track = 16 Sectors (8KB per Track)
- Cylinder = 16 Tracks (256 Sectors per Cylinder, 128KB per Cylinder)

If one is used, the FAT Partition must not overlap the CP/M slices. The FAT partition does not need to start immediately after the CP/M slices nor does it need to extend to the end of the hard disk. Its location and size are entirely determined by its corresponding partition table entry.

Drive letters in CP/M are ASSIGNed to the numbered slices as desired.
At boot, RomWBW automatically assigns up to 8 slices to drive letters starting with the first available drive letter (typically C:).

Microsoft Windows will assign a single drive letter to the FAT partition when the CF/SD Card is inserted. The drive letter assigned has no relationship to the CP/M drive letters assigned to CP/M slices.

In general, Windows, MacOS, or Linux know nothing about the CP/M slices and CP/M knows nothing about the FAT partition. However, the FAT application can be run under CP/M to access the FAT partition programmatically.

Before being used, A CP/M slice must be (re)initialized using the CP/M command CLRDIR. A CP/M slice can be made bootable by copying a system image to the System Area using SYSCOPY.

The FAT partition can be created from CP/M using the FDISK80 application. The FAT partition can be initialized using the FAT application from CP/M using the command `FAT FORMAT n:` where n is the RomWBW disk unit number containing the FAT partition to be formatted.

5.2.1 Modern Hard Disk Layout (hd1k)

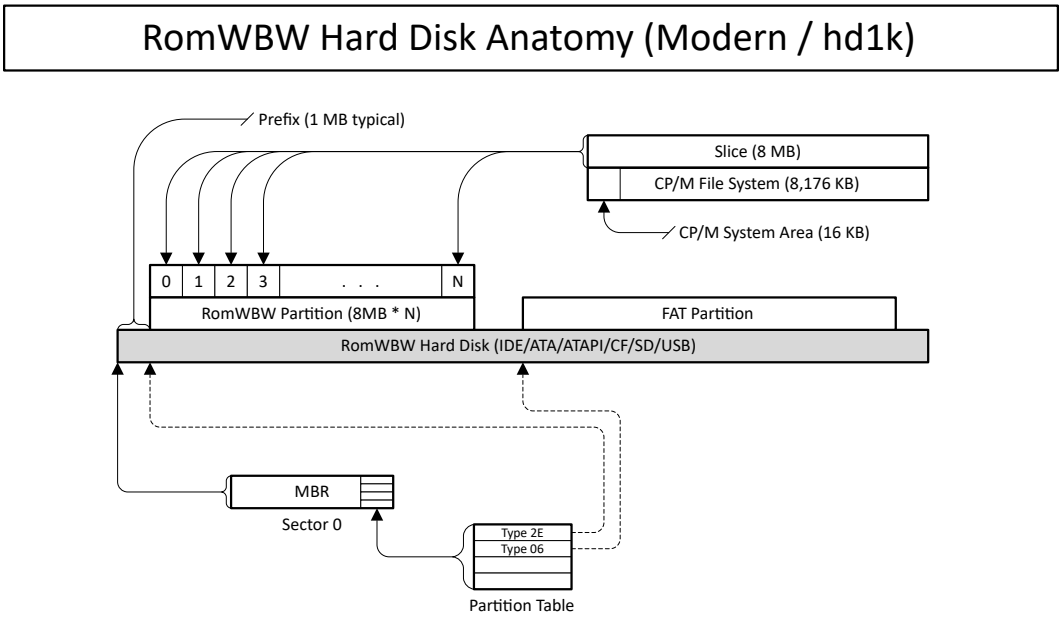

Figure 5.1: Modern Disk Layout

The CP/M filesystem on a Modern disk will accommodate 1,024 directory entries.

The CP/M slices reside entirely within a hard disk partition of type 0x2E. The number of slices is determined by the number of slices that fit within the partition spaces allocated up to the maximum of 256.

5.2.2 Classic Hard Disk Layout (hd512)

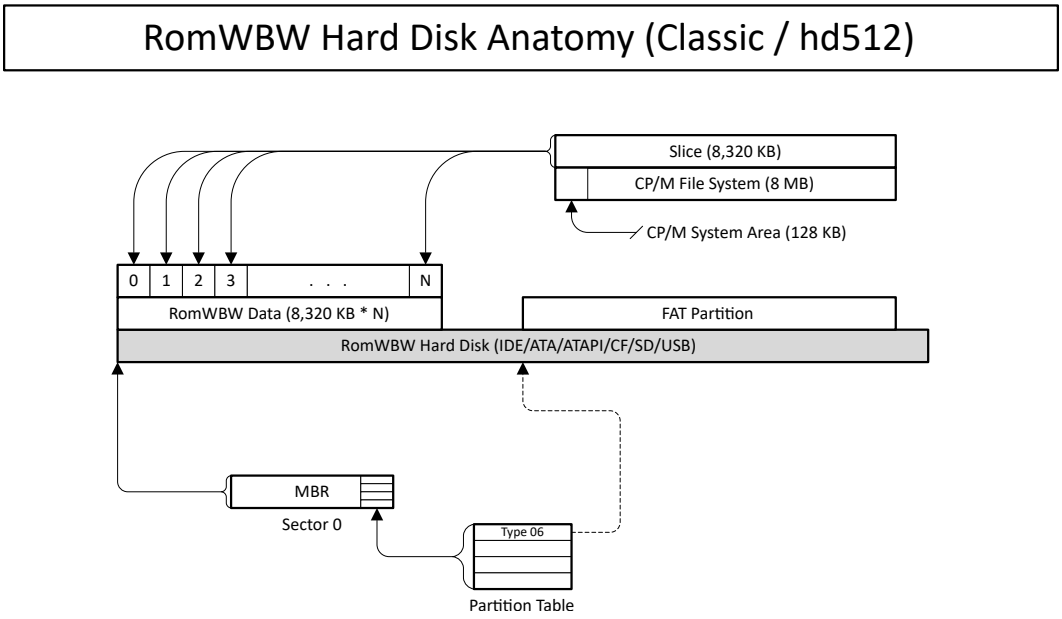

Figure 5.2: Classic Disk Layout

The CP/M filesystem on a Classic disk will accommodate 512 directory entries.

The CP/M slices reside on the hard disk starting at the first sector of the hard disk. The number of CP/M slices is not explicitly recorded anywhere on the hard disk. It is up to the system user to know how many slices are being used based on the size of the hard disk media and/or the start of a FAT partition.

A partition table may exist within the first sector of the first slice. For Classic disks, the partition table defines only the location and size of the FAT partition. The Partition Table does not control the location or number of CP/M slices in any way.

The Partition Table resides in a sector that is shared with the System Area of CP/M Slice 0. However, the RomWBW implementation of CP/M takes steps to avoid changing or corrupting the Partition Table area.

The FAT partition can be created from CP/M using the FDISK80 application. The user is responsible for ensuring that the start of the FAT partition does not overlap with the area they intend to use for CP/M slices. FDISK80 has a Reserve option to assist with this.

5.2.3 Mapping to Media ID

HBIOS has a definition of “Media ID”, which defines the type and physical properties of disk media provided by an underlying storage device. For a complete list of Media ID’s please see Disk Input/Output (DIO).

There are two important Media ID’s relating to Hard Disk Layouts:

Media	ID	Format / Meaning
MID_HD	4	Classic Disk Layout (hd512) –*and*– HBIOS Hard Disk Drive
MID_HDNEW	10	Modern Disk Layout (hd1k)

HBIOS typically does not understand the format of data on a device, instead just treating all hard disks as raw sectors. `MID_HD` is the typical Media ID used by HBIOS to describe high capacity hard disk media

When the Modern Disk Layout was added, the `MID_HDNEW`, was added to differentiate (at the operating system level) between the Classic and Modern layouts.

However HBIOS itself typically does NOT make this distinction, since the use of these two formats is determined by the operating system based on the partition table on the media. There are two important HBIOS functions that deal with Media ID.

- Function 0x18 – Disk Media (DIOMEDIA)
- Function 0xE0 – Calculate Slice (EXTSLICE)

Chapter 6

System Boot Process

A multi-phase boot strategy is employed. This is necessary because at cold start, the CPU is executing code from ROM in lower memory which is the same area that is bank switched.

RomWBW supports multiple boot techniques as described below. The most common of these is the ROM boot.

6.1 ROM Boot

The ROM boot process normally begins with a system cold start (power on or hardware reset). The hardware is responsible for ensuring that the lower 32K of CPU memory (bank window) is mapped to the initial 32K of the ROM. The Z80 CPU begins execution at address zero which will be address zero of the ROM.

The following steps occur during the ROM boot process:

1. The ROM code performs basic hardware initialization and ensures that the top 32K of CPU memory is mapped to the proper RAM bank.
2. The ROM code installs the HBIOS proxy code into the top 512 bytes of the CPU memory (0xFE00-0xFFFF).
3. Using the proxy code services, the full HBIOS code is copied from the ROM bank to the RAM bank that it will use for normal processing.
4. Again using the proxy code services, the RAM copy of HBIOS is activated in the bank window and execution transitions to the RAM copy of HBIOS.

5. The HBIOS initializes the system console so that output can now be displayed to the user.
6. The HBIOS now performs the full hardware discovery and initialization process while displaying it's progress.
7. The HBIOS displays a final summary of the hardware device unit assignments and various configuration information.
8. The HBIOS loads the RomWBW Boot Loader from ROM into RAM and jumps to it.

At this point, the user would normally use Boot Loader commands to select and launch an operating system or applications from either ROM or disk.

Note that the boot process is entirely operating system agnostic. It is unaware of the operating system being loaded. The Boot Loader prompts the user for the location of the binary image to load, but does not know anything about what is being loaded (the image is usually an operating system, but could be any executable code image). Once the Boot Loader has loaded the image at the selected location, it will transfer control to it. Assuming the typical situation where the image was an operating system, the loaded operating system will then perform its own initialization and begin normal operation.

6.2 Application Boot

Once the system is running (operating system loaded), it is possible to reboot the system from a system image (file) contained on the OS file system. This is referred to as an "Application Boot". The process is similar to a ROM boot, but the HBIOS code is loaded from an image file instead of ROM. This boot technique is useful to: 1) test a new build of a system image before programming it to the ROM; or 2) easily switch between system images on the fly.

During the RomWBW build process, one of the output files produced is an actual CP/M application (an executable .COM program file). Like the normal .ROM files, this file is placed in the Binary directory with the same name as the ROM file, but with the file extension of .ROM. Once you have a running CP/M (or compatible) system, you can upload/copy this application file to the filesystem. By executing this file, you will initiate an Application Boot using the system image contained in the application file itself.

Upon execution, the Application Boot program is loaded into memory by the previously running operating system starting at $0100. Note that the program image contains a full copy of the HBIOS to be installed and run. Once the Application Boot program is loaded by the previous operating system, control is passed to it and it performs a system initialization similar to the ROM Boot, but using the image loaded in RAM. Once the new HBIOS completes its initialization, it will launch the Boot Loader just like a ROM boot.

The Application Boot program actually contains two other components beyond the new HBIOS. It has a copy of the Boot Loader and a copy of the Z-System OS. This is done in case the new HBIOS requires updated versions of the Boot Loader or OS to run. The Boot Loader is aware of this boot mode and automatically adapts it's menu appropriately.

If you restart your system, then it will revert to a ROM Boot from the currently installed ROM.

6.3 RAM Boot

Some hardware supported by RomWBW has a special mechanism for loading the boot and HBIOS code. These systems have no ROM chips. However, they have a small hardware bootstrap that loads a chunk of code from a disk device directly into RAM at system startup.

The startup then proceeds very much like the Application Boot process described above. HBIOS is installed in its operating bank and control is passed to the Boot Loader.

6.4 Boot Recovery

To assist users when driver faults or mis-configuration causes a boot failure, RomWBW supports a limited recovery capability. This is achieved by allowing the user to reboot their machine, loading a minimal driver set. Implementation of this feature requires a hardware input "BOOT RECOVERY" button to be available and appropriate software configuration to be completed in the HBIOS.

When implemented, holding the "BOOT RECOVERY" button in after a reset or power cycle will cause the normal driver load process to be skipped in preference to a minimal set of drivers being loaded.

Typically this would be: Serial communication, RAM disk and parallel port IDE interface drivers.

Platforms supporting this option currently are the MBC, Duodyne and latter version of the SBC.

Chapter 7

Configuration

7.1 RomWBW NVRAM Configuration

On systems with RTC devices (that have Non-Volatile RAM), RomWBW supports storing some limited configuration option options inside this RAM.

Several configuration options are currently supported; these are referred to as Switches. In this case the term Switches refers to “soft” switches stored in NVRAM, not physical panel switches. The following switch ID’s are defined, and described in sections below.

Switch Number	Name	Description
0x00	-reserved-	Reserved
0x01	Boot Options	ROM or Disk Boot Settings
0x02	-n/a-	-n/a- high order byte of previous switch
0x03	Auto Boot	Automatically boot enabled without user input
0x04 - 0xFE	-future-	Future general usage
0xFF	Status Reset	Get Status or Reset Switches to Default

RomWBW uses bytes located at the start of RTC NVRAM, and includes a Parity check of the bytes in NVRAM to check for authenticity before using the configuration.

NVRAM Byte	Name	Description
0x00	Header Byte	Header Signature Byte ‘W’
0x01 - 0x03	Switch Data	Actual Switch Data
0x04	Parity Check	Checksum byte to check integrity

The above data is copied into the HBIOS Configuration Block (HCB) at startup at the location starting at CB_SWITCHES.

Although the switch data is stored in NVRAM, it is intended that you use SYSGET Subfunction 0xC0 – Get Switches (SWITCH) or SYSSET Subfunction 0xC0 – Set Switches (SWITCH) to read or write the switch values described here.

7.1.1 Boot Options (NVSW_BOOTOPTS)

16 bit Switch defining the ROM application or Disk device to boot if automatic booting is enabled.

Bit 15	Bits 14-8	Bits 7-0
1 = ROM App	-undefined-	App to Boot (Char)
0 = Disk	Disk Unit (0-127)	Disk Slice (0-255)

7.1.2 Auto Boot (NVSW_AUTOBOOT)

8 bit Switch defining if the system should auto boot at startup.

Bits 7-6	Bit 5	Bit 4	Bits 3-0
-unused-	1 = Auto Boot Enabled	-unused-	0 = Immediate Boot with no delay
-unused-	1 = Auto Boot Enabled	-unused-	(1-15) Timeout (seconds) before boot
-unused-	0 = Auto Boot Disabled	-unused-	-undefined-

7.1.3 Status Reset (0xFF)

The Status Reset switch is a virtual switch that does not have a corresponding stored value. It is a control mechanism to allow the global status of all switches to be determined. The meaning of the switch is different for Read (Get Status) and Write (Reset NVRAM)

GET (Get Status)

When the switch number 0xFF is read (using the Get Switches function), the status of the NVRAM switches will be returned as follows:

Status	A Register	Z / NZ Flag
NVRAM does not exist	A=0	NZ flag set

Status	A Register	Z / NZ Flag
NVRAM exists, but has not been initialised	A=1	NZ flag set
NVRAM exists, and has been fully initialised	A='W'	Z flag set

SET (Reset NVRAM)

When the switch number 0xFF is written (using the Set Switches function), the stored values of all switches will be reset to their default values. This will wipe any existing data and set default values into NVRAM.

Chapter 8

Driver Model

The framework code for bank switching also allows hardware drivers to be implemented mostly without concern for memory management. Drivers are coded to simply implement the HBIOS functions appropriate for the type of hardware being supported. When the driver code gets control, it has already been mapped to the CPU address space and simply performs the requested function based on parameters passed in registers. Upon return, the bank switching framework takes care of restoring the original memory layout expected by the operating system and application.

Drivers do need to be aware of the bank switching if a buffer address is being used in the function call.

- If the buffer address is in the lower 32K of RAM, then the memory it points to will be from the User Bank, not the HBIOS bank which is now active. In this case, the driver must use an inter-bank copy to access the data.
- If the buffer address is in the top 32K of RAM, then the driver will have access to it directly even after a bank switch, so no special steps are required.

For some functions, the location of the buffer is required to be in the top 32K of RAM to simplify the operation of the driver.

It is usually better if the OS or application calling a buffered function places the buffer in the top 32K because this may avoid a double-copy operation.

If driver code must make calls to other code, drivers, or utilities in the HBIOS bank, it must make those calls directly (it must not use RST 08). This is to avoid a nested bank switch which is not supported at this time.

Chapter 9

Character / Emulation / Video Services

In addition to a generic set of routines to handle typical character input/output, HBIOS also includes functionality for managing built-in video display adapters. To start with there is a basic set of character input/output functions, the CIOXXX functions, which allow for simple character data streams. These functions fully encompass routing byte stream data to/from serial ports. Note that there is a special character pseudo-device called “CRT”. When characters are read/written to/from the CRT character device, the data is actually passed to a built-in terminal emulator which, in turn, utilizes a set of VDA (Video Display Adapter) functions (such as cursor positioning, scrolling, etc.).

Figure 9.1 depicts the relationship between these components of HBIOS video processing:

Normally, the operating system will simply utilize the CIOXXX functions to send and receive character data. The Character I/O Services will route I/O requests to the specified physical device which is most frequently a serial port (such as UART or ASCI). As shown above, if the CRT device is targeted by a CIOXXX function, it will actually be routed to the Emulation Services which implement TTY, ANSI, etc. escape sequences. The Emulation Services subsequently rely on the Video Display Adapter Services as an additional layer of abstraction. This allows the emulation code to be completely unaware of the actual physical device (device independent). Video Display Adapter (VDA) Services contains drivers as needed to handle the available physical video adapters.

Note that the Emulation and VDA Services API functions are available to be called directly. Doing so must be done carefully so as to not corrupt the “state” of the emulation logic.

Before invoking CIOXXX functions targeting the CRT device, it is necessary that the underlying layers (Emulation and VDA) be properly initialized. The Emulation Services must be initialized to specify the desired emulation and specific physical VDA device to target. Likewise, the VDA

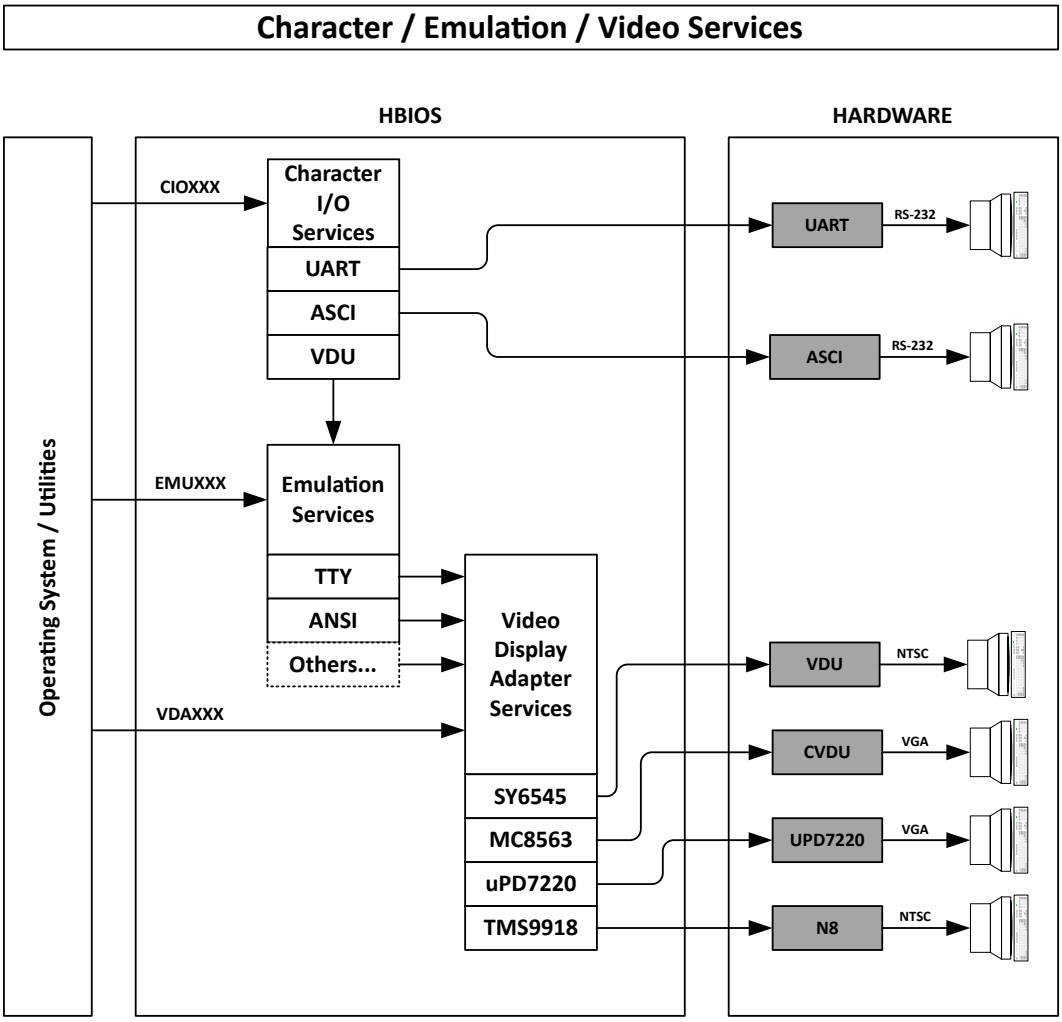

Figure 9.1: Character / Emulation / Video Services

Services may need to be initialized to put the specific video hardware into the proper mode, etc.

Chapter 10

HBIOS Reference

10.1 Invocation

HBIOS functions are invoked by placing the required parameters in CPU registers and executing an `RST 08` instruction. Note that HBIOS does not preserve register values that are unused. However, the values of the Z80 alternate registers and IX/IY will be preserved (these registers may be used within HBIOS, but will be saved and restored internally).

An alternate method of invoking HBIOS functions is to use `CALL $FFF0`. Since the `RST 08` vector exists in page zero of the CPU address space, it may be paged out when alternate memory banks are selected. If this may be true when you are invoking a function, you should use the `CALL` method.

Normally, applications will not call HBIOS functions directly. It is intended that the operating system makes all HBIOS function calls. Applications that are considered system utilities may use HBIOS, but must be careful not to modify the operating environment in any way that the operating system does not expect.

In general, the desired function is placed in the B register. Register C is frequently used to specify a sub-function or a target device unit number. Additional registers are used as defined by the specific function. Register A should be used to return function result information. See below for result code definitions.

The character, disk, and video device functions all refer to target devices using a logical device unit number that is passed in the C register. Keep in mind that these unit numbers are assigned dynamically at HBIOS initialization during the device discovery process. The assigned unit numbers are displayed on the console at the conclusion of device initialization. The unit assignments will never change after HBIOS initialization. However, they can change at the

next boot if there have been hardware or BIOS customization changes. Code using HBIOS functions should not assume fixed unit assignments.

Some functions utilize pointers to memory buffers. Unless otherwise stated, such buffers can be located anywhere in the Z80 CPU 64K address space. However, performance sensitive buffers (primarily disk I/O buffers) will require double-buffering if the caller's buffer is in the lower 32K of CPU address space. For optimal performance, such buffers should be placed in the upper 32K of CPU address space.

HBIOS also implements a small number of core functions in the HBIOS proxy area at the top of RAM. These exist primarily to facilitate the operation of normal HBIOS function calls. However, they are available to be used by OSes and applications. These functions can only be invoked by calling into a jump table in upper RAM.

10.2 Result Codes

The following function result codes are defined generically for all HBIOS functions. Most function calls will return a result in register A.

Code	Definition
0	function succeeded
-1	undefined error
-2	function not implemented
-3	invalid function
-4	invalid unit number
-5	out of memory
-6	parameter out of range
-7	media not present
-8	hardware not present
-9	I/O error
-10	write request to read-only media
-11	device timeout
-12	invalid configuration

10.3 Character Input/Output (CIO)

Character Input/Output functions require that a Character Unit number be specified in register C. This is the logical device unit number assigned during the boot process that identifies all character devices uniquely. A special value of 0x80 can be used for the Character Unit to refer to the current console device.

All character units are assigned a Device Type ID which indicates the specific hardware device driver that handles the unit. The table below enumerates these values.

Device Type	ID	Description	Driver
CIODEV_UART	0x00	16C550 Family Serial Interface	uart.asm
CIODEV_ASCI	0x01	Z180 Built-in Serial Ports	asci.asm
CIODEV_TERM	0x02	Terminal	ansi.asm
CIODEV_PRPCON	0x03	PropIO Serial Console Interface	prp.asm
CIODEV_PPPCON	0x04	ParPortProp Serial Console Interface	ppp.asm
CIODEV_SIO	0x05	Zilog Serial Port Interface	sio.asm
CIODEV_ACIA	0x06	MC68B50 Asynchronous Interface	acia.asm
CIODEV_PIO	0x07	Zilog Parallel Interface Controller	pio.asm
CIODEV_UF	0x08	FT232H-based ECB USB FIFO	uf.asm
CIODEV_DUART	0x09	SCC2681 Family Dual UART	duart.asm
CIODEV_Z2U	0x0A	Zilog Z280 Built-in Serial Ports	z2u.asm
CIODEV_LPT	0x0B	Parallel I/O Controller	lpt.asm
CIODEV_ESPCON	0x0C	ESP32 VGA Console	esp.asm
CIODEV_ESPSER	0x0D	ESP32 Serial Port	esp.asm
CIODEV_SCON	0x0E	S100 Console	scon.asm
CIODEV_SSER	0x0F	Simple Serial Console	sser.asm
CIODEV_EZ80UART	0x10	eZ80 Built-in UART0 Interface	ez80uart.asm

Character devices can usually be configured with line characteristics such as speed, framing, etc. A word value (16 bit) is used to describe the line characteristics as indicated below:

Bits	Characteristic
15-14	Reserved (set to 0)
13	RTS
12-8	Baud Rate (see below)
7	DTR
6	XON/XOFF Flow Control
5	1 = Stick Parity(Mark/Space), 0 = Normal Parity (odd/even)

Bits	Characteristic
4	1 = Even/Space, 0 = Odd/Mark
3	Parity Enable (set for true)
2	Stop Bits (set for true)
1-0	Data Bits (5-8 encoded as 0-3)

The 5-bit Baud Rate value (V) is encoded as V = 75 * 2^X * 3^Y. The bits are defined as YXXXX.

Actual character values are a single byte (8 bits). The Character I/O functions do not modify or interpret the values being sent/received so they can be used to pass 8-bit binary data without corruption. Note that some OSes will modify character data (truncate to 7 bits, etc.).

10.3.1 Function 0x00 – Character Input (CIOIN)

Entry Parameters	Returned Values
B: 0x00	A: Status
C: Character Unit	E: Character

Read and return a Character (E) from the specified Character Unit (C). If no character(s) are available in the unit's input buffer, this function will wait indefinitely. The returned Status (A) is a standard HBIOS result code.

10.3.2 Function 0x01 – Character Output (CIOOUT)

Entry Parameters	Returned Values
B: 0x01 C: Character Unit E: Character	A: Status (0-OK, else error)

Send a Character (E) via the specified Character Unit (C). If there is no space available in the unit's output buffer, the function will wait indefinitely. The returned Status (A) is a standard HBIOS result code.

10.3.3 Function 0x02 – Character Input Status (CIOIST)

Entry Parameters	Returned Values
B: 0x02 C: Character Unit	A: Status / Characters Pending

Return the count of Characters Pending (A) in the input buffer of the specified Character Unit (C). If the unit has no input buffer or the buffer utilization is not available, the function may return simply 0 or 1 where 0 means there is no character available and 1 means there is at least one character available.

The value returned in register A is used as both a Status (A) code and the return value. Negative values (bit 7 set) indicate a standard HBIOS result (error) code. Otherwise, the return value represents the number of characters in the input buffer.

10.3.4 Function 0x03 – Character Output Status (CIOOST)

Entry Parameters	Returned Values
B: 0x03 C: Character Unit	A: Status / Space Free

Return the count of buffer Space Free (A) for the specified Character Unit (C). For example, if a 16 byte output buffer contains 6 characters waiting to be sent out the unit's serial interface, this function would return 10; the number of positions available in the output buffer. If the port has no output buffer or the buffer utilization is not available, the function may return simply 0 or 1 where 0 means there is no buffer space available and 1 means there is space in the output buffer for at least one character.

The return value in register A is used as both a status code and the return value. Negative values (bit 7 set) indicate a standard HBIOS result (error) code. Otherwise, the return value represents the buffer space available.

10.3.5 Function 0x04 – Character I/O Initialization (CIOINIT)

Entry Parameters	Returned Values
B: 0x04 C: Character Unit DE: Line Characteristics	A: Status

Condition the interface of the specified Character Unit (C) according to the specified Line Characteristics (DE). The definition of the line characteristics value is described above. If DE contains -1 (0xFFFF), then the device will be reinitialized with the previous line characteristics used (a reset) and any buffer contents will be flushed. The Status (A) is a standard HBIOS result code.

Not all line characteristics are supported by all character interfaces. It is up to the driver of the character unit to decide how to deal with characteristics that are not available. For example, many character drivers do not allow flow control settings (RTS/CTS, XON/XOFF) to be modified dynamically. In most cases, these settings are ignored by the driver in this function call.

10.3.6 Function 0x05 – Character I/O Query (CIOQUERY)

Entry Parameters	Returned Values
B: 0x05	A: Status
C: Character Unit	DE: Line Characteristics

Returns the current Line Characteristics (DE) of the specified Character Unit (C). The definition of the line characteristics value is described above. The returned status (A) is a standard HBIOS result code.

10.3.7 Function 0x06 – Character I/O Device (CIODEVICE)

Entry Parameters	Returned Values
B: 0x06	A: Status
C: Character Unit	C: Device Attributes
	D: Device Type
	E: Device Number
	H: Device Mode
	L: Device I/O Base Address

Returns device information for the specified Character Unit (C). The status (A) is a standard HBIOS result code.

The two high bits of Device Attribute (C) are: 00 = RS/232, 01 = Terminal, 10 = Parallel. The remaining bits should be ignored and are used internally.

Device Type (D) indicates the specific hardware driver that handles the specified Character Unit. Values are listed at the start of this section. Device Number (E) indicates the physical

device number assigned per driver. For example, a Device Type of 0x50 with a Device Number of 2 refers to the third port being handled by the SIO driver.

Device Mode (H) is used to indicate the variant of the chip or circuit that is used by the specified unit. For example, for a UART, the value indicates the chip variant. The Device I/O Base Address (L) indicates the starting port address of the hardware interface that is servicing the specified unit. Both of these values are considered driver specific. Refer to the associated hardware driver for the values used.

10.4 Disk Input/Output (DIO)

Disk Input/Output functions require that a Disk Unit number be specified in register C. This is the logical device unit number assigned during the boot process that identifies all disk devices uniquely.

All character units are assigned a Device Type ID which indicates the specific hardware device driver that handles the unit. The table below enumerates their values.

Device Type	ID	Description	Driver
DIODEV_MD	0x00	Memory Disk	md.asm
DIODEV_FD	0x01	Floppy Disk	fd.asm
DIODEV_RF	0x02	RAM Floppy	rf.asm
DIODEV_IDE	0x03	IDE Disk	ide.asm
DIODEV_ATAPI	0x04	ATAPI Disk (not implemented)	
DIODEV_PPIDE	0x05	PPIDE Disk	ppide.asm
DIODEV_SD	0x06	SD Card	sd.asm
DIODEV_PRPSD	0x07	PropIO SD Card	prp.asm
DIODEV_PPPSD	0x08	ParPortProp SD Card	ppp.asm
DIODEV_HDSK	0x09	SIMH HDSK Disk	hdsk.asm
DIODEV_PPA	0x0A	Iomega PPA Disk	ppa.asm
DIODEV_IMM	0x0B	Iomega IMM Disk	imm.asm
DIODEV_SYQ	0x0C	Syquest Sparq Disk	syq.asm
DIODEV_CHUSB	0x0D	CH375/376 USB Disk	ch.asm
DIODEV_CHSD	0x0E	CH375/376 SD Card	ch.asm
DIODEV_USB	0x0F	CH376 Native USB Device	ch376.asm
DIODEV_ESPSD	0x10	S100 ESP32 SD Card	espsd.asm

A fixed set of media types are defined. The currently defined media types identifiers are listed below. Each driver will support one or more of the defined media types.

Media	ID	Format
MID_NONE	0	No media installed
MID_MDROM	1	ROM Drive
MID_MDRAM	2	RAM Drive
MID_RF	3	RAM Floppy (LBA)
MID_HD	4	Hard Disk (LBA) w/ 512 directory entries
MID_FD720	5	3.5” 720K Floppy
MID_FD144	6	3.5” 1.44M Floppy

Media	ID	Format
MID_FD360	7	5.25” 360K Floppy
MID_FD120	8	5.25” 1.2M Floppy
MID_FD111	9	8” 1.11M Floppy
MID_HDNEW	10	Hard Disk (LBA) w/ 1024 directory entries

NOTE: HBIOS typically does not actually differentiate between MID_HD and MID_HDNEW, it will generally only use MID_HD. See the section Mapping to Media ID for information on this.

HBIOS supports both Cylinder/Head/Sector (CHS) and Logical Block Addresses (CHS) when locating a sector for I/O (see DIOSEEK function). For devices that are natively CHS (e.g., floppy disk), the HBIOS driver can convert LBA values to CHS values according to the geometry of the current media. For devices that are natively LBA (e.g., hard disk), the HBIOS driver simulates CHS using a fictitious geometry provided by the driver (typically 16 sectors per track and 16 heads per cylinder).

10.4.1 Function 0x10 – Disk Status (DIOSTATUS)

Entry Parameters	Returned Values
B: 0x10 C: Disk Unit	A: Status

Returns the driver specific Status (A) of the specified disk device unit (C) based on the last operation performed.

The return value in register A is used as both a device status and a standard HBIOS result code. Negative values (bit 7 set) indicate a standard HBIOS result (error) code. Otherwise, the return value represents a driver-specific device status. In all cases, the value 0 means OK.

10.4.2 Function 0x11 – Disk Reset (DIORESET)

Entry Parameters	Returned Values
B: 0x11 C: Disk Unit	A: Status

This function performs a device dependent reset operation on the Disk Unit specified (C). The driver will clear any error status on the disk unit, attempt to reset the interface, and flag the

disk unit for initialization on the next I/O function call. Any prior media identification will be cleared. The returned Status (A) is a standard HBIOS result code.

If the specified disk unit (C) is one of multiple units on a single hardware bus, then all units on that bus will be reset. For example, if the master disk on an IDE bus is reset, then the slave disk will also be reset.

10.4.3 Function 0x12 – Disk Seek (DIOSEEK)

Entry Parameters	Returned Values
B: 0x12 C: Disk Unit DEHL: Sector Address	A: Status

This function will set the desired sector to be used for the next I/O operation on the specified Disk Unit (C). The returned Status (A) is a standard HBIOS result code.

An actual seek operation is generally not performed on the disk hardware by this function. The function typically just records the sector address for subsequent I/O function calls.

The double-word Sector Address (DEHL) can represent either a Logical Block Address (LBA) or a Cylinder/Head/Sector (CHS). Bit 7 of D is set (1) for LBA mode and cleared (0) for CHS mode.

For LBA mode operation, the high bit is set and the rest of the double-word is then treated as the logical sector address.

For CHS mode operation, the Sector Address (DEHL) registers are interpreted as: D=Head, E=Sector, and HL=Track. All values (including sector) are 0 relative.

Prior versions of the floppy driver did not accept LBA mode addresses. However, this restriction has been removed as of HBIOS v3.1. At this point, all disk drivers support both LBA and CHS addressing.

10.4.4 Function 0x13 – Disk Read (DIOREAD)

Entry Parameters	Returned Values
B: 0x13 C: Disk Unit D: Buffer Bank ID	A: Status E: Sectors Read

Entry Parameters	Returned Values
E: Sector Count HL: Buffer Address	

Read Sector Count (E) sectors into the buffer located in Buffer Bank ID (D) at Buffer Address (HL) starting at the Current Sector. The returned Status (A) is a standard HBIOS result code.

The Current Sector is established by a prior DIOSEEK function call; however, multiple read/write/verify function calls can be made after a seek function. The Current Sector is incremented after each sector successfully read. On error, the Current Sector will be the sector where the error occurred. Sectors Read (E) indicates the number of sectors successfully read.

The caller must ensure that the Buffer Address is large enough to contain all sectors requested. Disk data transfers will be faster if the buffer resides in the top 32K of memory because it avoids a double buffer copy.

Also for buffers in the top 32K of memory the Bank ID is not strictly required as this memory is alway mapped to the common bank. For buffers in the bottom 32KB ram, the Bank ID is used to identify the bank to use for the buffer. If you do not wih to use banked memory you will need to provide the current Bank ID, which can be obtained using Function 0xF3 – System Get Bank (SYSGETBNK)

10.4.5 Function 0x14 – Disk Write (DIOWRITE)

Entry Parameters	Returned Values
B: 0x14 C: Disk Unit D: Buffer Bank ID E: Sector Count HL: Buffer Address	A: Status E: Sectors Written

Write Sector Count (E) sectors from the buffer located in Buffer Bank ID (D) at Buffer Address (HL) starting at the Current Sector. The returned Status (A) is a standard HBIOS result code.

The Current Sector is established by a prior DIOSEEK function call; however, multiple read/write/verify function calls can be made after a seek function. The Current Sector is incremented after each sector successfully written. On error, the Current Sector will be the sector where the error occurred. Sectors Written (E) indicates the number of sectors successfully written.

Disk data transfers will be faster if the buffer resides in the top 32K of memory because it avoids a double copy.

10.4.6 Function 0x15 – Disk Verify (DIOVERIFY)

Entry Parameters	Returned Values
B: 0x15 C: Disk Unit E: Sector Count	A: Status E: Sectors Verified

***** Function Not Implemented *****

10.4.7 Function 0x16 – Disk Format (DIOFORMAT)

Entry Parameters	Returned Values
B: 0x16 C: Disk Unit D: Head E: Fill Byte HL: Cylinder	A: Status

***** Function Not Implemented *****

10.4.8 Function 0x17 – Disk Device (DIODEVICE)

Entry Parameters	Returned Values
B: 0x17 C: Disk Unit	A: Status C: Device Attributes D: Device Type E: Device Number H: Device Unit Mode L: Device I/O Base Address

Reports device information about the specified Disk Unit (C). The Status (A) is a standard HBIOS result code.

The Device Attribute (C) value returned indicates various feature indicators related to the device being referenced by the specified Disk Unit (C). The high 3 bits apply to all devices. The definition of the low 5 bits depends on whether the device is a Floppy (indicated by bit 5).

The common bits are:

Bits	Definition
7	Floppy
6	Removable
5	High Capacity (>8 MB)

The Floppy specific bits are:

Bits	Definition
4-3	Form Factor: 0=8”, 1=5.25”, 2=3.5”, 3=Other
2	Sides: 0=SS, 1=DS
1-0	Density: 0=SD, 1=DD, 2=HD, 3=ED

The non-Floppy specific bits are:

Bits	Definition
4	LBA Capable
3-0	Media Type: 0=Hard Disk, 1=CF, 2=SD, 3=USB, 4=ROM, 5=RAM, 6=FLASH, 7=RAMF, 8=CD-ROM, 9=Cartridge

Device Type (D) indicates the specific hardware driver that handles the specified Disk Unit (C). Values are listed at the start of this section. Device Number (E) indicates the physical device number assigned per driver. For example, a Device Type of 0x30 with a Device Number of 1 refers to the second disk being handled by the IDE driver.

Device Mode (H) is used to indicate the variant of the chip or circuit that is used by the specified unit. For example, for an IDE unit, the value indicates the IDE circuit variant. The Device I/O Base Address (L) indicates the starting port address of the hardware interface that is servicing the specified unit. Both of these values are considered driver specific. Refer to the associated hardware driver for the values used.

10.4.9 Function 0x18 – Disk Media (DIOMEDIA)

Entry Parameters	Returned Values
B: 0x18 C: Disk Unit E: Flags	A: Status E: Media ID

Report the Media ID (E) for the for media in the specified Disk Unit (C). If bit 0 of Flags (E) is set, then media discovery or verification will be performed. The Status (A) is a standard HBIOS result code. If there is no media in device, function will return an error status.

NOTE: This function will always return MID_HD for hard disk devices. See the section Mapping to Media ID for information on this. To determine if an HD1K formatted partition exists on the hard disk please see the following function.

Function 0xE0 – Calculate Slice (EXTSLICE)

10.4.10 Function 0x19 – Disk Define Media (DIODEFMED)

Entry Parameters	Returned Values
B: 0x19 C: Disk Unit E: Media ID	A: Status

***** Function Not Implemented *****

10.4.11 Function 0x1A – Disk Capacity (DIOCAPACITY)

Entry Parameters	Returned Values
B: 0x1A C: Disk Unit	A: Status DEHL: Sector Count BC: Block Size

Report the current media capacity information for the specified Disk Unit (C). The Sector Count (DEHL) is a double-word number representing the total number of blocks on the device. Block Size (BC) contains the block size in bytes. The Status (A) is a standard HBIOS result code. If the media is unknown, an error will be returned.

This function will not attempt to discover or verify the media loaded in the unit specified. You can use precede this function with the DIOMEDIA function to force this if desired.

10.4.12 Function 0x1B – Disk Geometry (DIOGEOMETRY)

Entry Parameters	Returned Values
B: 0x1B	A: Status
C: Disk Unit	D: Heads / LBA
	E: Sectors
	HL: Cylinder Count
	BC: Block Size

Report the geometry for the media in the specified Disk Unit (C). If a device uses LBA mode addressing natively, then the drivers simulated geometry will be returned. The Status (A) is a standard HBIOS result code. If the media is unknown, an error will be returned.

LBA capability is indicated by D:7. When set, the device is capable of LBA addressing. Refer to Function 0x12 – Disk Seek (DIOSEEK) for more information on specifying LBA vs. CHS addresses.

Heads (D:6-0) refers to the number of heads per cylinder. Sectors (E) refers to the number of sectors per track. Cylinder Count (HL) is the total number of cylinders addressable for the media. Block Size (BC) is the number of bytes in one sector.

10.5 Real Time Clock (RTC)

The Real Time Clock functions provide read/write access to the clock and related Non-Volatile RAM.

HBIOS only supports a single RTC device since there is no reason to have more than one at a time. The RTC unit is assigned a Device Type ID which indicates the specific hardware device driver that handles the unit. The table below enumerates these values.

Device Type	ID	Description	Driver
RTCDEV_DS	0x00	Maxim DS1302 Real-Time Clock w/ NVRAM	dsrtc.asm
RTCDEV_BQ	0x01	BQ4845P Real Time Clock	bqrtc.asm
RTCDEV_SIMH	0x02	SIMH Simulator Real-Time Clock	simrtc.asm
RTCDEV_INT	0x03	Interrupt-based Real Time Clock	intrtc.asm
RTCDEV_DS7	0x04	Maxim DS1307 PCF I2C RTC w/ NVRAM	ds7rtc.asm
RTCDEV_RP5	0x05	Ricoh RPC01A Real-Time Clock w/ NVRAM	rp5rtc.asm
RTCDEV_EZ80	0x07	eZ80 on-chip RTC	ez80rtc.asm
RTCDEV_PC	0x08	MC146818/DS1285/DS12885 RTC w/ NVRAM	pcrtc.asm

The time functions to get and set the time (RTCGTM and RTCSTM) require a 6 byte date/time buffer in the following format. Each byte is BCD encoded.

Offset	Contents
0	Year (00-99)
1	Month (01-12)
2	Date (01-31)
3	Hours (00-24)
4	Minutes (00-59)
5	Seconds (00-59)

10.5.1 Function 0x20 – RTC Get Time (RTCGETTIM)

Entry Parameters	Returned Values
B: 0x20 HL: Date/Time Buffer Address	A: Status

Read the current value of the real-time clock and store the date/time in the Date/Time Buffer pointed to by HL. The Status (A) is a standard HBIOS result code.

10.5.2 Function 0x21 – RTC Set Time (RTCSETTIM)

Entry Parameters	Returned Values
B: 0x21 HL: Date/Time Buffer Address	A: Status

Set the current value of the real-time clock based on the Date/Time Buffer pointed to by HL. The Status (A) is a standard HBIOS result code.

10.5.3 Function 0x22 – RTC Get NVRAM Byte (RTCGETBYT)

Entry Parameters	Returned Values
B: 0x22 C: Index	A: Status E: Value

Read a single byte Value (E) from the Non-Volatile RAM of the RTC at the byte offset Index (C). The Status (A) is a standard HBIOS result code.

10.5.4 Function 0x23 – RTC Set NVRAM Byte (RTCSETBYT)

Entry Parameters	Returned Values
B: 0x23 C: Index E: Value	A: Status

Set a single byte Value (E) of the Non-Volatile RAM of the RTC at the byte offset Index (C). The Status (A) is a standard HBIOS result code.

10.5.5 Function 0x24 – RTC Get NVRAM Block (RTCGETBLK)

Entry Parameters	Returned Values
B: 0x24 HL: Buffer Address	A: Status

Read the entire contents of the Non-Volatile RAM into to a buffer pointed to by Buffer Address (HL). The Status (A) is a standard HBIOS result code.

10.5.6 Function 0x25 – RTC Set NVRAM Block (RTCSETBLK)

Entry Parameters	Returned Values
B: 0x25 HL: Buffer Address	A: Status

Write the entire contents of the Non-Volatile RAM from the buffer pointed to by Buffer Address (HL). The Status (A) is a standard HBIOS result code.

10.5.7 Function 0x26 – RTC Get Alarm (RTCGETALM)

Entry Parameters	Returned Values
B: 0x26 HL: Date/Time Buffer Address	A: Status

Work in progress, documentation required…

10.5.8 Function 0x27 – RTC Set Alarm (RTCSETALM)

Entry Parameters	Returned Values
B: 0x27 HL: Date/Time Buffer Address	A: Status

Work in progress, documentation required…

10.5.9 Function 0x28 – RTC DEVICE (RTCDEVICE)

Entry Parameters	Returned Values
B: 0x28	A: Status C: Device Attributes D: Device Type E: Device Number H: Device Unit Mode L: Device I/O Base Address

Returns device information for the RTC unit. The Status (A) is a standard HBIOS result code.

Device Attributes (C) values are not yet defined. Device Type (D) indicates the specific hardware driver that handles the RTC unit. Values are listed at the start of this section. Device Number (E) indicates the physical device number assigned per driver which is always 0 for RTC.

Device Mode (H) is used to indicate the variant of the chip or circuit that is used by the specified unit. The Device I/O Base Address (L) indicates the starting port address of the hardware interface that is servicing the specified unit. Both of these values are considered driver specific. Refer to the associated hardware driver for the values used.

10.6 Display Keypad (DSKY)

The Display Keypad functions provide access to a segment or LCD style display and associated optional keypad

HBIOS only supports a single DSKY device since there is no reason to have more than one at a time. If the system contains multiple DSKY devices, only the first device discovered will be used. The DSKY unit is assigned a Device Type ID which indicates the specific hardware device driver that handles the unit. The table below enumerates these values.

Device Type	ID	Description	Driver
DSKYDEV_ICM	0x01	Original ICM7218 based DSKY	icm.asm
DSKYDEV_PKD	0x02	Next Gen Intel P8279 based DSKY	pkd.asm
DSKYDEV_GM7303	0x03	GM7303 LCD Display + Keypad	gm7303.asm
DSKYDEV_LCD	0x04	HD44780-based LCD Display	lcd.asm

The keypad keys are identified by the following key ids. Not all keypads will contain all keys.

Key Id	Key Definition	Key Id	Key Definition
$00	Hex Numeric 0	$10	Forward
$01	Hex Numeric 1	$11	Backward
$02	Hex Numeric 2	$12	Clear
$03	Hex Numeric 3	$13	Enter
$04	Hex Numeric 4	$14	Deposit
$05	Hex Numeric 5	$15	Examine
$06	Hex Numeric 6	$16	Go
$07	Hex Numeric 7	$17	Boot
$08	Hex Numeric 8	$18	F4
$09	Hex Numeric 9	$19	F3
$0A	Hex Numeric A	$1A	F2
$0B	Hex Numeric B	$1B	F1
$0C	Hex Numeric C		
$0D	Hex Numeric D		
$0E	Hex Numeric E		
$0F	Hex Numeric F		

10.6.1 Function 0x30 – DSKY Reset (DSKYRESET)

Entry Parameters	Returned Values
B: 0x30	A: Status

This function performs a device dependent reset operation on the DSKY. The display will be cleared, keyboard queue will be flushed, and chip will be reinitialized. The returned Status (A) is a standard HBIOS result code.

10.6.2 Function 0x31 – DSKY (DSKYSTATUS)

Entry Parameters	Returned Values
B: 0x31	A: Status / Characters Pending

Return the count of Characters Pending (A) in the input buffer of the DSKY. If the unit has no input buffer or the buffer utilization is not available, the function may return simply 0 or 1 where 0 means there is no character available and 1 means there is at least one character available.

The value returned in register A is used as both a Status (A) code and the return value. Negative values (bit 7 set) indicate a standard HBIOS result (error) code. Otherwise, the return value represents the number of characters in the buffer.

10.6.3 Function 0x32 – DSKY Get Key (DSKYGETKEY)

Entry Parameters	Returned Values
B: 0x32	A: Status E: Character Value

Read and return a Character (E) from the DSKY. If no character(s) are available in the unit's input buffer, this function will wait indefinitely. The returned Status (A) is a standard HBIOS result code.

The Character Value (E) returned is not ASCII. It is a keypad key id. The possible id values are listed at the start of this section.

10.6.4 Function 0x33 – DSKY Show HEX (RTCSHOWHEX)

Entry Parameters	Returned Values
B: 0x33 DE:HL=Binary Value	A: Status

Display the 32-bit binary value (DE:HL) in hex on the DSKY segment display. All decimal points of the display will be off. The Status (A) is a standard HBIOS result code.

10.6.5 Function 0x34 – DSKY Show Segments (DSKYSHOWSEG)

Entry Parameters	Returned Values
B: 0x34 HL: Buffer Address	A: Status

Display the segment-encoded values on the segment display. The encoding uses a small alphabet as defined below. The actual representation of a character is determined by the driver. The entire display is updated and it is assumed that an 8 character buffer will be pointed to by HL. The buffer must reside in high memory. The Status (A) is a standard HBIOS result code.

0x00: '0'	0x01: '1'	0x02: '2'	0x03: '3'
0x04: '4'	0x05: '5'	0x06: '6'	0x07: '7'
0x08: '8'	0x09: '9'	0x0A: 'A'	0x0B: 'B'
0x0C: 'C'	0x0D: 'D'	0x0E: 'E'	0x0F: 'F'
0x10: ' '	0x11: '-'	0x12: '.'	0x13: 'p'
0x14: 'o'	0x15: 'r'	0x16: 't'	0x17: 'A'
0x18: 'd'	0x19: 'r'	0x1A: 'G'	

10.6.6 Function 0x35 – DSKY Keypad LEDs (DSKYKEYLEDS)

Entry Parameters	Returned Values
B: 0x35 HL: Buffer Address	A: Status

Light the LEDs for the keypad keys according to the bitmap contained in the buffer pointed to by HL. The buffer must be located in high memory and is assumed to be 8 bytes.

At this time, the bitmap is specific to the PKD hardware and will be ignored by all other hardware.

10.6.7 Function 0x36 – DSKY Status LED (DSKYSTATLED)

Entry Parameters	Returned Values
B: 0x36 D: LED Number E: LED State	A: Status

Set or clear the status LED specified in D. The state of the LED is contained in E. If E=0, the LED will be turned off. If E=1, the LED will be turned on.

This function is specific to the PKD hardware and will be ignored by all other hardware. The Status (A) is a standard HBIOS result code.

10.6.8 Function 0x37 – DSKY Beep (DSKYBEEP)

Entry Parameters	Returned Values
B: 0x37	A: Status

Beep the onboard speaker of the DSKY. This function is specific to the PKD hardware. It will be ignored by the ICM hardware. The Status (A) is a standard HBIOS result code.

10.6.9 Function 0x38 – DSKY Device (DSKYDEVICE)

Entry Parameters	Returned Values
B: 0x38	A: Status C: Device Attributes D: Device Type E: Device Number H: Device Unit Mode L: Device I/O Base Address

Returns device information for the DSKY unit. The Status (A) is a standard HBIOS result code.

Device Attribute (C) values are not yet defined. Device Type (D) indicates the specific hardware driver that handles the specified character unit. Values are listed at the start of this section. Device Number (E) indicates the physical device number assigned per driver which is always 0 for DSKY.

Device Mode (H) is used to indicate the variant of the chip or circuit that is used by the specified unit. The Device I/O Base Address (L) indicates the starting port address of the hardware interface that is servicing the specified unit. Both of these values are considered driver specific. Refer to the associated hardware driver for the values used.

10.6.10 Function 0x39 – DSKY Device (DSKYMESSAGE)

Entry Parameters	Returned Values
B: 0x39 C: Message ID	A: Status

Instructs the display to show a textual representation of the associated message on the display. The IDs are defined in std.asm.

10.6.11 Function 0x3A – DSKY Device (DSKYEVENT)

Entry Parameters	Returned Values
B: 0x3A C: Event ID	A: Status

Instructs the display to update itself in response to an internal HBIOS state change. At this time the the events are:

0: CPU Speed Change
1: Disk Activity

10.7 Video Display Adapter (VDA)

The VDA functions are provided as a common interface to Video Display Adapters. Not all VDAs will include keyboard hardware. In this case, the keyboard functions should return a failure status.

All video units are assigned a Device Type ID which indicates the specific hardware device driver that handles the unit. The table below enumerates their values.

Device Type	ID	Description	Driver
VDADEV_VDU	0x00	MC6845 Family Video Display Controller	vdu.asm
VDADEV_CVDU	0x01	MC8563-based Video Display Controller	cvdu.asm
VDADEV_GDC	0x02	uPD7220 Video Display Controller	gdc.asm
VDADEV_TMS	0x03	TMS9918/38/58 Video Display Controller	tms.asm
VDADEV_VGA	0x04	HD6445CP4-based Video Display Controller	vga.asm
VDADEV_VRC	0x05	VGARC	vrc.asm
VDADEV_EF	0x06	EF9345	ef.asm
VDADEV_FV	0x07	S100 FPGA VGA	fv.asm
VDADEV_XOSERA	0x08	Xosera FPGA-based Video Display Controller	xosera.asm

Depending on the capabilities of the hardware, the use of colors and attributes may or may not be supported. If the hardware does not support these capabilities, they will be ignored.

Color byte values are constructed using typical RGBI (Red/Green/Blue/Intensity) bits. The high four bits of the value determine the background color and the low four bits determine the foreground color. This results in 16 unique color values for both foreground and background. The following table illustrates the color byte value construction:

	Bit	Color
Background	7	Intensity
	6	Blue
	5	Green
	4	Red
Foreground	3	Intensity
	2	Blue
	1	Green
	0	Red

The following table illustrates the resultant color for each of the possible 16 values for foreground or background:

Foreground	Background	Color
n0 nnnn0000	0n 0000nnnn	Black
n1 nnnn0001	1n 0001nnnn	Red
n2 nnnn0010	2n 0010nnnn	Green
n3 nnnn0011	3n 0011nnnn	Brown
n4 nnnn0100	4n 0100nnnn	Blue
n5 nnnn0101	5n 0101nnnn	Magenta
n6 nnnn0110	6n 0110nnnn	Cyan
n7 nnnn0111	7n 0111nnnn	White
n8 nnnn1000	8n 1000nnnn	Gray
n9 nnnn1001	9n 1001nnnn	Light Red
nA nnnn1010	An 1010nnnn	Light Green
nB nnnn1011	Bn 1011nnnn	Yellow
nC nnnn1100	Cn 1100nnnn	Light Blue
nD nnnn1101	Dn 1101nnnn	Light Magenta
nE nnnn1110	En 1110nnnn	Light Cyan
nF nnnn1111	Fn 1111nnnn	Bright White

Attribute byte values are constructed using the following bit encoding:

Bit	Effect
7	n/a (0)
6	n/a (0)
5	n/a (0)
4	n/a (0)
3	n/a (0)
2	Reverse
1	Underline
0	Blink

The following codes are returned by a keyboard read to signify non-ASCII keystrokes:

Value	Keystroke	Value	Keystroke
0xE0	F1	0xF0	Insert
0xE1	F2	0xF1	Delete
0xE2	F3	0xF2	Home
0xE3	F4	0xF3	End

Value	Keystroke	Value	Keystroke
0xE4	F5	0xF4	PageUp
0xE5	F6	0xF5	PadeDown
0xE6	F7	0xF6	UpArrow
0xE7	F8	0xF7	DownArrow
0xE8	F9	0xF8	LeftArrow
0xE9	F10	0xF9	RightArrow
0xEA	F11	0xFA	Power
0xEB	F12	0xFB	Sleep
0xEC	SysReq	0xFC	Wake
0xED	PrintScreen	0xFD	Break
0xEE	Pause	0xFE	
0xEF	App	0xFF	

10.7.1 Function 0x40 – Video Initialize (VDAINI)

Entry Parameters	Returned Values
B: 0x40 C: Video Unit E: Video Mode HL: Font Bitmap	A: Status

Performs a full (re)initialization of the specified Video Unit (C). The screen is cleared and the keyboard buffer is flushed. If the specified Video Unit (C) supports multiple video modes, a Video Mode (E) can be specified (set to 0 for default/not specified). Video Mode (E) values are specific to each VDA. The returned Status (A) is a standard HBIOS result code.

If the hardware and driver supports it, you can specify a Font Bitmap (HL) buffer address containing the character bitmap data to be loaded into the video processor. The buffer **must** be located entirely in the top 32K of the CPU memory space. HL must be set to zero if no character bitmap is specified (the driver will utilize a default character bitmap).

10.7.2 Function 0x41 – Video Query (VDAQRY)

Entry Parameters	Returned Values
B: 0x41	A: Status

Entry Parameters	Returned Values
C: Video Unit HL: Font Bitmap	C: Video Mode D: Rows E: Columns HL: Font Bitmap

Return information about the specified Video Unit (C). Video Mode (C) will be set to the current video mode. Rows (D) and Columns (E) will return the dimensions of the video display as measured in rows and columns. Note that this is the **count** of rows and columns, not the **last** row/column number. The returned Status (A) is a standard HBIOS result code.

If the hardware and driver support it, you can specify a Font Bitmap (HL) buffer address that will be filled with the current character bitmap data. The buffer **must** be located entirely in the top 32K of the CPU memory space. Font Bitmap (HL) **must** be set to zero if it does not point to a proper buffer area or memory corruption will result.

If HL is not zero, it must point to a suitably sized memory buffer in the upper 32K of CPU address space that will be filled with the current character bitmap data. It is critical that HL be set to zero if it does not point to a proper buffer area or memory corruption will result. If the video device driver does not have the ability to provide character bitmap data, then Font Bitmap (HL) will be set to zero on return.

10.7.3 Function 0x42 – Video Reset (VDARES)

Entry Parameters	Returned Values
B: 0x42 C: Video Unit	A: Status

Performs a non-destructive reset of the specified Video Unit (C).
Should re-initialize the video hardware without destroying the screen contents or cursor position. The current video mode will not be changed. The returned Status (A) is a standard HBIOS result code.

10.7.4 Function 0x43 – Video Device (VDADEV)

Entry Parameters	Returned Values
B: 0x43 C: Video Unit	A: Status C: Device Attributes D: Device Type E: Device Number H: Device Unit Mode L: Device I/O Base Address

Reports device information about the specified Video Unit (C). The Status (A) is a standard HBIOS result code.

Device Attribute (C) values are not yet defined.

Device Type (D) indicates the specific hardware driver that handles the specified Video Unit (C). Values are listed at the start of this section. Device Number (E) indicates the physical device number assigned per driver.

Device Mode (H) is used to indicate the variant of the chip or circuit that is used by the specified unit. For example, for an TMS video unit, the value indicates the TMS circuit variant. The Device I/O Base Address (L) indicates the starting port address of the hardware interface that is servicing the specified unit. Both of these values are considered driver specific. Refer to the associated hardware driver for the values used.

10.7.5 Function 0x44 – Video Set Cursor Style (VDASCS)

Entry Parameters	Returned Values
B: 0x44 C: Video Unit D: Start/End E: Style	A: Status

If supported by the specified Video Unit (C), adjust the format of the cursor such that the cursor starts at the pixel specified in the top nibble of Start/End (D) and ends at the pixel specified in the bottom nibble of Start/End (D). So, if D=0x08, a block cursor would be used that starts at the top pixel of the character cell and ends at the ninth pixel of the character cell. The Status (A) is a standard HBIOS result code.

Style (E) is reserved to control the style of the cursor (blink, visibility, etc.), but is not yet implemented.

Adjustments to the cursor style may or may not be possible for any given video hardware and may be dependent on the active video mode.

10.7.6 Function 0x45 – Video Set Cursor Position (VDASCP)

Entry Parameters	Returned Values
B: 0x45 C: Video Unit D: Row E: Column	A: Status

Reposition the cursor of the specified Video Unit (C) to the specified Row (D) and Column (E). Specifying a row/column that exceeds the boundaries of the display results in undefined behavior. Cursor coordinates are 0 based (0,0 is the upper left corner of the display). The Status (A) is a standard HBIOS result code.

10.7.7 Function 0x46 – Video Set Character Attribute (VDASAT)

Entry Parameters	Returned Values
B: 0x46 C: Video Unit E: Attribute	A: Status

Assign the specified character Attribute (E) code to be used for all subsequent character writes/fills on the specified Video Unit (C). This attribute is used to fill new lines generated by scroll operations. The character attributes values are listed above. Note that a given video display may or may not support any/all attributes. The Status (A) is a standard HBIOS result code.

10.7.8 Function 0x47 – Video Set Character Color (VDASCO)

Entry Parameters	Returned Values
B: 0x47 C: Video Unit D: Scope	A: Status

Entry Parameters	Returned Values
E: Color	

Assign the specified Color (E) code for character foreground/background. If Scope (D) is 0, the specified color will be used for all subsequent character writes/fills. This color is also used to fill new lines generated by scroll operations. If Scope (D) is 1, then the specified foreground/background color will be applied immediately to the entire screen. Refer to the color code table above for a list of the available color codes. Note that a given video display may or may not support any/all colors. The Status (A) is a standard HBIOS result code.

10.7.9 Function 0x48 – Video Write Character (VDAWRC)

Entry Parameters	Returned Values
B: 0x48 C: Video Unit E: Character	A: Status

Write the Character (E) value to the display of the specified Video Unit (C). The character is written starting at the current cursor position and the cursor is advanced. If the end of the line is encountered, the cursor will be advanced to the start of the next line. The display will **not** scroll if the end of the screen is exceeded. The Status (A) is a standard HBIOS result code.

10.7.10 Function 0x49 – Video Fill (VDAFIL)

Entry Parameters	Returned Values
B: 0x49 C: Video Unit E: Character HL: Count	A: Status

Write the Character (E) value to the Video Unit (C) display the number of times specified by Count (HL). Characters are written starting at the current cursor position and the cursor is advanced by the number of characters written. If the end of the line is encountered, the characters will continue to be written starting at the next line as needed. The display will **not** scroll if the end of the screen is exceeded. Writing characters beyond the end of the screen results in undefined behavior. The Status (A) is a standard HBIOS result code.

10.7.11 Function 0x4A – Video Copy (VDACPY)

Entry Parameters	Returned Values
B: 0x4A C: Video Unit D: Source Row E: Source Column L: Count	A: Status

Copy Count (L) bytes from the specified Video Unit (C) display Source Row (D) and Source Column (E) to the current cursor position. The cursor position is not updated. The maximum Count (L) value is 255. Copying to/from overlapping areas is not supported and will have an undefined behavior. The display will **not** scroll if the end of the screen is exceeded. Copying beyond the active screen buffer area is not supported and results in undefined behavior. The Status (A) is a standard HBIOS result code.

10.7.12 Function 0x4B – Video Scroll (VDASCR)

Entry Parameters	Returned Values
B: 0x4B C: Video Unit E: Lines	A: Status

Scroll the video display of the specified Video Unit (C) forward or backwards by number of Lines (E) specified. If Lines (E) is positive, then a forward scroll is performed. If Lines (E) contains a negative number, then a reverse scroll will be performed. This function will scroll the entire screen contents. New lines revealed during the scroll operation will be filled with space characters (0x20) using the active character attribute and color. The cursor position will **not** be updated. The Status (A) is a standard HBIOS result code.

10.7.13 Function 0x4C – Video Keyboard Status (VDAKST)

Entry Parameters	Returned Values
B: 0x4C C: Video Unit	A: Status / Codes Pending

Return a count of the number of key Codes Pending (A) in the keyboard buffer for the specified Video Unit (C). If it is not possible to determine the actual number in the buffer, it is acceptable to return 1 to indicate there are key codes available to read and 0 if there are none available.

The value returned in register A is used as both a Status (A) code and the return value. Negative values (bit 7 set) indicate a standard HBIOS result (error) code. Otherwise, the return value represents the number of key codes pending.

10.7.14 Function 0x4D – Video Keyboard Flush (VDAKFL)

Entry Parameters	Returned Values
B: 0x4D C: Video Unit	A: Status

If a keyboard buffer is in use on the Video Unit (C) specified, it should be purged and all contents discarded. The Status (A) is a standard HBIOS result code.

10.7.15 Function 0x4E – Video Keyboard Read (VDAKRD)

Entry Parameters	Returned Values
B: 0x4E C: Video Unit	A: Status C: Scancode D: Keystate E: Keycode

Read the next key data from keyboard of the specified Video Unit (C). If a keyboard buffer is used, return the next Keycode in the buffer. If no key data is available, this function will wait indefinitely for a keypress. The Status (A) is a standard HBIOS result code.

The Scancode (C) value is the raw scancode from the keyboard for the keypress. Scancodes are optional and may not be implemented by the driver. The Scancode values are driver dependent. In the case of a PS/2 keyboard driver, they should be the PS/2 scancode. Other keyboard drivers may return values appropriate for their specific keyboard. If the driver does not implement this, it should return 0 in C.

The Keystate (D) is a bitmap representing the value of all modifier keys and shift states as they existed at the time of the keystroke. The bitmap is defined as:

Bit	Keystate Indication
7	Key pressed was from the num pad
6	Caps Lock was active
5	Num Lock was active
4	Scroll Lock was active
3	Windows key was held down
2	Alt key was held down
1	Control key was held down
0	Shift key was held down

Not all of these bits may be relevant for all keyboards. Any bit that is not relevant should be returned as 0.

The Keycode (E) is generally returned as appropriate ASCII values, if possible. Special keys, like function keys and arrows, are returned as reserved codes as described at the start of this section.

10.7.16 Function 0x4F – Read a character at current video position (VDARDC)

Entry Parameters	Returned Values
B: 0x4F	A: Status
C: Video Unit	E: Character
	B: Color
	E: Attribute

This function will return the character data from the current cursor position of the display of the specified Video Unit (C). The data returned includes the Character (E) value, the Color (B), and the Attribute (E) corresponding to the current cursor position. If the display does not support colors or attributes then this function will return color white on black with no attributes. The ability to perform this function may not be available for all video devices. The Status (A) is a standard HBIOS result code.

10.8 Sound (SND)

Sound functions require that a Sound Unit number be specified in register C. This is the logical device unit number assigned during the boot process that identifies all sound devices uniquely.

All sound units are assigned a Device Type ID which indicates the specific hardware device driver that handles the unit. The table below enumerates these values.

Device Type	ID	Description	Driver
SND-DEV_SN76489	$00	SN76489 Programmable Sound Generator	sn76489.asm
SND-DEV_AY38910	$01	AY-3-8910/YM2149 Programmable Sound Generator	ay38910.asm
SNDDEV_BIT-MODE	$02	Bit-bang Speaker	spk.asm
SND-DEV_YM2612	$03	YM2612 Programmable Sound Generator	ym2612.asm

The Sound functions defer the actual programming of the sound chip until the SNDPLAY function is called. You will call the volume and period/note functions to preset the desired sound output, then call SNDPLAY when you want the sound to change.

The Sound functions do not manage the duration of the sound played. A sound will play indefinitely – the caller must implement an appropriate timing mechanism to manage the playing of a series of sounds.

```
HBIOS B=51 C=00 L=80      ; Set volume to half level
HBIOS B=53 C=00 HL=152    ; Select Middle C (C4)
HBIOS B=54 C=00 D=01      ; Play note on Channel 1
```

10.8.1 Function 0x50 – Sound Reset (SNDRESET)

Entry Parameters	Returned Values
B: 0x50 C: Sound Unit	A: Status

Reset the sound chip of specified Sound Unit (C). Turn off all sounds and set volume on all channels to silence. The returned Status (A) is a standard HBIOS result code.

10.8.2 Function 0x51 – Sound Volume (SNDVOL)

Entry Parameters	Returned Values
B: 0x51 C: Sound Unit L: Volume	A: Status

This function sets the sound chip Volume (L) for the specified Sound Unit (C). Volume (L) is a binary value ranging from 0 (silence) to 255 (maximum). The volume will be applied when the next SNDPLAY function is invoked. The returned Status (A) is a standard HBIOS result code.

Note that not all sounds chips implement 256 volume levels. The driver will scale the volume to the closest possible level the chip provides.

10.8.3 Function 0x52 – Sound Period (SNDPRD)

Entry Parameters	Returned Values
B: 0x52 C: Sound Unit HL: Period	A: Status

This function sets the sound chip Period (HL) for the specified Sound Unit (C). The period will be applied when the next SNDPLAY function is invoked. The returned Status (A) is a standard HBIOS result code.

The Period (HL) value is **not** a standardized value. The value is programmed directly into the period or frequency register of the sound chip. It is therefore a hardware dependent value. To play standardized notes, use the SNDNOTE function.

10.8.4 Function 0x53 – Sound Note (SNDNOTE)

Entry Parameters	Returned Values
B: 0x53 C: Sound Unit HL: Note	A: Status

This function sets the frequency generated by the sound of the specified Sound Unit (C). The frequency is standardized and is specified by using values that correspond to musical notes. The frequency will be applied when the next SNDPLAY function is invoked. The returned Status (A) is a standard HBIOS result code.

The Note (HL) values correspond to eighth tones. Increasing/decreasing the value by 8 results in a full tone increment/decrement.
Increasing/decreasing the value by 48 results in a full octave increment/decrement. The value 0 corresponds to Bb/A# in octave 0.

The sound chip resolution and its oscillator limit the range and accuracy of the notes played. The typical range of the AY-3-8910 is six octaves: Bb2/A#2 to A7, where each value is a unique tone. Values above and below can still be played but each eighth tone step may not result in a tone change.

The following table shows the mapping of the Note (HL) value to the corresponding octave and note.

Note	Octave							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
C	-	8	56	104	152	200	248	296
C#/Db	-	12	60	108	156	204	252	300
D	-	16	64	112	160	208	256	304
D#/Eb	-	20	68	116	164	212	260	308
E	-	24	72	120	168	216	264	312
F	-	28	76	124	172	220	268	316
F#/Gb	-	32	80	128	176	224	272	320
G	-	36	84	132	180	228	276	324
G#/Ab	-	40	88	136	184	232	280	328
A	-	44	92	140	188	236	284	332
A#/Bb	0	48	96	144	192	240	288	336
B	4	52	100	148	196	244	292	340

10.8.5 Function 0x54 – Sound Play (SNDPLAY)

Entry Parameters	Returned Values
B: 0x54 C: Sound Unit D: Channel	A: Status

This function applies the previously specified volume and frequency of the specified Sound Unit (C) by programming the sound chip with the appropriate values. The values are applied to the specified Channel (D) of the chip. The returned Status (A) is a standard HBIOS result code.

Note that there is no duration for the sound output – the programmed sound will be played indefinitely. It is up to the user to wait the desired amount of time, then change or silence the sound output as desired.

The number of channels available on a sound chip varies. It is up to the caller to ensure that the appropriate number of channels are being programmed.

10.8.6 Function 0x55 – Sound Query (SNDQUERY)

Entry Parameters	Returned Values
B: 0x55 C: Sound Unit E: Subfunction	A: Status

This function will return a variety of information for a specified Sound Unit (C) according to the Subfunction (E) specified. The returned Status (A) is a standard HBIOS result code.

SNDQUERY Subfunction 0x01 – Get count of audio channels supported (SNDQ_CHCNT)

Entry Parameters	Returned Values
B: 0x55 C: Sound Unit E: 0x01	A: Status B: Tone Channels C: Noise Channels

SNDQUERY Subfunction 0x02 – Get current volume setting (SNDQ_VOL)

Entry Parameters	Returned Values
B: 0x55 C: Sound Unit E: 0x02	A: Status L: Volume

SNDQdERY Subfunction 0x03 – Get current period setting (SNDQ_PERIOD)

Entry Parameters	Returned Values
B: 0x55 C: Sound Unit E: 0x03	A: Status HL: Period

SNDQUERY Subfunction 0x04 – Get device details (SNDQ_DEV)

Entry Parameters	Returned Values
B: 0x55 C: Sound Unit E: 0x04	A: Status B: Driver Identity HL: Ports DE: Ports

This subfunction reports detailed device information for the specified Sound Unit (C).

Driver Identity (B) reports the audio device type. Ports (HL & DE) return relevant port addresses for the hardware specific to each device type.

The following table defines the specific port information per device type:

Audio ID	*Value*	*Device*	*Returned Registers*
SND_SN76489	0x01	SN76489	E=Left channel port, L=Right channel port
SND_AY38910	0x02	AY-3-8910	D=Address port, E=Data port
SND_BITMODE	0x03	I/O PORT	D=Address port, E=Bit mask
SND_YM2612	0x04	YM2612	Part 0: D=Address port, E=Data port Part 1: D=Address port, L=Part 1 Data port

10.8.7 Function 0x56 – Sound Duration (SNDDUR)

Entry Parameters	Returned Values
B: 0x56 C: Sound Unit HL: Duration	A: Status

This function sets the Duration (HL) of the note to be played in milliseconds for the specified Sound Unit (C). This function just sets the duration, the actual duration is applied in the SNDPLAY function.

If the Duration (HL) is set to zero, then the SNDPLAY function will operate in a non-blocking mode. i.e. a tone will start playing and the play function will return. The tone will continue to play until the next tone is played. If the Duration (HL) is greater than zero, the sound will play for the duration defined in HL and then return.

***** Function Not Implemented ****

10.8.8 Function 0x57 – Sound Device (SNDDEVICE)

Entry Parameters	Returned Values
B: 0x57	A: Status
C: Sound Unit	C: Device Attributes
	D: Device Type
	E: Device Number
	H: Device Unit Mode
	L: Device I/O Base Address

Reports device information about the specified Sound Unit (C). The Status (A) is a standard HBIOS result code.

The Device Attributes (C) value is not yet defined.

Device Type (D) indicates the specific hardware driver that handles the specified Sound Unit (C). Values are listed at the start of this section. Device Number (E) indicates the physical device number assigned per driver.

Device Mode (H) is used to indicate the variant of the chip or circuit that is used by the specified unit. The Device I/O Base Address (L) indicates the starting port address of the hardware interface that is servicing the specified unit. Both of these values are considered driver specific. Refer to the associated hardware driver for the values used.

10.8.9 Function 0x58 – Sound Beep (SNDBEEP)

Entry Parameters	Returned Values
B: 0x58	A: Status

Entry Parameters	Returned Values
C: Sound Unit	

Play a beep tone on the specified Sound Unit (C). The beep will normally be about 1/3 second in duration and the tone will be approximately B5.

10.9 Extension (EXT)

Helper (extension) functions that are not a core part of a BIOS.

10.9.1 Function 0xE0 – Calculate Slice (EXTSLICE)

Entry Parameters	Returned Values
B: 0xE0	A: Status
D: Disk Unit	B: Device Attributes
E: Slice	C: Media ID
	DEHL: Sector Address

Report the Media ID (C), and Device Attributes (B) for the for media in the specified Disk Unit (D), and for hard disks the absolute Sector offset to the start of the Slice (E). The Status (A) is a standard HBIOS result code.

This function extends upon Function 0x18 – Disk Media (DIOMEDIA) for hard disk media by scanning for a partition to determine if the disk uses HD512 or HD1K, correctly reporting MID_HD or MID_HDNEW respectively. See the following for some background Mapping to Media ID

It will also return the sector number of the first sector in the slice if the slice number is valid. If the slice number is invalid (it wont fix on the media) an error will be returned.

The slice calculation is performed by considering the partition start (if it exists), the size of a slice for the given format type, and ensuring that the slice fits within the media or partition size, taking into consideration other partitions that may exist.

The Device Attributes (B) are the same as defined in Function 0x17 – Disk Device (DIODEVICE)

If the Unit specified is not a hard disk the Media ID will be returned and the slice parameter ignored. If there is no media in device, or the slice number is invaid (Parameter Out Of Range) the function will return an error status.

NOTE: This function was placed in HBIOS to be shared between the different CP/M variants supported by RomWBW. It is not strictly a BIOS function, and may be moved in future.

10.10 System (SYS)

10.10.1 Function 0xF0 – System Reset (SYSRESET)

Entry Parameters	Returned Values
B: 0xF0 C: Subfunction	A: Status

This function performs various forms of a system reset depending on the value of Subfunction (C):

Soft Reset (0x00): Perform a soft reset of HBIOS. Releases all HBIOS memory allocated by current OS. Does not reinitialize physical devices.

Warm Start (0x01): Warm start the system returning to the boot loader prompt. Does not reinitialize physical devices.

Cold Start (0x02): Perform a system cold start (like a power on). All devices are reinitialized.

User Restart (0x03): Perform a video terminal reset. Terminal emulation and visual display systems are reset.

The Status (A) is a standard HBIOS result code.

10.10.2 Function 0xF1 – System Version (SYSVER)

Entry Parameters	Returned Values
B: 0xF1 C: Reserved	A: Status DE: Version L: Platform

This function will return the HBIOS Version (DE) number and Platform (L) identifier. The Status (A) is a standard HBIOS result code.

The Version (DE)number is encoded as BCD where the 4 digits are:

[Major Version][Minor Version][Patch Level][Build Number]

So, for example, a Version (DE) number of 0x3102 would indicate version 3.1.0, build 2.

The hardware Platform (L) is identified as follows:

Name	Id	Platform
PLT_SBC	1	ECB Z80 SBC
PLT_ZETA	2	ZETA Z80 SBC
PLT_ZETA2	3	ZETA Z80 V2 SBC
PLT_N8	4	N8 (HOME COMPUTER) Z180 SBC
PLT_MK4	5	MARK IV
PLT_UNA	6	UNA BIOS
PLT_RCZ80	7	RCBUS W/ Z80
PLT_RCZ180	8	RCBUS W/ Z180
PLT_EZZ80	9	EASY/TINY Z80
PLT_SCZ180	10	SMALL COMPUTER CENTRAL Z180
PLT_DYNO	11	DYNO MICRO-ATX MOTHERBOARD
PLT_RCZ280	12	RCBUS W/ Z280
PLT_MBC	13	NHYODYNE MULTI-BOARD COMPUTER
PLT_RPH	14	RHYOPHYRE GRAPHICS SBC
PLT_Z80RETRO	15	Z80 RETRO COMPUTER
PLT_S100	16	S100 COMPUTERS Z180
PLT_DUO	17	DUODYNE Z80 SYSTEM
PLT_HEATH	18	HEATHKIT H8 Z80 SYSTEM
PLT_EPITX	19	Z180 MINI-ITX
PLT_MON	20	MONSPUTER (DEPRECATED)
PLT_GMZ180	21	GENESIS Z180 SYSTEM
PLT_NABU	22	NABU PC W/ ROMWBW OPTION BOARD
PLT_FZ80	23	S100 FPGA Z80
PLT_RCEZ80	24	RCBUS W/ eZ80

For more information on these platforms see RomWBW Hardware

10.10.3 Function 0xF2 – System Set Bank (SYSSETBNK)

Entry Parameters	Returned Values
B: 0xF2	A: Status
C: Bank ID	C: Prior Bank ID

Activates the specified memory Bank ID (C) and returns the Prior Bank ID (C).

The function **must** be invoked from code located in the upper 32K and the stack **must** be in the upper 32K. The Status (A) is a standard HBIOS result code.

If the system is using interrupt mode 1 interrupts, the you **must** take steps to ensure interrupts are properly handled. You generally have two choices:

- Disable interrupts while the User Bank is switched out
- Duplicate the interrupt mode 1 vector from the User Bank into the bank you are switching to.

If the User Bank has been switched out, you will not be able to invoke the HBIOS API functions using an `RST 08` instruction. You can use the alternative mechanism using `CALL $FFF0` as described in Invocation.

10.10.4 Function 0xF3 – System Get Bank (SYSGETBNK)

Entry Parameters	Returned Values
B: 0xF3	A: Status C: Bank ID

Returns the currently active Bank ID (C). The Status (A) is a standard HBIOS result code.

10.10.5 Function 0xF4 – System Set Copy (SYSSETCPY)

Entry Parameters	Returned Values
B: 0xF4 D: Destination Bank ID E: Source Bank ID HL: Byte Count	A: Status

Prepare for a subsequent interbank memory copy (SYSBNKCPY) function call by setting the Source Bank ID (E), Destination Bank ID (D), and Byte Count (HL) to be copied. The bank ID's are not range checked and must be valid for the system in use. The Status (A) is a standard HBIOS result code.

No bytes are copied by this function. The SYSBNKCPY function must be called to actually perform the copy. The values setup by this function will remain unchanged until another call is make to this function. So, after calling SYSSETCPY, you may make multiple calls to SYSBNKCPY as long as you want to continue to copy between the already established Source/Destination Banks and the same size copy is being performed.

10.10.6 Function 0xF5 – System Bank Copy (SYSBNKCPY)

Entry Parameters	Returned Values
B: 0xF5	A: Status
DE: Destination Address	DE: New Destination Address
HL: Source Address	HL: New Source Address

Copy a block of memory between banks. The Source Bank, Destination Bank, and Byte Count to copy **must** be established with a prior call to SYSSETCPY. However, it is not necessary to call SYSSETCPY prior to subsequent calls to SYSBNKCPY if the source/destination banks and copy length do not change.

On return, the New Destination Address (DE) will be value of the original Destination Address (DE) incremented by the count of bytes copied. Likewise for the New Source Address (HL). This allows iterative invocations of this function to continue copying where the prior invocation left off.

The Status (A) is a standard HBIOS result code.

WARNINGS:

- This function is inherently dangerous and does not prevent you from corrupting critical areas of memory. Use with **extreme** caution.
- Overlapping source and destination memory ranges are not supported and will result in undetermined behavior.
- Copying of byte ranges that cross bank boundaries is undefined.

10.10.7 Function 0xF6 – System Alloc (SYSALLOC)

Entry Parameters	Returned Values
B: 0xF6	A: Status
HL: Block Size	HL: Block Address

This function will attempt to allocate a Block Size (HL) bytes block of memory from the internal HBIOS heap. The HBIOS heap resides in the HBIOS bank in the area of memory left unused by HBIOS. If the allocation is successful, the Block Address (HL) of the allocated memory block is returned in HL. You will typically need to use the SYSBNKCPY function to read/write the allocated memory. The Status (A) is a standard HBIOS result code.

10.10.8 Function 0xF7 – System Free (SYSFREE)

Entry Parameters	Returned Values
B: 0xF7 HL: Block Address	A: Status

***** Function Not Implemented *****

Note that all allocated memory can be freed by calling the SYSRESET function with a subfunction code of 0x00 (Soft Reset).

10.10.9 Function 0xF8 – System Get (SYSGET)

Entry Parameters	Returned Values
B: 0xF8 C: Subfunction	A: Status

This function will report various system information based on the sub-function value. The following lists the subfunctions available along with the registers/information utilized. The Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0x00 – Get Character Device Unit Count (CIOCNT)

Entry Parameters	Returned Values
B: 0xF8 C: 0x00	A: Status E: Count

Return the Count (E) of character device units. The Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0x01 – Get Serial Unit Function (CIOFN)

Entry Parameters	Returned Values
B: 0xF8 C: 0x01 D: Function	A: Status HL: Function Address DE: Unit Data Address

Entry Parameters	Returned Values
E: Unit	

This function will lookup the actual driver function address and unit data address inside the HBIOS driver. On entry, place the CIO function number to lookup in D and the CIO unit number in E. On return, HL will contain the address of the requested function in the HBIOS driver (in the HBIOS bank). DE will contain the associated unit data address (also in the HBIOS bank). See Appendix A for details. The returned Status (A) is a standard HBIOS result code.

This function can be used to speed up HBIOS calls by looking up the function and data address for a specific driver function. After this, the caller can use interbank calls directly to the function in the driver which bypasses the overhead of the normal function invocation lookup.

SYSGET Subfunction 0x10 – Get Disk Device Unit Count (DIOCNT)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x10	E: Count

Return the Count (E) of disk device units. The Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0x11 – Get Disk Unit Function (DIOFN)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x11	HL: Function Address
D: Function	DE: Unit Data Address
E: Unit	

This function will lookup the actual driver function address and unit data address inside the HBIOS driver. On entry, place the DIO function number to lookup in D and the DIO unit number in E. On return, HL will contain the address of the requested function in the HBIOS driver (in the HBIOS bank). DE will contain the associated unit data address (also in the HBIOS bank). See Appendix A for details. The returned Status (A) is a standard HBIOS result code.

This function can be used to speed up HBIOS calls by looking up the function and data address for a specific driver function. After this, the caller can use interbank calls directly to the function in the driver which bypasses the overhead of the normal function invocation lookup.

SYSGET Subfunction 0x20 – Get RTC Device Unit Count (RTCCNT)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x20	E: Count

Return the Count (E) of RTC device units. The Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0x40 – Get Video Device Unit Count (VDACNT)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x40	E: Count

Return the Count (E) of video device units. The Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0x41 – Get Video Unit Function (VDAFN)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x41	HL: Function Address
D: Function	DE: Unit Data Address
E: Unit	

This function will lookup the actual driver function address and unit data address inside the HBIOS driver. On entry, place the VDA function number to lookup in D and the VDA unit number in E. On return, HL will contain the address of the requested function in the HBIOS driver (in the HBIOS bank). DE will contain the associated unit data address (also in the HBIOS bank). See Appendix A for details. The returned Status (A) is a standard HBIOS result code.

This function can be used to speed up HBIOS calls by looking up the function and data address for a specific driver function. After this, the caller can use interbank calls directly to the function in the driver which bypasses the overhead of the normal function invocation lookup.

SYSGET Subfunction 0x50 – Get Sound Device Unit Count (SNDCNT)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x50	E: Count

Return the Count (E) of sound device units. The Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0x51 – Get Sound Unit Function (SNDFN)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0x51	HL: Function Address
D: Function	DE: Unit Data Address
E: Unit	

This function will lookup the actual driver function address and unit data address inside the HBIOS driver. On entry, place the SND function number to lookup in D and the SND unit number in E. On return, HL will contain the address of the requested function in the HBIOS driver (in the HBIOS bank). DE will contain the associated unit data address (also in the HBIOS bank). See Appendix A for details. The returned Status (A) is a standard HBIOS result code.

This function can be used to speed up HBIOS calls by looking up the function and data address for a specific driver function. After this, the caller can use interbank calls directly to the function in the driver which bypasses the overhead of the normal function invocation lookup.

SYSGET Subfunction 0xC0 – Get Switches (SWITCH)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xC0	HL: Switch Value
D: Switch Key	

This function will return the current value (HL) of the switch (D) from NVRAM.

Switches may be returned as a 16 bit (HL) or 8 bit (L) value. It is up to the caller to process the returned value correctly. Note for Switch 0xFF (status) the returned value is primarily in the Status (A) register.

Errors are signaled in the return by setting the NZ flag. When set the (A) register may contain an error code, but this code does not conform to RomWBW standard

Success is indicated by setting the Z flag

For a description of switches please see RomWBW NVRAM Configuration

SYSGET Subfunction 0xD0 – Get Timer Tick Count (TIMER)

Entry Parameters	Returned Values
B: 0xF8 C: 0xD0	A: Status DEHL: Tick Count C: Frequency

Return the value of the global system timer Tick Count (DEHL). This is a double-word binary value. The frequency of the system timer in Hertz is returned in Frequency (C). The returned Status (A) is a standard HBIOS result code.

The tick count is a 32 bit binary value. It will rollover to zero if the maximum value for a 32 bit number is reached.

Note that not all hardware configuration have a system timer. You can determine if a timer exists by calling this function repeatedly to see if it is incrementing.

SYSGET Subfunction 0xD1 – Get Seconds Count (SECONDS)

Entry Parameters	Returned Values
B: 0xF8 C: 0xD1	A: Status DEHL: Seconds Count C: Remainder Ticks

Return the Seconds Count (DEHL) with the number of seconds that have elapsed since the system was started. This is a double-word binary value. Additionally, Remainder Ticks (C) is returned and contains the number of ticks that have elapsed within the current second.

Note that Remainder Ticks (C) will have a value from 0 to 49 since there are 50 ticks per second. So, Remainder Ticks does not represent a fraction of the current second. Remainder Ticks (C) can be doubled to derive the hundredths of milliseconds elapsed within the current second.

The availability of the Seconds Count (DEHL) is dependent on having a system timer active. If the hardware configuration has no system timer, then Seconds Count (DEHL) will not increment.

SYSGET Subfunction 0xE0 – Get Boot Information (BOOTINFO)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xE0	L: Boot Bank ID
	D: Boot Disk Unit
	E: Boot Disk Slice

This function returns information about the most recent boot operation performed. It includes the Boot Bank ID (L), the Boot Disk Unit (D), and the Boot Disk Slice (E). The returned Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0xF0 – Get CPU Information (CPUINFO)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xF0	H: Z80 CPU Variant
	L: CPU Speed MHz
	DE: CPU Speed KHz
	BC: Oscillator Speed KHz

This function returns information about the active CPU environment. The Z80 CPU Variant (H) will be one of: 0=Z80, 1=Z180, 2=Z180-K, 3=Z180-N, 4=Z280. The current CPU speed is provided as both CPU Speed MHz (L) and CPU Speed KHz (DE). The raw oscillator speed is provided as Oscillator Speed KHz (BC). The returned Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0xF1 – Get Memory Information (MEMINFO)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xF1	D: ROM Bank Count
	E: RAM Bank Count

This function returns the systems ROM Bank Count (D) and RAM Bank Count (E). Each bank is 32KB by definition. The returned Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0xF2 – Get Bank Information (BNKINFO)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xF2	D: BIOS Bank ID
	E: User Bank ID

Certain memory banks within a RomWBW system are special. The exact bank id for each of these varies depending on the configuration of the system. This function can be used to determine the BIOS Bank ID (D) and the User Bank ID (E). The returned Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0xF3 – Get CPU Speed (CPUSPD)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xF3	L: Clock Mult
	D: Memory Wait States
	E: I/O Wait States

This function will return the running CPU speed attributes of a system. The Clock Mult (L) returned indicates the frequency multiple being applied to the raw oscillator clock. If is defined as: 0=Half, 1=Full, and 2=Double. The wait states for the system are also provided as Memory Wait States (D) and I/O Wait States (E). The value of Memory Wait States (D) is the actual number of wait states, not the number of wait states added. The returned Status (A) is a standard HBIOS result code.

SYSGET Subfunction 0xF4 – Get Front Panel Swithes (PANEL)

Entry Parameters	Returned Values
B: 0xF8	A: Status
C: 0xF4	L: Switches

This function will return the current value of the switches (L) from the front panel of the system. If no front panel is available in the system, the returned Status (A) will indicate a No Hardware error.

SYSGET Subfunction 0xF5 – Get Application Banks Information (APPBNKS)

Entry Parameters	Returned Values
B: 0xF8 C: 0xF5	A: Status H: App Banks Start ID L: App Banks Count E: Bank Size

HBIOS may be configured to reserve a number of RAM memory banks that will be available for application use. This function returns information about the RAM memory banks currently available for application use. The function provides the bank id of the first available application bank (H) and the count of banks available (L). It also returns the size of a bank expressed as a number of 256-byte pages (E). The returned Status (A) is a standard HBIOS result code.

The application banks are always a contiguous set of banks, so the App Banks Start ID can be incremented to address additional banks up to the limit indicated by App Banks Count. If the App Banks Count is zero, then there are no application banks available (regardless of the value of App Banks Start ID).

HBIOS does not provide any mechanism to reserve application banks. Any concept of allocation of application banks must be implemented within the OS or application.

This function does not change the current bank selected. You must use Function 0xF2 – System Set Bank (SYSSETBNK) or the proxy function Bank Select (BNKSEL) for this. Be sure to observe the warnings in the description of this function.

10.10.10 Function 0xF9 – System Set (SYSSET)

Entry Parameters	Returned Values
B: 0xF9 C: Subfunction	A: Status

This function will set various system parameters based on the sub-function value. The following lists the subfunctions available along with the registers/information utilized. The Status (A) is a standard HBIOS result code.

SYSSET Subfunction 0xC0 – Set Switches (SWITCH)

Entry Parameters	Returned Values
B: 0xF9 C: 0xC0 D: Switch Key HL: Switch Value	A: Status

This function will set the value (HL) into the switch (D) and store it into NVRAM.

Switches may be passed as a 16 bit (HL) or 8 bit (L) value. It is up to the caller to send the value correctly. Note for Switch 0xFF (reset) the value (HL) is ignored

Errors are signalled in the return by setting the NZ flag. When set the (A) register may contain an error code, but this code does not conform to RomWBW standard

Success is indicated by setting the Z flag

For a description of switches please see RomWBW NVRAM Configuration

SYSSET Subfunction 0xD0 – Set Timer Tick Count (TIMER)

Entry Parameters	Returned Values
B: 0xF9 C: 0xD0	A: Status DEHL: Timer Tick Count

This function will explicitly set the system Timer Tick Count (DEHL) value. DEHL is a double-word binary value. The Status (A) is a standard HBIOS result code.

SYSSET Subfunction 0xD1 – Set Seconds Count (SECONDS)

Entry Parameters	Returned Values
B: 0xF9 C: 0xD1 DEHL: Seconds Count	A: Status

This function will explicitly set the system Seconds Count (DEHL) value. DEHL is a double-word binary value. The Status (A) is a standard HBIOS result code.

SYSSET Subfunction 0xE0 – Set Boot Information (BOOTINFO)

Entry Parameters	Returned Values
B: 0xF9 C: 0xE0 L: Boot Bank ID D: Boot Disk Unit E: Boot Disk Slice	A: Status

This function sets information about the most recent boot operation performed. It includes the Boot Bank ID (L), the Boot Disk Unit (D), and the Boot Disk Slice (E). The returned Status (A) is a standard HBIOS result code.

SYSSET Subfunction 0xF3 – Set CPU Speed (CPUSPD)

Entry Parameters	Returned Values
B: 0xF9 C: 0xF3 L: Clock Mult D: Memory Wait States E: I/O Wait States	A: Status

This function will modify the running CPU speed attributes of a system. Note that it is frequently impossible to tell if a system is capable of dynamic speed changes. This function makes the changes blindly. You can specify 0xFF for either of the wait state settings to have them left alone. If an attempt is made to change the speed of a system that is definitely incapable of doing so, then an error result is returned. The returned Status (A) is a standard HBIOS result code.

The function will attempt to set the CPU speed based on the Clock Mult (L) value: 0=Half, 1=Full, 2=Double. Memory Wait States (D) and I/O Wait States (E) will be set if possible. The value of Memory Wait States (D) is the actual number of wait states, not the number of wait states added.

Some peripherals are dependent on the CPU speed. For example, the Z180 ASCI baud rate and system timer are derived from the CPU speed. The Set CPU Speed function will attempt to adjust these peripherals for correct operation after modifying the CPU speed. However, in some cases this may not be possible. The baud rate of ASCI ports have a limited set of

divisors. If there is no satisfactory divisor to retain the existing baud rate under the new CPU speed, then the baud rate of the ASCI port(s) will be affected.

SYSSET Subfunction 0xF4 – Set Front Panel LEDs (PANEL)

Entry Parameters	Returned Values
B: 0xF9 C: 0xF4 L: LEDs	A: Status

This function will set the front panel LEDs based on the bits in L. If no front panel is available in the system, the returned Status (A) will indicate a No Hardware error.

10.10.11 Function 0xFA – System Peek (SYSPEEK)

Entry Parameters	Returned Values
B: 0xFA D: Bank ID HL: Memory Address	A: Status E: Byte Value

This function retrieves and returns the Byte Value from the specified Bank ID (D) and Memory Address (HL). The bank specified is not range checked. The Status (A) is a standard HBIOS result code.

10.10.12 Function 0xFB – System Poke (SYSPOKE)

Entry Parameters	Returned Values
B: 0xFB D: Bank ID HL: Memory Address E: Byte Value	A: Status

This function sets the Byte Value (E) in the specified Bank ID (D) and Memory Address (HL). The bank specified is not range checked. The Status (A) is a standard HBIOS result code.

10.10.13 Function 0xFC – System Interrupt Management (SYSINT)

Entry Parameters	Returned Values
B: 0xFC C: Subfunction	A: Status

This function allows the caller to query information about the interrupt configuration of the running system and allows adding or hooking interrupt handlers dynamically. Register C is used to specify a sub-function. Additional input and output registers may be used as defined by the sub-function. The Status (A) is a standard HBIOS result code.

Note that during interrupt processing, the lower 32K of CPU address space will contain the RomWBW HBIOS code bank, not the lower 32K of application TPA. As such, a dynamically installed interrupt handler does not have access to the lower 32K of TPA and must be careful to avoid modifying the contents of the lower 32K of memory. Invoking RomWBW HBIOS functions within an interrupt handler is not supported.

Interrupt handlers are different under IM1 and IM2.

Interrupt Mode 1: The new interrupt handler is responsible for chaining (JP) to the previous vector if the interrupt is not handled. If the interrupt is handled, the new handler may simply return (RET). When chaining to the previous interrupt handler, ZF must be set if interrupt is handled and ZF cleared if not handled. The interrupt management framework takes care of saving and restoring AF, BC, DE, HL, and IY. Any other registers modified must be saved and restored by the interrupt handler.

Interrupt Mode 2: The new interrupt handler may either replace or hook the previous interrupt handler. To replace the previous interrupt handler, the new handler just returns (RET) when done. To hook the previous handler, the new handler can chain (JP) to the previous vector. Note that initially all IM2 interrupt vectors are set to be handled as "BAD" meaning that the interrupt is unexpected. In most cases, you do not want to chain to the previous vector because it will cause the interrupt to display a "BAD INT" system panic message.

The interrupt framework will take care of issuing an EI and RETI instruction. Do not put these instructions in your new handler. Additionally, interrupt management framework takes care of saving and restoring AF, BC, DE, HL, and IY. Any other registers modified must be saved and restored by the interrupt handler.

If the caller is transient, then the caller must remove the new interrupt handler and restore the original one prior to termination. This is accomplished by calling this function with the Interrupt Vector set to the Previous Vector returned in the original call.

The caller is responsible for disabling interrupts prior to making an INTSET call and enabling them afterwards. The caller is responsible for ensuring that a valid interrupt handler is installed prior to enabling any hardware interrupts associated with the handler. Also, if the handler is transient, the caller must disable the hardware interrupt(s) associated with the handler prior to uninstalling it.

SYSINT Subfunction 0x00 – Interrupt Info (INTINF)

Entry Parameters	Returned Values
B: 0xFC	A: Status
C: 0x00	D: Interrupt Mode
	E: IVT Size

Return current Interrupt Mode (D) of the system. Also return the number of Interrupt Vector Table (IVT) entries in IVT (E). For IM1, the size of the table is the number of vectors chained together. For IM2, the size of the table is the number of slots in the vector table. The Status (A) is a standard HBIOS result code.

SYSINT Subfunction 0x10 – Get Interrupt (INTGET)

Entry Parameters	Returned Values
B: 0xFC	A: Status
C: 0x10	HL: IVT Address
E: IVT Index	

This function will return the IVT Address (HL) of the current interrupt vector for the specified IVT Index (C). The Status (A) is a standard HBIOS result code.

SYSINT Subfunction 0x20 – Set Interrupt (INTSET)

Entry Parameters	Returned Values
B: 0xFC	A: Status
C: 0x20	HL: Previous Interrupt Address
E: IVT Index	
HL: Interrupt Address	

This function will set a new Interrupt Address (HL) at the IVT Index (E) specified. On return, the Previous Interrupt Address (HL) will be provided.

10.11 Proxy Functions

The following special functions are implemented inside of the HBIOS proxy area at the top of RAM. They do not cause a bank switch and are, therefore, much faster than their corresponding HBIOS API functions.

The functions are invoked via the following dedicated jump table:

Function	Address	** Equate **
Invoke HBIOS Function (INVOKE)	0xFFF0	HB_INVOKE
Bank Select (BNKSEL)	0xFFF3	HB_BNKSEL
Bank Copy (BNKCPY)	0xFFF6	HB_BNKCPY
Bank Call (BNKCALL)	0xFFF9	HB_BNKCALL

The function addresses are also defined as equates in hbios.inc. It is suggested that you use the equates when possible.

To use the functions, you may either call or jump to them. Some examples:

```
CALL    $FFF0
JP      $FFF3
CALL    HB_BNKCPY
```

These functions are inherently dangerous and generally not value checked. Use with extreme caution.

10.11.1 Invoke HBIOS Function (INVOKE)

Address 0xFFF0

This function is an alternate mechanism for invoking the normal HBIOS API functions. The parameters and return values are as documented above. To put it another way, `CALL $FFF0` is equivalent to `RST 08`, but it can be used in any scenario when the normal bank is not selected.

10.11.2 Bank Select (BNKSEL)

Address 0xFFF3

Entry Parameters	Returned Values
A: Bank ID	

This function will select the memory bank identified by Bank ID (A). Register AF is destroyed. All other registers are preserved.

The warnings described in Function 0xF2 – System Set Bank (SYSSETBNK) should be observed.

10.11.3 Bank Copy (BNKCPY)

Address 0xFFF6

Entry Parameters	Returned Values
HL: Source Address	HL: Ending Source Address
DE: Destination Address	DE: Ending Destination Address
BC: Count	BC: 0
HB_SRCBNK: Source Bank ID	
HB_DSTBNK: Destination Bank ID	

This function will copy Count (BC) bytes from Source Address (HL) in Source Bank ID (HB_SRCBNK) to Destination Address (DE) in Destination Bank ID (HB_DSTBNK). The HB_SRCBNK and HB_DSTBNK fields are dedicated locations in the proxy. These locations are defined in hbios.inc:

- Source Bank ID: `HB_SRCBNK` = $FFE4
- Destination Bank ID: `HB_DSTBNK` = $FFE7

The Source Bank ID and Destination Bank ID values must be populated in the specified addresses before calling this function.

During processing, HL and DE, will be incremented. At termination, HL and DE will contain the “next” source/destination addresses that would be copied. This allows this function to be invoked repeatedly to copy continuous blocks of data.

Register AF is destroyed by this function. Register BC will be 0.

10.11.4 Bank Call (BNKCALL)

Address 0xFFF9

Entry Parameters	Returned Values
A: Target Bank ID	
IX: Target Address	

This function will perform a function call to a routine in another bank. It does this by selecting the Target Bank ID (A) and then calling the Target Address (IX). On return from the target function, the originally active bank is selected.

Register usage is determined by the routine that is called.

Since a different bank will be selected while the target function is active, the warnings described in Function 0xF2 – System Set Bank (SYSSETBNK) should be observed.

Chapter 11

Errors and diagnostics

ROMWBW tries to provide useful information when a run time or build time error occurs. Many sections of the code also have code blocks that can be enable to aid in debugging and in some cases the level of reporting detail can be customized.

11.1 Run Time Errors

11.1.1 PANIC

A panic error indicates a non-recoverable error. The processor status is displayed on the console and interrupts are disabled and execution is halted. A cold boot or reset is required to restart.

Example error message:

```
>>> PANIC: @06C4[DFA3:DFC3:0100:F103:04FC:0000:2B5E]

*** System Halted ***
```

The format of the information provided is

```
@XXXX [-AF-:-BC-:-DE-:-HL-:-SP-:-IX-:-IY-]
```

Where `@XXXX` is the address the panic was called from. The other information is the CPU register contents.

Possible reasons a PANIC may occur are:

- RAM Bank range error when attempting a read or write to a RAM disk.

- Sector read function has not been setup but a read was attempted.
- An interrupt vector has not been set up when an interrupt was received.
- There was an attempt to add more devices than the device table had room for.
- An illegal SD card command was encountered.

The @XXXX memory address can be cross referenced with the build source code to identify which section of the software or hardware caused the fault.

11.1.2 SYSCHK

A syschk error is identified when an internal error is detected. When this occurs an error code is returned to the calling program in the A register. A non-zero result indicates an error.

Syschk errors may be reported to the console. Whether this occurs depends on the value of the diagnosis level equate DIAGLVL. By default syschk errors are not reported to the console.

If the diagnosis level is set to display the diagnosis information, then memory address, register dump and error code is displayed. A key difference with the PANIC error is that execution may be continued.

Example error message:

»> SYSCHK: @06C4 [DFA3:DFC3:0100:F103:04FC:0000:2B5E] FD Continue (Y/N)

The format of the information provided is similar the PANIC report.

@XXXX [-AF-:-BC-:-DE-:-HL-:-SP-:-IX-:-IY-] YY

The syschk error codes YY is returned in the A register.

Error	Code YY
Success	0x00
Undefined Error	0xFF
Function Not Implemented	0xFE
Invalid Function	0xFD
Invalid Unit Number	0xFC
Out Of Memory	0xFB
Parameter Out Of Range	0xFA
Media Not Present	0xF9
Hardware Not Present	0xF8
I/O Error	0xF7
Write Request To Read-Only Media	0xF6
Device Timeout	0xF5

Error	Code YY
Invalid Configuration	0xF4
Internal Error	0xF3

11.1.3 Error Level reporting

placeholder

11.2 Build time errors

11.2.1 Build chain tool errors

place holder

11.2.2 Assembly time check errors

placeholder

11.3 Diagnostics

11.3.1 Diagnostic LEDs

Progress through the boot and initialization process can be difficult to monitor due to the lack of console or video output. Access to these output devices does not become available until late the in the boot process. If these output devices are also involved with the issue trying to be resolved then trouble shooting is even more difficult.

ROMWBW can be configured to display boot progress with the assistance of additional hardware. This can take the form of a front panel LED display or LED breakout debugging board connected to an 8-bit output port. Or it can utilize existing platform status LEDS.

As the boot code executes, the LED output display is updated to indicate the execution progress.

Platforms that have these capabilities built in have them enabled by default.

Front Panel display

A LED front panel or breakout board needs to be connected the computers data, reset and port select lines.

To enable this option the following settings can be made in the platforms custom configuration file.

```
FPLED_ENABLE    .SET    TRUE            ; ENABLE FRONT PANEL
```

Custom hardware can be configured with :

```
FPLED_IO    .SET    $nn         ; USE PORT ADDRESS nn
FPLED_INV   .SET    FALSE       ; INVERTED LED BITS
```

Platform Status LEDS

These status LEDs use preexisting status LEDs on each platform.

Enable using:

```
LEDENABLE   .SET    TRUE            ; ENABLES STATUS LED
```

Customize using:

```
LEDMODE     .SET    LEDMODE_STD     ; LEDMODE_[STD|SC|RTC|NABU]
LEDPORT     .SET    $nn             ; STATUS LED PORT ADDRESS
```

The following table shows the ROMWBW process steps in relation to the panel display.

PANEL	RomWBW Processes
. . . . .	Initial boot
	Jump to start address
	Disable interrupts
	Set interrupt mode
	Initialize critical ports and baud rate
. 0	Setup initial stack
	Memory manager and CPU configuration
	Set top bank to be RAM
. 00	Get and save battery condition
	Install HBIOS proxy in upper memory
	If platform is MBC reconfigure memory manager
	Setup “ROMLESS” HBIOS image or …
	Copy HBIOS from ROM to RAM if RAM flag not set
	Jump to HBIOS in RAM
	Set running in RAM flag
. 000	Finalize configuration for running in RAM
	Check battery condition

PANEL	RomWBW Processes
	Check for recovery mode boot
....0000	Identify CPU type
...00000	Set cpu oscillator speed
	Setup counter-timers
	Setup heap
..000000	Preconsole initialization
.0000000	Boot delay
	Set boot console device
	Bios announcement
00000000	Display platform information
	Display memory configuration
	Display CPU family
	Verify ROM checksum
	Report battery condition
	Perform device driver initialization
	Report watchdog status
	Mark HBIOS heap so it is preserved
	Switch from boot console to CRT if active
	Display device summary
	Execute boot loader

11.3.2 Appendix A Driver Instance Data fields

This section is a work in progress…

The following section outlines the read only data referenced by the `SYSGET`, subfunctions `xxxFN` for specific drivers.

TMS9918 Driver:

Name	Offset	Bytes	Description
PPIA	0	1	PPI PORT A
PPIB	1	1	PPI PORT B
PPIC	2	1	PPI PORT C
PPIX	3	1	PPI CONTROL PORT
DATREG	4	1	IO PORT ADDRESS FOR MODE 0
CMDREG	5	1	IO PORT ADDRESS FOR MODE 1
			Below are the register mirror values
			that HBIOS used for initialisation
REG. 0	6	1	$00 - NO EXTERNAL VID
REG. 1	7	1	$50 or $70 - SET MODE 1 and interrupt if enabled
REG. 2	8	1	$00 - PATTERN NAME TABLE := 0
REG. 3	9	1	$00 - NO COLOR TABLE
REG. 4	10	1	$01 - SET PATTERN GENERATOR TABLE TO $800
REG. 5	11	1	$00 - SPRITE ATTRIBUTE IRRELEVANT
REG. 6	12	1	$00 - NO SPRITE GENERATOR TABLE
REG. 7	13	1	$F0 - WHITE ON BLACK
DCNTL*	14	1	Z180 DMA/WAIT CONTROL

- ONLY PRESENT FOR Z180 BUILDS