Scientific Control Program

SCP/BIOS
Technical Reference Manual

Version 0.9

Written by: Yll Buzoku
for the SCP project

PREFACE

The Scientific Control Program Basic Input/Output System (SCP/BIOS) Technical Reference Manual describes the operation of the SCP/BIOS system on supported hardware. The information in this manual is designed to be both an introductory text and a reference for hardware or software developers, engineers and interested parties who wish to make use of the SCP/BIOS system in developing their own applications.

The user should be familiar with the notions of programming, in particular for the x86-64 family of processors and supporting hardware, and have some knowledge of the modern personal computer (PC) architecture.

This manual has two sections:

"Section 1: Hardware" describes the hardware for which SCP/BIOS is written. A technical description of how each driver functions and abstracts the hardware is also included in this section.

"Section 2: BIOS and its usage" describes the use of the BIOS by an application programmer. This section includes a description of how SCP/BIOS bootstraps itself, the bootstrapping process for a user application, a memory map of the system according to the SCP/BIOS requirements, software interrupt listings and a set of simple memory maps.

This publication also has five appendicies:

Appendix A: BIOS Listing

Appendix B: Character Set and Scan Codes

Appendix C: Using SYSDEBUG

Appendix D: Supported Hardware Configurations

Appendix E: CPU Exception Reference

A programmer programming in assembly for SCP/BIOS should be familiar with either the MASM or NASM assembler and its relevant syntax. The examples contained herein all use NASM syntax, and SCP/BIOS itself was written using the Netwide Assembler (NASM) version 2.15.

TABLE OF CONTENTS

Section 1: Hardware

Minimum System Requirements

SCP/BIOS has the following minimum system requirements:

- An x86-64 CPU.
- A “PC compatible” system board supporting legacy boot and at least one physical PS/2 port.
- At least 2MB of system memory.
- A PS/2 Keyboard.
- A VGA compatible graphics card or subsystem.

Additionally, it may be desirable to have the following features to enable full functionality:

- An RS-232 9-pin serial port.
- An EHCI controller.

The Intel/AMD x86-64 Processor and System Board

SCP/BIOS was written as a 64-bit BIOS for PC compatible machines utilising the x86-64 CPU architecture. Full effort was made to ensure that the instructions used are compatible across the entire x86-64 architecture line, thus enabling the code to run on the widest possible array of x86-64 based machines.

x86-64 CPU's support multiple running modes and addressing modes which can be mixed and matched to create a very complex programming environment. SCP/BIOS will run in long mode, the 64-bit mode of the CPU, which allows the programmer access to 16 64-bit general purpose registers. Though SCP/BIOS does not make use of them, all x86-64 compatible CPUs support FPU, MMX, SSE and SSE 2 instruction set extensions, to allow access to more registers and instructions which can act on more data at once. During system initialisation, SCP/BIOS switches from real mode to long mode directly, a little used method of entering long mode. In long mode, the number of addressing modes is reduced to one - memory paging. Unfortunately, x86-64 does not support a segmented memory model and as such, system software must set up memory paging. SCP/BIOS's paging model is designed to be transparent, i.e uses an identity paging mechanism, whereby physical memory address are mapped to virtual memory addresses in a bijective manner. This is done because the aim of SCP/BIOS is to provide the system's programmer with an environment that is as close to the real hardware as possible. SCP/BIOS only pages the the first 4 GB of system memory. A programmer can page more system memory should the need arise. A guide on how to do so whilst remaining compatible with SCP/BIOS is present in Section 2, SCP/BIOS Memory Map and Memory Paging.

Though x86-64 processors are multi-core CPU's, SCP/BIOS only initialises and uses one CPU. This will be changed in future revisions to allow for full multi-core support.

Finally, it is imperative that the system board supports and uses legacy boot (either through a ROM BIOS or an emulated ROM BIOS-like interface) for SCP/BIOS to function properly.

The Programmable Interrupt Controller

Note: The PIC will be replaced with the APIC in future revisions of SCP/BIOS

The Programmable Interrupt Controller or PIC for short is an emulated device that is usually built into the system board southbridge chipset. The PIC is functionally and programmatically compatible with the Intel 8259 chip. Its purpose is to provide multiple prioritised interrupt lines to the CPU, and a configurable programmable interface through which it can do so.

The PIC has the capacity for prioritised interrupts with eight priority levels. These eight priority levels are tied to eight interrupt request lines, IRQ 0-IRQ 7 which may be real or emulated. IRQ 0 is the highest priority interrupt line and IRQ 7 is the lowest priority line. The PIC has two registers through which commands are issued:

- The Command Register at I/O port 20H
- The Data Register at I/O port 21H

A command is issued to the PIC by writing the command byte to the PIC command register and then sending the relevant data byte to the PIC data register. For a full reference of commands, please see the Intel 8259 data sheet.

The PIC can operate in tandem with another PIC in the system. All system boards that emulate the PIC will emulate two PICs for compatibility with the IBM PC/AT architecture. SCP/BIOS will also initialise the two system PICs similar to the PC/AT standard. SCP/BIOS connects these two PICs (henceforth called PIC1 and PIC2) such that IRQ 2 of PIC1 is connected to IRQ 1 of PIC2. This connected structure (often called cascading interrupt structure) means that the system now has 16 levels of hardware interrupt IRQ 0 - IRQ 15. They are mapped such that system IRQ 8 is equivalent to IRQ 0 of PIC2. However, due to the “cascading interrupt” line of the two PICs, system IRQ2 is now PIC2 IRQ1 and system IRQ9 is PIC1 IRQ2.

SCP/BIOS maps these system IRQ 0- IRQ 15 to interrupts INT 20H - INT 2FH. Furthermore, it is important to note that PIC2 uses I/O address A0H for its command register and I/O address A1H for its data register.

A system programmer should generally leave the system PICs alone except for two purposes; masking and sending an End of Interrupt. Systems programmers should also write any interrupt handlers that are to run under

SCP/BIOS in such a way that provisions IRQ sharing. For more information, please read the section of Section 2, Interrupts 20H-2FH, Hardware Interrupts and IRQs on IRQ sharing.

Masking Interrupt Lines

Sometimes, it can be necessary to ensure that particular interrupt lines are not accidentally triggered high by the system. A programmer familiar with programming the x86 family of processors will issue a CLI command to ensure the processor does not respond to any external interrupts. However, the PIC will keep track of which IRQ was raised and when the programmer re-enables interrupts with the STI command, the CPU will be flooded with IRQ's Thus, the programmer can mask interrupt lines. Masking a particular IRQ line will force the PIC to simply ignore all signals on that particular IRQ line. Note, that as a consequence of how SCP/BIOS initialises the PIC, to mask all interrupts from PIC2, one needs only to mask IRQ2 on PIC1.

To mask an interrupt line/IRQ level, the programmer must simply set the bit corresponding to the IRQ line they wish to mask, i.e to mask system IRQ's 6 and 12, the programmer can issue the command

```
mov al, 40H      ;Bit 6 set
out 21H, al      ;PIC1 data register at 21H
mov al, 04H      ;System IRQ 12 = PIC2 IRQ 4
out 0A1H, al     ;PIC2 data register at 0A1H
```

The programmer can also read the current set mask for a particular PIC by doing in IN instruction on that particular PIC's data register.

End of Interrupt

Once the PIC receives an interrupt signal, it sets the interrupt pin high on the CPU. The PIC then places the IRQ number on the data bus, so that the CPU can know which interrupt occurred, and which interrupt handler to dispatch. The PIC will now not send another signal to the CPU, even if an interrupt occurs, until the CPU acknowledges to the PIC that it has completed the interrupt handling routine. This is done by the Interrupt handler, in the form of sending an End of Interrupt (EOI) message to the PIC. If the IRQ is on PIC1, then an EOI only needs to be sent to PIC1. If the IRQ is on PIC2, then an EOI needs to be sent, first to PIC2 and then to PIC1.

The following code snippet shows how to send an EOI in response to an IRQ that happened on PIC2:

```
EOI   equ  20H      ;EOI = 20H
mov  al,  EOI       ;
out  0A0H,  EOI     ;PIC2 command register at A0H
out  20H,  EOI      ;PIC1 data register at 0A1H
```

There is a third case that should be considered; the case of spurious interrupts which affect the interrupt handlers for IRQ 7 and IRQ 15. However, the default SCP/BIOS interrupt handlers handle this case, so as long as any custom installed interrupt handlers are written as outlined in the section on IRQ sharing provisions, then SCP/BIOS will handle this case for the programmer. The system also keeps count of how many spurious interrupts have occurred on both PIC1 and PIC2 separately, for diagnostic purposes.

The USB and EHCI support

SCP/BIOS makes very limited usage of the USB architecture to support accessing data from USB mass storage devices (MSD) such as flash drives. One of the main goals of SCP/BIOS is to completely abstract away the USB subsystem from the end user, only allowing users to interact with the devices on the USB and in the event of a needing to reset the bus system, to allow users that functionality. The user may investigate the internal structure of the USB subsystem included in SCP/BIOS though the user **must not** make any changes to any aspect of the USB subsystem that is not exposed via software interrupts, as doing so may leave the machine in an unusable state. For support with a wide array of machines, the Extended Host Controller Interface (EHCI) was chosen as the first USB controller type to have a driver written for it in SCP/BIOS. SCP/BIOS has full support for up to four separate EHCI buses per system.

At this stage, SCP/BIOS makes use of only the Asynchronous list using a very restricted model that has been dubbed the "SCP/BIOS ping-pong" model. The whole system has space allocated for only a single Asynchronous list, with space for two queue heads and ten transfer descriptors, which is shared between all buses as data can only be transferred to/from a single device at any one time. The "ping-pong" model works as follows: One EHCI controller is active at any one time. If a device on a different bus needs to be communicated with, the old EHCI Asynchronous list mechanism is stopped and a new self pointing queue head is placed in the Asynchronous list as the head of the Asynchronous list. The new EHCI controllers' Asynchronous list mechanism is then started pointing to this queue head. Once this queue head has been processed, the "ping" occurs, whereby the address of the next queue head is changed to the location of the next queue head. Once the next request is made, the queue head is written in the second queue head position, and the new queue head is linked to the old queue head via the Queue head horizontal link pointer. Once the transfer has been completed, the address of the next queue head then "pongs" back to the original position, the new queue head is then made the head of the Asynchronous queue and is made to point to itself and the process repeats until a device on a different EHCI bus needs to be communicated with. This simplified model allows for doing transfers with minimal initial memory allocation.

During system initialisation, the SCP/BIOS enumeration algorithm will speak to all the USB devices on the system and disable and ignore all USB ports that are empty or contain devices which do not comply with the SCP/BIOS driver requirements. These requirements are:

- The device must be USB 2.0 compliant.
- The device must be a USB High Speed device.
- The device must have the following USB class/subclass/protocol triples:
  - USB MSD, 08H/00H/50H or 08H/06H/50H
  - USB Hubs, 09H/00H/00H or 09H/00H/01H or 09H/00H/02H

USB Hub Class device support and Rate Matching Hubs

As of writing this document, there is very limited USB hub support, sufficient only to allow access to USB MSDs behind one tier of of USB hubs. This is written to allow systems with an Intel Integrated Chipset access to USB MSD's without disabling the integrated Rate Matching Hub (RMH) functionality of the chipset. RMHs are highly undocumented devices and their behaviour may vary from machine to machine, though at the time of writing this document no incompatibilities have been found with any systems containing RMHs. Note that this means that if an external hub is inserted into a port routed through a RMH then any devices, even valid MSD type devices, downstream of the external hub will not be enumerated.

SCP/BIOS has not been tested on *physical* systems without RMHs, or with systems where the USB ports of the root hub are exposed directly and MSD devices either inserted directly into a root port or inserted into a high speed external hub that is inserted into a root port and as such, such systems may experience some instabilities related to this. Please contact us directly by raising an issue on the SCPBIOS GitHub repository or sending an email directly if such an issue is encountered, giving as much information about the system configuration as possible.

Note, port extenders and head converters, such as those that convert USB-C type devices to USB-A type devices are implemented as USB hubs and therefore they must be treated as external hubs. Please be aware before inserting such devices into your system.

USB Mass Storage Device Class support

The USB Mass Storage Device (MSD) class of devices use a number of different but very similar communication protocols. At the time of writing this document, only the “Bulk Only Transport” protocol is supported, which encapsulates most types of USB flash drives, which is the target mass storage medium of SCP/BIOS. As noted previously in the EHCI section, the following class/subclass/protocol triples of MSD devices are supported:

- 08H/00H/50H ↔ MSD Class/SCSI command set not reported/Bulk Only Transport
- 08H/06H/50H ↔ MSD Class/SCSI transparent command set/Bulk Only Transport

Some other common USB MSDs, notably USB Floppy Drives, belong to a different subclass of devices, called the UFI subclass. These devices are not yet supported though support for such devices will be introduced very shortly, thereby adding a new protocol type, the CBI protocol, and a new subclass of devices, the UFI subclass, to allow for an even wider range of USB MSD class devices that is supported under SCP/BIOS. Of note is that USB MSD class devices communicate with the host system using a subset of the SCSI command set and that most protocols and subclasses involve some subset of the SCSI command set with specific bytes changed (for an example, please refer to the Format Unit command in the UFI specification vs Format Unit command in the USB specification for Bootability).

The SCP/BIOS USB MSD driver does not support any features in a MSD that do not belong to LUN 0, that is to say, if a device has two logical units, only the first logical unit will ever be accessed.

The SCP/BIOS USB MSD class initialisation procedure issues the following commands as part of a MSD initialisation routine:
In the following cases, if the device stalls, the stall is cleared. If the stall cannot be cleared, enumeration is cancelled.

- MSD Reset
- MSD Report LUNS
- MSD Reset
- SCSI Inquiry (12)

After this, if a command returns fail, a Request Sense command is issued. If a device needs to Request Sense more than 5 times, it fails enumeration.

- SCSI Request Format Capacities (10)
- SCSI Request Mode Sense (6)
- SCSI Request Capacity (10)

This sequence of commands was chosen as a minimal set of commands which follows roughly what some major operating systems issue to MSDs during their initialisation procedure of MSDs. After this sequence has successfully completed, the device will be successfully written to the internal SCP/BIOS data tables, and will be ready for use. Unfortunately, some USB Flash drives, respond poorly to the previously outlined set of commands and require specific initialisations and require certain commands to be issued in a more particular order. Such devices are not officially supported under SCP/BIOS as they violate the SCP/BIOS enumeration procedure.

SCP/BIOS's recommended USB MSD device

During testing, three Verbatim 16Gb Slider USB drives[1] were used, with great success. As a result, the SCP/BIOS team highly recommends users acquires such a flash drive or similar. They proved extremely reliable and seemed to adhere very well to the USB standards, behaving exactly as expected. Please refer to the SCP/BIOS GitHub repository where a list of supported USB MSD devices will be added to as time goes on. A list of supported devices and links to said purchase devices will also be available in Appendix D.

[1] This is not a sponsored endorsement.

Asynchronous Serial Communication Adapter support

The serial communication adapter, commonly known as a serial port, is a legacy device, which is still commonly found on computers today. It can be used to communicate between computers, or between computers and terminal stations via a physical link - the 9- or 25-pin serial cable. This allows for easy networking between computers. Serial communication adapters may be either built in to a computers' system board and be presented as either a header on the system board or have have a full 9- or 25-pin port on the system board, or may exist as a separate PCI or PCIe card, inserted into the computer bus. At the heart of the serial communication adapter is the Universal Asynchronous Receiver Transmitter (UART) chip. The UART is usually a NS16550 type chip or functionally equivalent, but *may* be an older NS16450 or INS8250 type chip. Newer type UARTs may also be used in a Serial Communication Adapter however, they are also generally functionally compatible with the NS16550. At present time, SCP/BIOS has no provisions for distinguishing between UART chip types and thus it is recommended that the user assumes that if there is a serial communication adapter, that it contains a NS16550 UART. All three quoted UARTs are functionally compatible however, the NS16550 has additional features that are not present in the NS16450 or the INS8250 UARTs.

A table comparing the main differences between the three quoted UARTs is provided below:

Table 1: Features of different UARTs

UART	Maximum Recommended Baud Rate	FIFO?
INS8250	9600 Baud	No
NS16450	115200 Baud	No
NS16550	115200 Baud	Yes

In the following sections if the word UART is used without a identifying chipset, it can be taken to mean any of the three above quoted UARTs.

Asynchronous serial communications adapters contain fully programmable Baud Rate generators with a clock which oscillates at 1.8432Mhz. The baud rate is calculated by using a baud rate divisor in a similar fashion to the Programmable Interval Timer. The INS8250 UART allows for baud rates between 50 baud to 9600 baud, becoming unstable at higher baud rates, with the 16450 and 16550 UARTs extending this to 115200 baud. The serial communication adapters' UART allows for serial communications using 5-, 6-, 7- or 8-bit words with 1-, 1.5- or 2-stop bits and has a programmable parity bit detection and generation setting for even, odd or no parity bit settings. The UART is also capable of automatic detection of false start bit transmission and has a loopback functionality, to allow for easy device and driver testing. The UART automatically strips all start, stop and parity bits from a data packet, thus freeing the computer to read received words directly from the UART's receive buffer. The UART additionally has a fully programmable interrupt interface with prioritised interrupt levels which a programmer can set under which conditions the UART will trigger an interrupt, and the UART will handle triggering interrupts under the programmed conditions. The interrupt handler can then discern what the reason for the interrupt was and how best to handle the interrupt. A unique aspect of the NS16550 type UART (and newer) is that the programmer can enable the UART to use a 16 byte FIFO instead of the standard byte buffer to buffer both input and output. In addition the programmer can program the NS16550 UART to raise an interrupt by using an additional FIFO interrupt trigger feature to trigger an interrupt after either 1, 4, 8 or 14 bytes have entered the FIFO buffer. Finally, all UARTs also have a standard MODEM control interface which it can use to govern serial communications with another computer, peripheral or MODEM.

SCP/BIOS provides additionally buffered support for up to four attached serial communication adapters on a machine and reserves two interrupt channels for serial communication adapters with serial ports 2 and 4 programmed to use IRQ 3 and serial ports 1 and 3 being programmed to use IRQ 4. The user is free to reprogram attached serial communication adapters however they see fit, choosing to either use the built in SCP/BIOS drivers or replacing them as needed. The serial communication adapter's UART registers are accessed via the processors' I/O instructions. Each supported serial communication adapter has its base I/O address as follows:

- Adapter 1 - I/O Address 03F8H
- Adapter 2 - I/O Address 02F8H
- Adapter 3 - I/O Address 03E8H
- Adapter 4 - I/O Address 02E8H

Note, that if a serial communications adapter is not programmed to respond at one of these addresses or there is a gap in the addresses (for example if a user has three adapters at I/O addresses 03F8H, 02F8H and 02E8H) then the devices after the gap will not be enumerated and will not be usable via the SCP/BIOS programming interface. These issues can be usually rectified by setting the device address either on the card itself using DIP switches or via your system boards' ROM BIOS. Please refer to your particular adapters technical reference documentation for more information.

For serial communication at greater than 9600 baud, when using the 16550 UART it is recommended that the user reprograms the FIFO programmable interrupt trigger feature to either 4, 8 or 14 bytes.

At system initialisation, SCP/BIOS will set each serial adapter to 2400 Baud, 8 bits, no parity bits, 1 stop bit, with a 1 byte FIFO interrupt trigger to emulate the behaviour of the INS8250 UART.

The serial communication adapter's UART registers are described in Table 2. Each register is 8-bits in width with bit 0 being the least significant bit. The base column in Table 2 refers to the offset of the register from the base I/O address of the serial communication adapter. Note that all registers except those marked as "16550 UART only" exist for all three aforementioned UART types, though not all options in these common registers will exist for each aforementioned UART type. These cases will be explicitly denoted in the register descriptions below:

Table 2: UART register definitions and I/O addresses

Base	R/W	Register Name	Notes
+0	WO	Transmit Holding Register (THR)	LCR Bit 7 = 0
+0	RO	Receiver Buffer Register (RBR)	LCR Bit 7 = 0
+0	R/W	LSB of Divisor Latch (DLL)	LCR Bit 7 = 1
+1	R/W	MSB of Divisor Latch (DLM)	LCR Bit 7 = 1
+1	R/W	Interrupt Enable Register (IER)	LCR Bit 7 = 0
+2	RO	Interrupt Identification Register (IIR)	N/A
+2	WO	FIFO Control Register (FCR)	16550 UART only
+3	R/W	Line Control Register (LCR)	N/A
+4	R/W	MODEM Control Register (MCR)	N/A
+5	RO	Line Status Register (LSR)	N/A
+6	RO	MODEM Status Register (MSR)	N/A
+7	R/W	Scratch Register (SCR)	N/A

The descriptions of each register and how to program them are described below:

Transmit Holding Register, WO at BASE + 0, LCR Bit 7 = 0
This register is used to provide the serial communication adapter with the

word to transmit. The data bits are written by writing the least significant bit of the word to transmit to the least significant bit of the register. This data is then shifted into the internal Transmit Shift Register and out onto the serial line, where additional bits may have been added to form the transmitted data packet. This register is frequently known as the TX Buffer.

Receiver Buffer Register, RO at BASE + 0, LCR Bit 7 = 0
This register contains the received word from the serial line. The data bits are read such that Bit 0 of this register is the least significant bit of the received word and is the first bit serially received by the serial communication adapter. This register is frequently known as the RX Buffer.

LSB of Divisor Latch, R/W at BASE + 0, LCR Bit 7 = 1
This register contains the least significant byte of the Divisor Latch value.

MSB of Divisor Latch, R/W at BASE + 1, LCR Bit 7 = 1
This register contains the most significant byte of the Divisor Latch value.

The divisor latch values can be calculated by dividing the Baud Rate generator frequency of 1.8432MHz by the desired baud rate, to obtain a value that is 16 times larger than the divisor as in the following formula:

$$\text{Baud Rate Divisor} = \frac{1.8432\text{MHz}}{\text{Baud Rate} \times 16}$$

Table 3 illustrates some common baud rates and their divisor values multiplied by 16.

Table 3: Baud rates and divisors at 1.8432MHz

Desired Baud Rate	Baud rate Divisor multiplied by 16
50 Baud	2304 = 0900H
75 Baud	1536 = 0600H
110 Baud	1047 = 0417H
134.5 Baud	857 = 0359H
150 Baud	768 = 0300H
300 Baud	384 = 0180H
600 Baud	192 = 00C0H
1200 Baud	96 = 0060H
1800 Baud	64 = 0040H
2000 Baud	58 = 003AH
2400 Baud	48 = 0030H
3600 Baud	32 = 0020H
4800 Baud	24 = 0018H
7200 Baud	16 = 0010H
9600 Baud	12 = 000CH
19200 Baud	6 = 0006H
38400 Baud	3 = 0003H
57600 Baud	2 = 0002H
115200 Baud	1 = 0001H

Warning!
You must not attempt to set the baud rate divisor to 0. Doing so may damage your serial communication adapter!

Interrupt Enable Register, R/W at BASE + 1, LCR Bit 7 = 0
This register can be used to set which on which events you wish the UART to raise an interrupt on. By writing a 0 to all setting bits, a programmer can effectively disable the UART's ability to raise an interrupt as the device will not raise an interrupt under any circumstances. Table 4 gives a description of the bits of the IER register which explain under what circumstances a programmer can program the UART to raise an interrupt.

Table 4: Interrupt Enable Register Bit definitions

Bit	Function
7-4	Reserved, always 0
3	1 = Enable MODEM Status Interrupt
2	1 = Enable Receiver Line Status Interrupt
1	1 = Enable Transmitter Holding Register Empty Interrupt
0	1 = Enable Receive Data Available Interrupt.

If the UART is in FIFO mode, bit 0 also enables the time-out interrupt, which occurs if there is at least one word in the FIFO for a time equivalent to the transmission of four words.

Interrupt Identification Register, RO at BASE + 2

The Interrupt Identification Register can be used to identify what was the reason for the UART firing the interrupt if an interrupt is fired. It also serves a second purpose, in that it can be used to identify if the UART is a NS16550 or functionally compatible or an older UART chip. This is done by setting Bit 0 of the FIFO control register (FCR, BASE + 2) to enable the UART FIFO, and then reading bit 7 of the IIR. If this bit is set, the UART is functionally compatible with the NS16550 UART. If it is not set, then the device is an older type of UART. Table 5 gives a description of the bits of the IIR register and under what circumstances a programmer the might expect to find a bit set.

Table 5: Interrupt Identification Register Bit definitions

Bit	Function
7	1 = FIFO enabled, 0 = FIFO not enabled
5-4	Reserved, Set to 0
3-0	Interrupt Control Function bits

The meanings of the Interrupt Control Function bits and the appropriate action to take when they are set can be read from Table 5.

Table 6: Interrupt Control Functions

Interrupt ID Bits	Interrupt Set and Reset Actions			
Bits 3-0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Action
0001b	-	None	None	-
0110b	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Error	Reading the Line Status Register
0100b	Second, Shared	Recieved Data Available	Reciever Data Available or If in FIFO mode, the FIFO interrupt threshold has been reached or surpassed	Reading the Reciever Buffer Register or If in FIFO mode, reading until the FIFO has fewer words than the interrupt threshold
1100b	Second, Shared	FIFO Timeout Interrupt	FIFO Buffer contains at least a word which has been present for the duration of four transmitted words or No new word have been recieved in the same period	A word is read from the FIFO or A new word enters the FIFO
0010b	Third	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Reading the IIR register (if it is the source of the interrupt) or Writing to the Transmitter Holding Register
0000b	Fourth	MODEM status	Clear To Send or Data Set Ready or Ring Indicator or Received Line Signal Detect	Reading the MODEM Status Register

FIFO Control Register, WO at BASE + 2, 16550 compatible UART only

The FIFO control register allows the programmer to program the UART to control various aspects of the FIFO architecture. The programmer can program the UART to either work in character mode (without the FIFO enabled, with a single word receive/transmit buffer) or in FIFO mode (with the FIFO enabled, with a 16 word receive/transmit buffer). If the programmer wishes to use the UART in FIFO mode, they may set the threshold at which an interrupt is raised by the UART by setting bits 7 and 6 of this register. The bit definitions can be read from Table 7 below.

Table 7: FIFO Control Register Bit definitions

Bits	Function
7-6	Reciever FIFO Register Trigger 00b = 1 byte 01b = 4 bytes 10b = 8 bytes 11b = 14 bytes
5-3	Reserved, 0
2	1 = Clears transmitter FIFO and resets the transmitter FIFO counter. The shift register is not cleared. 0 = No effect. This bit is Write Clear.
1	1 = Clears receiver FIFO and resets the receiver FIFO counter. The shift register is not cleared. 0 = No effect. This bit is Write Clear.
0	0 = Clears reciever and transmitter FIFOs and enters Character Mode. 1 = Recieve and Transmit FIFOs enabled and enters FIFO Mode.

Line Control Register, R/W at BASE + 3

The Line Control Register allows the programmer to specify the format of the data that is to be exchanged using the serial adapter. In addition to being able to specify the format of the data to be exchanged, a programmer can also read this register to get the current format of the data that may be exchanged by the serial adapter. The bit definitions are as follows:

Table 8: Line Control Register Bit definitions

Bits	Function
7	Divisor Latch Access Bit (DLAB) 1 = BASE + 0, BASE + 1 form the divisor latch. 0 = BASE + 0 are THR and RBR. BASE + 1 is IER.
6	Set Break Control Bit. 1 = Transmit BREAK to alert a connected device. 0 = In normal operation.
5	Stick Parity Bit.
4	Even Parity Enable Bit. 1 = Even Parity 0 = Odd Parity
3	Parity Enable Bit. 1 = Enable Parity Bit 0 = Disable Parity Bit
2	Number of stop bits Bit. 0 = 1 stop bit 1 = 1.5 stop bits if 5-bit words 2 stop bits otherwise
1-0	Word length Bits. 00b = 5-bit word 01b = 6-bit word 10b = 7-bit word 11b = 8-bit word

It must be noted that the LCR simultaneously governs the format of data both received and transmitted. Thus we have that

- Bits 0 and 1 govern the word length that will be transmitted and received by the UART.
- Bit 2 governs the number of stop bits to be added to data and the number of stop bits the UART will check for in each packet of data.
- Bit 3 governs whether the UART will add a parity bit and check for a parity bit in each packet of data. The parity bit is used to produce an even or odd number of 1's, according to Bit 4, when the data word bits and the parity bit are summed.
- Bit 4 governs whether the number of parity bits added to or checked for by the UART will be odd or even.
- Bit 5 logic works as follows. If Bit 3 = 1 and Bit 5 = 1, the parity bit is transmitted and detected by the receiver as a 0 if bit 4 = 1 or as a 1 if bit 4 = 0.
- Bit 6 is used to halt all communication on the line and alert the computer/device on the other side of the connection by sending a BREAK command.
- Bit 7 should only be set if the programmer wishes to load or read the current Baud Rate divisor value. The programmer should remember to clear this bit after accessing the Baud Rate divisor, to return the UART to a regular working state.

MODEM Control Register, R/W at BASE + 4
The MODEM control register controls the interface of the serial adapter with a MODEM or a device emulating a MODEM. The bit definitions are as follows:

Table 9: MODEM Control Register bit definitions

Bits	Function
7-5	Reserved, 0
4	Loopback control Bit 1 = Enable Loopback 0 = Disable Loopback
3	OUT 2 control Bit
2	OUT 1 control Bit
1	Request to Send Control Bit 1 = Assert RTS line 0 = Deassert RTS line
0	Data Terminal Ready Control Bit 1 = Assert DTR line 0 = Deassert DTR line

A point about Bits 0-3 of the MCR is that by writing a 0 to these bits, the corresponding lines to which they are connected are inverted high. Similarly, by writing a 1 to these bits sets the corresponding lines low.

- Bit 0 controls the level of the Data Terminal Ready (DTR) line, and indicates to an attached device that the host system is ready to receive a data packet.

- Bit 1 controls the level of the Request to Send (RTS) line, and indicated to an attached device that the host system is ready to send a data packet.

- Bit 2 controls the auxiliary implementation specific OUT 1 signal.

- Bit 3 controls the auxiliary OUT 2 signal. This bit must be cleared for interrupts that are raised by the UART to be propagated onto the system bus.

- Bit 4 controls the loopback feature of the UART. If this bit is set, the MODEM control inputs (CTS, DSR, DCD and RI) are disconnected from the physical port and are connected internally to the MODEM control outputs (DTR, RTS, OUT1, OUT2). Additionally the output of the THR is immediately looped back into the RBR. Whilst in this mode, all interrupt features work as normal and are set to be triggered as usual using the IER, and a programmer can artificially trigger any type of interrupt as the bottom four bits of the LSR are directly con-

nected to the bottom four bits of the MCR. That allows a programmer to test an interrupt handler whilst the device is in this loopback mode, by setting the bottom four bits of the MCR however they deem necessary.
To return the system back to a normal operating mode, the programmer must reprogram the bottom four bits of the MCR as needed, before then clearing bit 4 of the MCR.

Line Status Register, RO at BASE + 5
This register gives the programmer the ability to read the current state of the line. Reading this register is particularly useful in interrupt processing as this register tells the programmer the reason(s) for the interrupt having been raised, as well as the current state of the line. The bit definitions of this register are as follows:

Table 10: Line Status Register Bit definitons

Bits	Function
7	UART 16550 specific
6	1 = Transmit Shift Register empty (TSRE)
5	1 = Transmit Holding Register empty (THRE)
4	1 = Break Interrupt
3	1 = Framing Error
2	1 = Parity Error
1	1 = Overrun Error
0	1 = Data Ready

Bits 0-5 will trigger an interrupt if the bit corresponding to each condition is set in the IER register. Each bit of the LSR corresponds to the following conditions:

- Bit 0 is set to 1 whenever a complete incoming word has been received by the UART and has been transferred into the RBR. This bit is reset to 0 by reading the RBR.
  If a UART 16550 type serial adapter is in FIFO mode, this bit is set to 1 whenever there is at least one complete word in the FIFO. This bit remains 1 until the FIFO is emptied.

- Bit 1 is set to 1 whenever a complete incoming word has been received by the UART and was moved into the RBR before the RBR was read by the host system. This bit is reset to 0 by reading the LSR.
  If a UART 16550 type serial adapter is in FIFO mode, this bit is set to 1 whenever the FIFO is full and the receive shift register continues to receive more complete words. These additional words override each other and are not stored in the FIFO. This bit is reset to 0 as before.
- Bit 2 is set to 1 whenever a received word does not have a correct even or odd parity. This bit is reset to 0 by reading the LSR.
  If a UART 16550 type serial adapter is in FIFO mode, this bit is set to 1 whenever the word at the top of the FIFO has incorrect parity. This bit is reset to 0 as before.
- Bit 3 is set to 1 whenever a received word does not have valid stop bits. This bit is reset to 0 by reading the LSR.
  If a UART 16550 type serial adapter is in FIFO mode, this bit is set to 1 whenever the word at the top of the FIFO has incorrect stop bits. This bit is reset to 0 as before.
- Bit 4 is set to 1 whenever the received data word is held in Spacing mode (set to 0) for longer than a full word transmission time, that is to say, for longer than it would take to transmit a Start bit, all n-data bits, a parity bit and the stop bits together. This indicated to the host that a BREAK has been sent and a 0 word is placed in the RSR. This bit is reset to 0 by reading the LSR.
  If a UART 16550 type serial adapter is in FIFO mode, this bit is set to 1 in non-FIFO mode, with a 0 word being placed in the FIFO. This bit is reset to 0 as before.
- Bit 5 is the Transmitter Holding Register Empty (THRE) indicator bit. This bit indicates that the THR is ready to accept a new character for transmission. If the Transmit Holding Register Empty Interrupt bit is set in IER (bit 1), then this bit being set will trigger that interrupt. This bit goes high upon transferring a word from the THR to the Transmit Shift Register. This bit is reset to 0 by loading a new word into the THR.
  If a UART 16550 type serial adapter is in FIFO mode, this bit is set to 1 if the output FIFO is empty. This bit is reset to 0 as before.
- Bit 6 is the Transmitter Shift Register Empty. This bit is set to 1 whenever the Transmitter Shift Register is idle, i.e. not transmitting data on the serial line. This bit is set to zero when a word has been

transferred from the THR to the Transmitter Shift Register.
If using a UART 16550 type serial adapter, this bit is set to 1 both the FIFO and the THR are empty. This bit is reset to 0 as before.

- Bit 7 is reserved as 0 for UARTs older than the 16550. On a serial adapter with a UART 16550, this bit indicates that there is a word in the FIFO queue that was received with a Parity, Framing or Break error. This bit is cleared by reading the byte from the FIFO. Using this, the programmer can thus identify and discard the erroneous word. If the UART is not operating in FIFO mode, than this bit indicates that the word in the RBR was received with a Parity, Framing or Break error.

MODEM Status Register, RO at BASE + 6

This register gives the current state of the control line from the MODEM or peripheral device to the host system. This includes four “delta” bits which indicate that a change has occurred on the other four bits of the register since the last time the MODEM Status Register has been read. The MODEM Status Register bits are defined as follows:

Table 11: MODEM Status Register Bits

Bits	Function
7	1 = Data Carrier Detect (DCD) asserted
6	1 = Ring Indicator (RI) asserted
5	1 = Data Set Ready (DSR) asserted
4	1 = Clear to Send (CLS) asserted
3	1 = Delta DCD (DDCD)
2	1 = Trailing Edge Ring Indicator (TERI)
1	1 = Delta DSR (DDSR)
0	1 = Delta CTS (DCTS)

Of note is that the upper 4 bits indicate the current state of the MODEM connection, with the lower 4 “delta” bits indicating that the corresponding upper bit has changed state since the last read of the MSR. If MODEM Status

Interrupts are enabled (Bit 3 of IER), then if a condition arises such that any of Bits 0-3 of the MSR are set, then the UART will trigger an interrupt. These “delta” bits are reset to 0 by reading the MSR. The programmatic bit descriptions are as follows:

- The CTS bit is intended to be set if the attached device is ready to receive a data packet from the UART.
- The DSR bit is intended to be set by the attached device if it is ready to transmit a data packet to the UART.
- The RI bit intended to be set by the attached device to get the device’s attention.
- The DCD bit, sometimes called the Received Line Signal Detect (RLSD) bit, intended to indicate that a MODEM or MODEM-like device has been attached to the serial adapter.

Note finally that if Bit 4 of the MCR is set, i.e. the device is in loopback mode, then the following connections are made.

- The Clear To Send (CLS) bit of the MSR is connected to the Request To Send (RTS) bit of the MCR.
- The Data Set Ready (DSR) bit of the MSR is connected to the Data Terminal Ready (DTR) bit of the MCR.
- The Ring Indicator (RI) bit of the MSR is connected to the OUT1 bit of the MCR.
- The Data Carrier Detect (DCD) bit of the MSR is connected to the OUT2 bit of the MCR.

Scratch Register, RO at BASE + 7

This register is a single, 8-bit storage space that can be used by a programmer to store a single byte of data. It is believed, at the time of writing, that this storage register was intended as a temporary storage location for data that might be have been transmitted to the UART in the event that the host system did not have anywhere in system memory to put the byte. To allow the host system to remove the byte from the RBR, this byte might have been provided. However, it is up to the programmer to decide what they use this byte location for.

Enabling Interrupts for a Serial Adapter

The programmer should be aware that they can configure the interrupt subsystem of a UART without any need to mask the hardware interrupts on the system bus, in the event that the programmer needs to hook, write or test interrupt handling routines. This is because the interrupt line of all compatible serial adapters are multiplexed with the OUT2 bit of the MCR register. That is, assuming the CPU is operating with hardware interrupts on (i.e. a STI instruction has been executed and no CLI has followed it) and that the IRQ levels into the (Advanced) Programmable Interrupt Controller on the system board are unmasked, then

- To enable interrupts from the Serial Adapter, a programmer must first configure which events they wish the UART to trigger an interrupt on, and then clear OUT2, i.e. Set bit 3 of the MCR to 0.
- To disable interrupts from the Serial Adapter, a programmer must set OUT2 to 1.

Video Graphics Array Support

The SCP/BIOS implementation of VGA support at this time is minimal and as such this section of the documentation will focus on programming with the features of the VGA that are supported by SCP/BIOS. A brief overview of the features of the VGA used in SCP/BIOS include:

- An 80x25 text mode display resolution.
- 8x16 Character cells.
- Separate 4-bit colour support for character cells and characters.
- 256 display characters.
- 32 KB of display memory.
- Up to 8 separate display pages.
- Fully programmable CRTC support.
- Hardware cursor support.

Warning! The programmer may damage their monitor or adapter by attempting to reprogram the VGA using registers and features not explicitly mentioned in the following section. This includes attempting to set your own display modes. SCP/BIOS is presently unable to switch display modes once it has initially set up the VGA. The programmer must refrain from attempting to switch display modes by directly accessing the VGA registers, unless they are absolutely sure they know what they are doing.

Modes of Operation

SCP/BIOS sets up the VGA into “VGA Mode 3”, more commonly known as “Text mode”. This display mode is an 80 Column by 25 Line display mode, allowing for 4000 character cells that are 8 pixels wide by 16 pixels high to be simultaneously displayed, taking up a 4KB screen buffer. The VGA allows for up to 8 pages of text to be written at once, with each page taking up 4KB of memory, though only one page may be displayed at any one time. Graphics modes and other Text modes are not supported at this time but are planned for a future release.

The VGA in this mode allows for 256 characters which are defined by the graphics adapter, though broadly speaking, they follow the standard IBM Extended ASCII Character Set, which includes drawing characters. To write a character to the display, the programmer must write a 16-bit word to the adapters' display buffer, with the lower 8-bit character code being saved at an even address and the upper 8-bit attribute code, which governs the properties of the character, being saved at an odd address, that is one greater than the character the attribute code is attributed to.

The character attributes can be read as two 4-bit fields, with the upper nybble defining the properties of the background of the character cell and the lower nybble defining the properties of the foreground. These properties can be combined to produce an array of colourful text and are defined as follows:

Table 12: Various VGA Mode 3 colour attributes

xGround				Hex Value	Colour
I	R	G	B		
0	0	0	0	0	Black
0	0	0	1	1	Blue
0	0	1	0	2	Green
0	0	1	1	3	Cyan
0	1	0	0	4	Red
0	1	0	1	5	Purple
0	1	1	0	6	Brown
0	1	1	1	7	Light Grey
1	0	0	0	8	Dark Grey
1	0	0	1	9	Light Blue
1	0	1	0	A	Light Green
1	0	1	1	B	Light Cyan
1	1	0	0	C	Light Red
1	1	0	1	D	Light Purple
1	1	1	0	E	Yellow
1	1	1	1	F	White

In Table 12, xGround refers to either the Foreground or the Background. There is one difference between the two. The intensity bit, I, does not produce light colours in the case of the background, instead becoming a flashing toggle bit. This means that if a value greater than 7 is used for the background attribute nybble, then the foreground character will flash.

Programming Considerations

As implemented in SCP/BIOS, there are two aspects of the VGA that a programmer should primarily concern themselves with; manipulating display memory and programming the CRTC.

Manipulating Display Memory

The 32KB display memory buffer for the VGA is set up to be between memory addresses 0B8000H-0BFFFFH. The VGA in this mode has support for up to 8, 4KB display pages and these pages are laid out in memory as shown in Table 13.

Table 13: VGA Mode 3 Memory Pages

Page Number	Base Address
Page 0	0B8000H
Page 1	0B9000H
Page 2	0BA000H
Page 3	0BB000H
Page 4	0BC000H
Page 5	0BD000H
Page 6	0BE000H
Page 7	0BF000H

A programmer can write to these pages and rapidly swap between them by manipulating the CRTC start address registers or by using the SCP/BIOS video services, to avoid redrawing screens of text in an application. Furthermore, SCP/BIOS keeps track of all screen pages allowing each page to have an independent cursor.

Programming the CRTC

The CRTC used in the VGA is derived from the Motorola 6845 CRT Controller chip. The CRTC is accessed by first writing the index of the CTRC register you wish to access to I/O address 03D4H, the CRTC Index Register, and then sending the data you wish to send, or reading (if the register is R/W) the data to/from I/O address 03D5H, the CRTC Data Register. Table 14 gives a table with the CRTC register values.

Table 14: CRTC Register Description

Register Index	Register Name	R/W	Units
00H	Horizontal Total	WO	Char.
01H	Horizontal Displayed	WO	Char.
02H	HSync Position	WO	Char.
03H	HSync Width	WO	Char.
04H	Vertical Total	WO	Char. Row
05H	Vertical Adjust	WO	Scan Line
06H	Vertical Displayed	WO	Char. Row
07H	VSync Position	WO	Char. Row
08H	Interlace Mode	WO	-
09H	Max Scan Line Address	WO	Scan Line
0AH	Cursor Start	WO	Scan Line
0BH	Cursor End	WO	Scan Line
0CH	Start Address (H)	WO	Regen Offset
0DH	Start Address (L)	WO	Regen Offset
0EH	Cursor (H)	R/W	Regen Offset
0FH	Cursor (L)	R/W	Regen Offset
10H	Reserved	-	-
11H	Reserved	-	-

In Table 14, the unit Regen offset means that these registers take a value that is equal to the number of character cells from the base of the display memory region at 0B8000H. By manipulating the Start Address registers, the programmer can set the base of the 4KB region that they wish to display on screen to anywhere within in 32KB region between 0B8000H and 0BFFFFH.

The programmer can similarly manipulate the cursor position by manipulating the cursor registers. The Cursor Start and Cursor End registers can be used to manipulate the size of the cursor or to hide the cursor completely.

Using SCP/BIOS functions
In any case, unless it is absolutely necessary, a programmer should access the VGA by using the SCP/BIOS video services as described in Section 2. Using SCP/BIOS video services will make an application program much more portable and will allow the programmer access to additional screen modes when they become available, without having to concern themselves with the calculations and conversions needed to deal with the potential different memory layouts, different page or buffer sizes or even the specifics of different video adapters.

PS/2 Controller and Keyboard Support

The PS/2 protocol is a bidirectional serial communications protocol that is an extension of the IBM PC AT keyboard protocol. The protocol uses odd parity with an 8-bit data word (sending the least significant bit first), one start bit (that is indicated by a 0), one stop bit (that is indicated by a 1) and when the PS/2 controller is communicating to a PS/2 device, an additional acknowledgement bit is appended to the end of the data packet. On most desktops, the physical layer uses a 6-pin mini-DIN type though on laptops, the physical layer may be implemented via a proprietary connector.

SCP/BIOS has full support for PS/2 keyboards, including an interrupt handler to asynchronously read bytes from a PS/2 keyboard into a circular buffer so that the host system can read bytes from the buffer on demand. The buffer is large enough to store information about up to 16 keystrokes which is a value large enough to keep up with a fast typist. If the buffer should fill up, either due to a really fast typist or the host system not reading keystroke information from the buffer, then the host system will sound a beep from the PC speaker.

The Intel 8042 microcontroller is a device which interfaces with PS/2 devices, and is utilised by SCP/BIOS during system initialisation to ascertain if a PS/2 Keyboard is attached to your computer. It is known as the PS/2 controller in this context. Note that most system boards no longer have a physical i8042 on them, and emulate its behaviour. If your system board poorly emulates the i8042 chip, this could cause operational problems. The SCP/BIOS PS/2 driver disables all emulation layers that may exist on top of the i8042 emulation to get as close to the hardware as is possible, so it is hoped that buggy emulation layers will not cause any issues. The PS/2 subsystem was chosen to be supported due to the ease of programming a PS/2 keyboard driver and the fact that most system boards still have a PS/2 header on them and that most laptops systems use the PS/2 protocol for their keyboards.

The Keyboard Buffer

The keyboard buffer is a 32 byte buffer capable of storing data about at most 16 keystrokes. It is located in the BIOS Data Area. Information about the modifier keys are also saved in bitfields in the BIOS Data Area. The SCP/BIOS keyboard driver also supports the setting and resetting of the three keyboard state lights, that is “Num Lock”, “Caps Lock” and “Scroll Lock”. The driver also has support for handling CTRL + BREAK events, by way of calling a user interrupt handler, Interrupt 3BH.

Data is stored in the buffer in 16-bit words. Each entry is split into a low and a high byte. The low byte contains the ASCII value of the key, and corresponds to the IBM Extended ASCII character to be placed on screen. The high byte contains the scan-code, the unique keyboard key identifier. Not all keys are placed into the buffer with an ASCII symbol. In such cases, an ASCII NUL (00h) value is placed in the low byte of the buffer entry. Examples of such keys include the function keys. The keyboard driver has full support for all the standard modifier keys and combinations of modifier keys with all other keys. The driver also fully supports for the SysReq key, F1-F12 function keys, Pause/Break handling and CTRL+ALT+DEL.

In the event the user wishes to quickly restart the system, the user may hold the Control and Alt modifier keys and strike the Delete key. By doing so, Keyboard Controller pulses the CPU reset line, thus placing the CPU into a reset state. Once the reset is complete, the CPU will restart the processing. If for some reason, this fails, or your keyboard controller is incapable of pulsing the CPU reset line, a CTRL modified Delete character is placed into the keyboard buffer.

Modes of Operation

In most situations a user should not have to communicate with either the PS/2 controller or the PS/2 keyboard directly. To communicate with the PS/2 controller or keyboard, two I/O ports are used. They are defined as follows:

Table 15: PS/2 subsystem I/O addresses

I/O Addr	Port Name	R/W
060H	Data Port	R/W
064H	Status Port	RO
064H	Command Port	WO

- Data port 060H can be used to read a data byte from a PS/2 device or write a data byte to a PS/2 device. When a keyboard key is pressed, the keyboard places the keyboard scan-code of the key that was pressed to be read from this port and raises an interrupt on the system bus. The line is kept high until the interrupt handler has read the scan-code from the I/O port.

- Status port 064H can be used to read the status byte of the PS/2 controller. It's bit definitions can be read in Table 16.
- Command port 064H is used to issue PS/2 controller specific command bytes to the PS/2 controller.

The Status byte defines its bits as follows:

Table 16: PS/2 Status byte bit definitions

Bit	Function
7	Parity Error Bit 1 = Parity error 0 = No error
6	Timeout Error Bit 1 = Timeout error 0 = No error
5	Vendor Specific
4	Vendor Specific
3	Command/Data Bit 1 = Data written to the data port is for the PS/2 Controller 0 = Data written to the data port is for a PS/2 device
2	System Flag
1	Input Buffer Status 1 = Input Buffer full 0 = Input Buffer empty
0	Output Buffer Status 1 = Output Buffer full 0 = Output Buffer empty

The system flag is used to identify if the system POST-ed correctly. This bit should be 0 in normal operation. Note that the Input and Output buffers are single byte buffers.

Communicating with the PS/2 Controller and PS/2 Devices

Under normal operation a programmer will not need to issue commands to the PS/2 controller or to another PS/2 device. An example case where a programmer might might need to issue commands to the PS/2 controller and devices is if the programmer wishes to add support for an additional PS/2 device that will be plugged into PS/2 port 2, such as a mouse.

If the programmer should need to communicate with the PS/2 controller, they do so by issuing the command to the PS/2 controller by writing the command to I/O port 064H. If the command is longer than one byte, then the programmer will need to send the first byte via I/O port 064H and all subsequent bytes must be sent via I/O port 060H. The programmer must only write a byte to port I/O 060H if the Input Buffer Status bit of the PS/2 status port is 0. Otherwise, the programmer must poll this bit until it is.

If the programmer needs to communicate with a PS/2 device, they can do so in a similar fashion to communicating with the PS/2 controller. To communicate with a device on the PS/2 port 1 (reserved for the system keyboard), the programmer must first poll to make sure that the Input Buffer Status bit of the PS/2 status port is 0. If it is, it is safe for the programmer to write the first byte of the command to I/O port 060H. If there are subsequent bytes to be sent to the device, the programmer must poll the Input Buffer Status bit of the PS/2 status port again, and only write the next byte once that bit is 0.

To communicate with a device on PS/2 port 2, the programmer must first issue the command byte 0D4H to the PS/2 controller, by writing it to I/O port 064H. Then, as normal, the programmer must poll the Input Buffer Status bit of the PS/2 status port until it is 0. Once it is, it is safe for the programmer to write the command byte that is to go to the device on PS/2 port 2, to I/O port 060H. If there are more bytes in the command, the programmer must poll Input Buffer Status bit of the PS/2 status port until it is 0, before writing the subsequent bytes to I/O port 060H. If the programmer wishes to send multiple commands to the PS/2 device on PS/2 port 2, they must always start by first issuing the command byte 0D4H to the PS/2 controller.

The PS/2 controller and PS/2 devices respond to many commands with at least one byte of data. In these cases, the programmer can either install an interrupt handler and let the PS/2 controller raise an interrupt when a response byte is ready to be read from I/O port 060H or poll the Output Buffer Status bit of the PS/2 status port until it is 1, indicating there is a response byte to be read from I/O port 060H.

Hardware interrupts associated with the PS/2 subsystem

The PS/2 subsystem has two hardware interrupts associated with it. IRQ 1 is the hardware interrupt level connected to PS/2 port 1, and is reserved for use by SCP/BIOS for the keyboard. When a key on the keyboard is struck, the keyboard controller places the scan-code on I/O port 060H and raises IRQ 1 high. The interrupt then remains high until the scan-code is read from I/O port 060H.

The second hardware interrupt level is IRQ 12. This hardware interrupt is free to be used, and may be used by devices on PS/2 port 2, and functions much like in the previous case, but is connected to PS/2 port 2.

The PS/2 subsystem during system initialisation

During the system initialisation procedure the PS/2 subsystem initialises the PS/2 controller and checks the device on PS/2 port 1. If this device is not present OR is not a keyboard, the system initialisation fails and the system will permanently halt. The screen will clear and a message will display saying “Keyboard Error. Halting...”. At this point, please power down your system, ensure your PS/2 keyboard is connected properly and restart your system.

The Programmable Interval Timer

The Programmable Interval Timer or PIT, is a device based on the Intel 8254 chip, which is used for system-wide timekeeping. It consists of an oscillator, oscillating at roughly 1.193182 MHz and three separate 16-bit frequency dividers, each with their own independent I/O channels, allowing for a programmer access to three system separate timers based on the PIT. These timers are called the Channel 0 timer, Channel 1 timer and Channel 2 timer. Each timer has multiple operating modes, and can be set to operate independently of one another.

Historically, in the PC architecture, channel 0 was used for timekeeping in the system, as it is the only one of the three timers which is connected to a hardware interrupt. It was hardwired to IRQ 0, the highest priority hardware interrupt on the system. Channel 1 was reserved on older systems and was used for DRAM refresh. A programmer could have changed the frequency of channel 1 however, doing so would have more than likely crashed the system. Channel 2 was left as a general purpose timer to be used by the programmer however they saw fit. It was also the only channel directly connected to the PC Speaker and could be used to modulate signals to generate audio at particular frequencies.

Under SCP/BIOS, channel 0 is used as the system timekeeping interrupt and it is set to tick at a frequency of 18.2Hz, or roughly every 55ms. The highest priority hardware interrupt level, IRQ 0, is connected to this timer and as such, the handler for IRQ 0 is entered once every 55ms (or roughly 18.2 times a second) unless reprogrammed. The SCP/BIOS handler for IRQ 0 also has a user hook interrupt, Int 3CH, that can be hooked by a systems programmer for whatever purpose they may need. This user hook interrupt is entered every time IRQ 0 is raised.

Due to historical reasons, channel 1 is not implemented on most system boards and thus a programmer *must not under any circumstances* attempt to access PIT Timer 1.

Channel 2 is left for the programmer to use however they wish. If your system board supports a PC speaker, then the programmer can use this timer to produce beeps with the PC speaker on the system board, as it is connected to the PC speaker.

Modes of Operation

The PIT has at most three channels. Each channel has a 16-bit internal counter, whose use is determined by the mode the channel is set to function in. The PIT can be fully programmed using four I/O ports. These four I/O ports are byte sized and are defined as follows:

Table 17: The PIT programming interface

I/O Addr	Port Name	R/W
040H	Channel 0 Data Port	R/W
041H	Channel 1 Data Port	R/W
042H	Channel 2 Data Port	R/W
043H	Command Register	WO

Each of the three data ports allows the programmer read/write access to a particular channel's internal counter. To use this programming interface a programmer must first send a command byte to the command register at I/O port 043H. Then, if the command necessitates more bytes (such as in the case of latching a new 16-bit value into a channels counter), the programmer sends bytes to the Data Port of the appropriate channel. Similarly, if the programmer wishes to read the divisor value at a particular moment in time, the programmer may do so by sending the appropriate command byte, and then reading either one or two bytes from the chosen channels' data port. The structure of the 8-bit command byte is as shown in Table 18.

The Select Counter bits can be used to select which PIT channel the programmer wishes to program. If the programmer wishes to read the settings of a current counter, they can do so by using the Read Back Mode. The operation of this command is outlined at the end of this section.

The Read/Write bits are used by a programmer to specify whether the programmer wishes to set that channels data port to be used to read or write the least significant byte of the chosen channels' counter, the most significant byte of the chosen channels' counter, or read or write the least significant byte and then the most significant byte of the chosen channels' counter.

Table 18: The structure of the PIT command byte

Bits	Function			
7-6	Select Counter Bits			
	-	0	0	Channel 0
	-	0	1	Channel 1
	-	1	0	Channel 2
	-	1	1	Read Back Command
5-4	Read/Write Bits			
	-	0	0	Counter Latch Command
	-	0	1	Access Mode: Low byte only
	-	1	0	Access Mode: Hi byte only
	-	1	1	Access Mode: Low/Hi byte
3-1	Mode Bits			
	0	0	0	Mode 0
	0	0	1	Mode 1
	x	1	0	Mode 2
	x	1	1	Mode 3
	1	0	0	Mode 4
	1	0	1	Mode 5
0	BCD/Binary Mode			

The Read/Write bits can also be used by the programmer to command the PIT to latch the counter for a channel at a specific moment in time by issuing the Counter Latch Command. The PIT continues to use that channels' counter as normal, but once the command is received by the PIT, the counter value is latched and system can then read the latched value from the data port when convenient for the system. Two bytes must be read by the system, from the selected channels data port, to get both the low and high bytes of the counter. If another Counter Latch Command is issued to the same channel before both bytes for that channel have been read, the second command is ignored.

In Table 18, the symbol x means "dont care", but should be set to zero.

The Mode bits specify the "counter mode", i.e. how the channel will use the counter and generate its' output signal. The full details of these modes and their specifications can be found in the Intel 8254 Datasheet, but briefly

- Mode 0 is the Interrupt on Terminal count. It can be used typically to count events.
- Mode 1 is the Hardware Re-Triggerable One-Shot.
- Mode 2 is the Rate Generator. This mode functions like a frequency divider and can be used for Real Time Clock like functionality, decrementing the counter by 1,on each oscillator tick (clock pulse). Once the counter falls to 1, the channels' output signal goes low and the counter is reloaded to its initial value. After one clock pulse, the channels' output signal goes high again. If a new counter value is loaded to a channel whilst in this mode, the new value is not loaded until the counter falls to 1. A counter value of 1 must not be loaded to a channel in this mode. Mode 2 is periodic and the described sequence of events continues indefinitely.
- Mode 3 is the Square Wave Mode. This mode is very similar to Mode 2 and is useful for baud rate generation as it outputs a square wave rather than a short burst signal as in mode 2. The square wave is generated by feeding the signal into an internal flip-flop. As in mode 2, after each oscillator tick (clock pulse), the counter is decremented by 1. When the counter reaches 1, the channels' output signal goes low, and the counter is reloaded after one clock pulse. However, now the signal remains low until the counter reaches 1 again, at which point the channels' output signal goes high, with the counter being reloaded again after one clock pulse. This process is repeated indefinitely as mode 3 is periodic.

This is the mode used by SCP/BIOS as IRQ 0 can be triggered to go off on the rising edge of the square wave. The divisor value used is 0H to give a divisor of 65536. A counter value of 1 must not be loaded to a channel in this mode. This mode can also be used to generate sound with the PC Speaker, since it generates a proper square wave.

- Mode 4 is the Software Triggered Strobe.
- Mode 5 is the (Retriggerable) Hardware Triggered Strobe.

If the programmer wishes to use mode 2 for a particular channel, they may compute the divisor value necessary to make the channel tick at a particular frequency using the following formula.

$$\text{Tick Frequency in Hz} = \frac{1193182\text{Hz}}{\text{Divisor}}$$

Finally, the BCD/Binary mode allows the programmer to set the channel to use the counter as either a two byte BCD value or a single 16-bit binary value.

Reading a channels' counter and state

If a programmer wishes to read the counter value for a particular channel, they have three options:

1. During the channel initialisation, the programmer can set up the command byte to set the channels access mode as desired. Then the programmer can read one or two bytes from that channels data port whenever desired by the programmer. This is the simplest way to read a channels count, but can lead to nonsensical results.
2. Issue a Counter Latch Command for a particular channel, whereby the programmer instructs the PIT to latch the counter value of the chosen counter at the moment the PIT receives the command. The programmer is then free to read the 16-bit value at their leisure, though they cannot issue another Counter Latch Command to that channel until they have read all the latched data from that channels data port. The programmer is free however to issue other commands to the PIT.
3. Issue a Read-Back command. This command is the most complex and returns the full state of the chosen channel(s) at the moment the Read-Back command is issued.

The Read-Back command can be used to get the state of up to all three of the channels at once. This saves the programmer from having to save a copy of the command bytes sent to each channel in system memory. The Read-Back command is structured slightly differently to the command byte though it is still written to I/O port 043H. The structure of the Read-Back command is as in Table 19.

Once the Read-Back command has been issued, the programmer can then read back the data they chose to receive from the PIT, by reading the data from each selected channels data port.

If the programmer chooses to latch the status of the selected channels, then the status byte they will read will be as follows:

1. Bit 7 will indicate the status of the physical OUT pin on that channel. A value of 1 means that OUT is high and a value of 0 means that OUT is low. The function of the OUT pin is described in the Intel 8254 datasheet.

2. Bit 6 will indicate NULL count. This bit indicates whether the last count written to the channel has been loaded to be used by the counter yet or not. If the counter has not started using the loaded value (such as in the event that the channel is waiting for an interrupt to begin counting down) then this bit will be set to 1. Otherwise this bit will be set to 0.

3. Bits 5-0 reflect bits 5-0 of the command byte that initialised that particular channel.

As with the Counter Latch Command, if multiple Read-Back commands are issued with data latched, then only the data from the first Read-Back command will be latched on those selected channels data ports. Both count and status of the selected counter(s) may be latched simultaneously and doing so is functionally the same as issuing two separate Read-Back commands at once, but latching different data and thus this is not a no-op.

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched counts.

Table 19: Read-back Command Byte structure

Bits	Function
7-6	Set to 11b
5	0 = Latch Count of selected channels. 1 = Do not latch.
4	0 = Latch Status of selected channels. 1 = Do not latch.
3	1 = Select Channel 2
2	1 = Select Channel 1
1	1 = Select Channel 0
0	Reserved, Must be set to 0

Finally, during system initialisation, SCP/BIOS only initialises Channel 0 for usage, leaving Channel 2 uninitialised for the programmer to use as they see fit. The setup used by SCP/BIOS for Channel 0 is:

- Access mode: Low/High byte
- Mode 3
- Counter value of 0 = 65536

This is done during system initialisation with an x86-64 assembly routine similar to the following:

```
mov al, 36h
out 43h, al
mov ax, 0
out 40h, al
mov al, ah
out 40h, al
```

Figure 1: Setting Channel 0 of the PIT as in SCP/BIOS in x86-64 Assembly

Programming the PC Speaker

If a PC Speaker is available on your system board, or through emulation, then the programmer can use Channel 2 to program the speaker at a specified frequency. To do so, the programmer should first configure Channel 2 to Mode 3 and set the frequency divisor for the note they wish to produce. The programmer can then logically OR the value of I/O register 61h with 3h, to produce the note. To stop the note from sounding, the programmer can then logically AND the value of I/O register 61h with FCh.

Real Time Clock facilities and the CMOS

SCP/BIOS also makes partial use of the CMOS subsystem and the built in Real Time Clock (RTC). The CMOS subsystem consists of two banks of 128 bytes of battery backed RAM that is built into the system board and historically was used to convey information to a programmer or application about the system it was working on. Many fields of the CMOS were left empty or undocumented and as such very few RAM locations contain any certain information. Of the few registers that are well known, the majority are related to timekeeping and the RTC subsystem. SCP/BIOS only makes use of the timekeeping functionality of the CMOS.

Programming with the CMOS

Access to the CMOS RAM is governed using two byte sized I/O registers:

- The CMOS Index register at I/O address 070H.
- The CMOS Data register at I/O address 071H.

To send or receive data to or from a CMOS memory location, called a CMOS register, the programmer must first write the offset into the CMOS memory to the CMOS Index register. Once that has been completed, it is recommended that the programmer allows some bus cycles to pass before reading or writing the data to or from the CMOS Data register. This is to allow the CMOS system to connect the selected byte of CMOS RAM to I/O address 071H. This can be quite a slow process so it is recommended the programmer does a dummy read or write from an unused I/O address, such as I/O address 80H (which is conventionally used as the Output port for diagnostic purposes), as this should give the CMOS subsystem enough time to process the request. After the dummy I/O read/write, the programmer is free to read or write the byte at I/O address 071H. The programmer should not wait too long writing to I/O address 070H and reading/writing from I/O address 071H as this can damage the CMOS subsystem. Therefore, the programmer should immediately read/write to address 071H after the dummy I/O read/write.

Note that all interrupts should be disabled when programming the CMOS. This includes all hardware interrupts and the NMI. The NMI is discussed in the next section.

The RTC is also capable of raising hardware interrupts. The RTC system is connected to hardware interrupt level IRQ 8 and can be programmed to trigger this interrupt either periodically (like the Channel 0 of the PIT is configured to do) or at a specific time, using the RTC alarm feature. The

SCP/BIOS interrupt handler for IRQ 8 also has a user hook interrupt, Int 6Ah, for the RTC Alarm feature only. This interrupt is entered when the reason for the interrupt firing is the RTC Alarm. This interrupt can be hooked by a programmer to do some event at a specific time-of-day, using the RTC Alarm functionality.

The RTC is also capable of raising hardware interrupts every time it updates its internal clock, however this functionality is not used by SCP/BIOS. A programmer looking to make use of this functionality must write their own interrupt handler to replace the SCP/BIOS IRQ 8 handler.

Non-Maskable Interrupts and the CMOS registers

A feature of I/O port 070H is that it is connected to the Non-Maskable Interrupt (NMI) subsystem of the computer and acts as a mask for the physical NMI line. If a 1 is written to this bit, then the NMI line is masked and the NMI system is effectively disabled. A 0 must then be written to this bit to unmask and enable the NMI system again. It is not recommended to leave the NMI system disabled for any lengthy period of time.

As mentioned, when programming the CMOS, all interrupt sources, including the NMI, should be disabled. However, once the programmer has completed programming the CMOS, they must remember to re-enable all hardware interrupts that were disabled, including the NMI. This is because if an NMI is triggered whilst programming the CMOS, it can leave the CMOS in an unusable state, damaging the CMOS subsystem.

It is recommended that when programming the CMOS, all CMOS register addresses are written to the CMOS Index register with bit 7 set to mask off the NMI line. Then, once the programmer has completed all CMOS related activity, they re-enable the NMI by writing a 0DH byte to the CMOS Index register, waiting one I/O cycle and then reading a byte from the CMOS Data register.

Warning!
Changing the values of any CMOS registers that are not listed in Table 20 as R/W may cause damage your system and/or may brick your system board. This is because the values in CMOS RAM are generally not re-initialised during POST. However they may be checksummed by the system ROM during POST. You may read (subject to the NMI rules outlined above) the value of any CMOS register however you must not attempt to change the values of any registers not explicitly marked as R/W in Table 20, unless you are absolutely sure you know what you are doing.

Table 20: CMOS RTC Register Locations

CMOS Register Number / Offset	CMOS Register Name	R/W
00H	RTC Seconds	R/W
01H	RTC Seconds Alarm	R/W
02H	RTC Minutes	R/W
03H	RTC Minutes Alarm	R/W
04H	RTC Hours	R/W
05H	RTC Hours Alarm	R/W
06H	RTC Day of the Week	R/W
07H	RTC Day of the Month	R/W
08H	RTC Month	R/W
09H	RTC Year	R/W
0AH	RTC Status Register A	R/W
0BH	RTC Status Register B	R/W
0CH	RTC Status Register C	RO
0DH	RTC Status Register D	RO

The Alarm registers contain the Hours/Minutes/Seconds that the RTC alarm will trigger on if it is configured to do so. The RTC also has four status registers, called Status Register A, B, C and D. These registers can be used to read the state of the RTC subsystem and program how the system uses the RTC functionality. These registers are defined and used as follows:

Table 21: Status Register A

Bit 7	Update in progress (UIP) - A 1 indicates that the RTC date and time registers are being updated. A 0 indicates that the RTC date and time registers are available to read.
Bits 6-4	22-Stage Divider (DV2 - DV0) - These three bits allow the programmer to choose which time-base frequency is being used. This will change the base frequency of the RTC and can impact timekeeping by the RTC. The system initialises this field to 010b which selects a 32.768kHz time base.
Bits 3-0	Rate Selection Bits (RS3 - RS0) - These four bits allow the programmer to select a divisor to generate a frequency divided output signal. The base frequency divided by this divisor is set by the 22-stage divider bits. The system initialises these bits to 0110, which selects a 1.024kHz square wave output frequency and and a 976.562 microsecond periodic interrupt rate.

Table 22: Status Register B

Bit 7	Set - A 0 updates the cycle normally by advancing the counts at one-per-second. A 1 aborts any update cycle in progress and the programmer can initialise the date/time-keeping registers without any further updates occuring until a 0 is written to this bit. The system initialises this bit to 0.
Bit 6	Periodic Interrupt Enable (PIE) - A 1 enables the Periodic interrupt to occur at a rate specified by the rate and divider bits in Status Register A. The system initialises this bit to 0.
Bit 5	Alarm Interrupt Enable (AIE) - A 1 enables the Alarm interrupt to trigger when the RTC clock reaches the time in the Hour/Minute/Seconds alarm registers. The system initialises this bit to 0.
Bit 4	Update Ended Interrupt Enabled (UEIE) - A 1 enables the RTC date/time register update interrupt and a 0 disables it. The system initialises this bit to 0. Note, this interrupt is not supported by the SCP/BIOS RTC Interrupt handler.
Bit 3	Square Wave Enabled (SQWE) - A 1 enables the square wave generator to generate a square wave at a frequency set by the Rate Selection Bits in Status Register A. The system initialises this bit to 0.
Bit 2	Date Mode (DM) - This bit indicates whether the date and time registers are to use binary or BCD formats. A 1 indicates binary and a 0 indicates BCD. The system initialises this bit to 0.
Bit 1	24/12 - This bit establishes whether the hours byte is in 24 hours or 12 hour mode. A 1 indicates 24 hour mode and a 0 indicates 12 hour mode. The system initialises this bit to 1.
Bit 0	Daylight Savings Enabled (DSE) - A 1 enables daylight savings and a 0 disables daylight savings. The system initialises this bit to 0.

Table 23: Status Register C

Bit 7	Interrupt Request Flag (IRQF) - A 1 means that the RTC Interrupt Request line is set. This flag is only affected if either AIE, PIE or UIE in Status Register B is 1.
Bit 6	Periodic Interrupt Flag (PF) - A 1 means that the Periodic Interrupt was triggered. This flag is only affected if PIE is set to 1.
Bit 5	Alarm Interrupt Flag (AF) - A 1 means that the Alarm Interrupt was triggered. This flag is only affected if AIE is set to 1.
Bit 4	Update Ended Interrupt Flag (UF) - A 1 means that the Update Ended interrupt was triggered. This flag is only affected if UEIE is set to 1.
Bits 3-0	Reserved, 0

Table 24: Status Register D

Bit 7	Valid RAM Bit (VRB) - A 1 means the RTC is powered. A 0 means the RTC has lost power.
Bits 6-0	Reserved, 0

At startup, SCP/BIOS initialises Status Registers A and B as follows:

- Status Register A - 026H
- Status Register B - 002H

RTC Interrupts

As we have seen, there are three types of RTC interrupt:

- Update-Ended
- Alarm
- Periodic

Assuming the 22-Stage Divider is kept as initialised by SCP/BIOS (and it is recommended to do so as changing the 22-Stage Divider will change the rate at which the RTC works, thus ruining its timekeeping functionality), then the following section will describe how to use each interrupt.

The Update-Ended Interrupt, Lowest Priority

The Update Ended interrupt is a simple interrupt that will trigger once every second as it only triggers once the internal clock has updated, which occurs once every second. The SCP/BIOS interrupt handler for the RTC will simply return out of the interrupt handler if this interrupt event is the only reason for the interrupt being triggered. If a programmer wishes to use this feature, they must replace the SCP/BIOS IRQ 8 handler with their own.

The Alarm Interrupt, Medium Priority

The Alarm interrupt allows a programmer to set an alarm to occur at a particular time of day. The time of day at which the alarm should occur can be selected by writing the values to the RTC Hours/Minutes/Seconds registers before enabling the alarm. Note that these registers take either BCD values or 8-bit binary values and must be in either 12 or 24 hour mode, depending on the state of Bits 1 and 2 of Status Register B. The system default is a 24 hour clock in BCD mode. When the Alarm is triggered, the interrupt handler will read Status Register C, check if there is a periodic interrupt, handle that first and then enter Interrupt 6AH.

The Periodic Interrupt, Highest Priority

The Periodic interrupt allows a programmer to have a second periodic timer in the system much like the PIT's channel 0. However, the user must be warned that this interrupt line is of a lower priority than IRQ lines 0-7 and thus, using this interrupt for accurate timekeeping might lead to drifting due to each higher priority IRQ being serviced before the RTC thus potentially preventing the IRQ 8 interrupt handler from running promptly. The interrupt handler for this simply decrements a global 64-bit count. The Periodic Interrupt does not automatically turn off once the count reaches zero and simply overflows. The frequency at which the periodic interrupt is generated can be set by the Rate Selection Bits of Status Register A. Table 25 demonstrates all possible frequencies the RTC can trigger the periodic interrupt at, assuming the 22-Stage Divider is left untouched.

Finally, when using the interrupt functions of the RTC, the interrupt handler must additionally read RTC Status Register C, to indicate to the RTC that the interrupt has been handled. No further RTC interrupts will occur until Status Register C is read.

RS bit				Frequency	Period
3	2	1	0		
0	0	0	0	-	-
0	0	0	1	256 Hz	3.90625 ms
0	0	1	0	128 Hz	7.8125 ms
0	0	1	1	8192 Hz	122.070 μs
0	1	0	0	4092 Hz	244.141 μs
0	1	0	1	2048 Hz	488.281 μs
0	1	1	0	1024 Hz	976.562 μs
0	1	1	1	512 Hz	1.93125 ms
1	0	0	0	256 Hz	3.90625 ms
1	0	0	1	128 Hz	7.8125 ms
1	0	1	0	64 Hz	15.625 ms
1	0	1	1	32 Hz	31.25 ms
1	1	0	0	16 Hz	62.50 ms
1	1	0	1	8 Hz	125.0 ms
1	1	1	0	4 Hz	250.0 ms
1	1	1	1	2 Hz	500.0 ms

Table 25: All standard frequency divisors for the RTC

Section 2: BIOS and its usage

Introduction to the BIOS

The SCP Basic Input/Output System (SCP/BIOS) is a software layer, designed to simplify writing low-level 64-bit applications for x86-64 based systems. The function of SCP/BIOS is to abstract the system hardware from the programmer, but to not prevent the programmer from accessing the system hardware directly, should the need or want arise. The routines provided by SCP/BIOS are such that they can be used by both assembly and high-level language programmers. A programmer can use SCP/BIOS routines to make character level or block level requests to devices without specific concern about the hardware specifics, such as hardware addresses or specific timings. SCP/BIOS also provides simple system management services such as a hardware memory map, system timing and sleep functionality and a simplified time-of-day interface.

Operating systems and programs that wish to make use of SCP/BIOS should however make requests to SCP/BIOS rather that attempting to program hardware directly, as doing so may cause issues in code portability. Using SCP/BIOS removes the need for programmers to consider issues such as timings and hardware configurations and increases future code portability.

SCP/BIOS has been written in such a way that ensures maximum compatibility with application programs written for IBM compatible BIOSes. As such, most such applications that were written to solely use an IBM compatible BIOS should be very easy to port to use SCP/BIOS, subject to their design. However, there are fundamental architectural differences at the hardware level that prevent 100% compatibility and as such, total compatibility was not always possible to maintain. A programmer who is comfortable with writing programs for IBM compatible BIOSes should be extra aware of these particular differences. This section allows a new and experienced programmer to understand the structure and function of SCP/BIOS and how to use the functions provided by SCP/BIOS.

The SCP/BIOS Boot Specification and System Initialisation Procedure

Successfully using SCP/BIOS as part of your bootable application or operating system requires the use of a SCP/BIOS compatible bootloader, which adheres to the SCP/BIOS boot specification, outlined herein. In the language of SCP/BIOS, a sector is equivalent to one logical block of data on a block storage device.

A SCP/BIOS compatible bootloader may be used by any operating system or bootable application program to load SCP/BIOS before loading the rest of the operating system or bootable application. The bootloader may be used in conjunction with any file system or file systems that may be used by an operating system or bootable application.

SCP/BIOS Boot Specification

An SCP/BIOS compatible bootloader must provide the following information to SCP/BIOS, load SCP/BIOS as described below and behave in a well defined manner, before transferring system control to SCP/BIOS:

- The bootloader should remain in real-mode and load SCP/BIOS to the contiguous memory space beginning at address 0000H:0800H. Going to another CPU mode is permitted though any data structures that may be created as a result of having gone to another CPU mode may be destroyed by SCP/BIOS.
- The bootloader is responsible for loading all sectors containing SCP/BIOS from the block storage device which contains SCP/BIOS. This allows an operating system which wishes to fragment SCP/BIOS to do so without providing SCP/BIOS information about the file system structure. This, however, is not recommended.
- The bootloader is responsible for constructing a structure called the System Initialisation (SysInit) Parameter Table at any valid address that will not be occupied by SCP/BIOS, which will be used by SCP/BIOS during the system initialisation phase and to transfer control to the correct program once system initialisation has completed. The Boot Parameter Table pointer is passed to SCP/BIOS in ES:BX.
- The bootloader must transfer control to SCP/BIOS in real-mode by

doing either a near or far jump to address 0000H:0800H, ensuring that the CS register is set to 0.

- The bootloader must preserve the integrity of the IVT and the firmware BIOS data area. Therefore the address range 0000H:0000H - 0000H:0600H is to be considered as reserved by the bootloader, and must not be modified in any way. The Extended BIOS Data Area must also be preserved, though this area isn't uniform across different hardware configurations and so it is recommended the memory above 9000H:0000H is also left preserved by bootloaders.
- The bootloader should not store any data in memory with the aim of it being used by a future program. If an operating system or bootable application program wishes to use SCP/BIOS, it should be written such that the bootloader is used to load SCP/BIOS only. SCP/BIOS considers all memory in the system to be under its control during system initialisation and therefore any data written by a bootloader may be overwritten by SCP/BIOS without warning.

The SysInit Parameter Table is used by the operating system or bootable program which uses SCP/BIOS to describe to SCP/BIOS where the next sectors of the operating system/bootable program can be found and how many sectors to copy into memory, up to a maximum of 42 sectors. Any larger values get ignored by SCP/BIOS and is assumed to be an alias for 42 sectors. These sectors are copied to address 7C00H and the contiguous memory space thereafter. This copy count must be a count of contiguous sectors. If the next file to be loaded is fragmented, then the programmer using SCP/BIOS should use this table to point to the maximum number of non-fragmented sectors of data and and that program should be able to load the rest of the file thereafter. The structure of the SysInit Parameter Table is as follows:

1 Byte	- Length of the SysInit Parameter Table (0CH)
1 Byte	- Number of Contiguous Blocks to copy
2 Bytes	- Reserved, must be set to 0
8 Bytes	- First Logical Block of Next File

Table 26: Structure of the SysInit Parameter Table

SCP/BIOS requires that the first sector pointed to by the SysInit Parameter Table has the signature 055H, 0AAH at the beginning of the first sector. If SCP/BIOS detects this signature in the loaded sector, then SCP/BIOS will transfer control to address 7C02H. If this signature is not detected, the boot process will fail and the computer will load SYSDEBUG, the system debugger, to allow a programmer to try and load the correct sectors from the correct device and continue the boot process manually. Note that if your computer has multiple SCP/BIOS bootable media inserted, SCP/BIOS cannot guarantee that the same bootable device will be loaded from and thus the boot process may fail. Therefore, it is recommended that only one SCP/BIOS bootable device is inserted during the boot process.

The values provided to SCP/BIOS in the SysInit Parameter Table are saved by SCP/BIOS in the SCP/BIOS data area and can be used in INT 39H.

An example bootloader which can be used to load SCP/BIOS is provided in Appendix A after the SCP/BIOS listing. This bootloader defines a simple FAT file system on a 1.44Mb medium (though the file system can be overlaid on a logical block device of higher capacity). This bootloader loads the SysInit Parameter Table at the recommended address of 0000H:0800H.

System Initialisation Procedure

Once the bootloader has loaded SCP/BIOS into memory, SCP/BIOS will first proceed to set the CPU A20 line. SCP/BIOS then builds a “low data table” where information about the system is read from the system firmware BIOS and arranged into a data table in the conventional memory arena, after which SCP/BIOS will reset the screen mode, set up a memory mapped page table, set up a simple GDT and an empty IDT and will go straight to long mode.

Once in long mode, SCP/BIOS then initialises its own internal data areas and copies the resident portion of SCP/BIOS to its correct location in system memory and adds an entry for SCP/BIOS into the system memory map. It then creates a new GDT, new identity paging tables for the first 4GB of memory and sets up the interrupt descriptors for the SCP/BIOS interrupts in a new IDT, after which it begins using all three of these new structures.

Device discovery and initialisation can then begin. First the system programmable interrupt controllers are reinitialised and remapped to allow hardware interrupts to trigger the correct interrupt handlers. Then SCP/BIOS scans the PCI bus for any type of USB controller (UHCI, OHCI, EHCI and xHCI), to then initiate the OS/BIOS handover, and terminate any USB legacy support emulation that may be running on the system (this is often buggy and not guaranteed to work in long mode). The enumeration process makes note of EHCI and xHCI controllers. The procedure also continues to search the PCI address space in search of ATA controllers. All ATA controllers are registered and those controllers that support being reconfigured into IDE compatibility mode are reconfigured so.

SCP/BIOS then returns to configure the system timers. The Programmable Interval Timer is reinitialised and configured to tick at the correct frequency. The Real-Time Clock is also reconfigured and setup as outlined in the hardware section of this manual. The system then unmasks interrupt lines for the PIT and RTC IRQs to allow them to begin operating normally. SCP/BIOS then detects the system's asynchronous serial communication ports and initialises them to operate at 2400 baud with 8 bit words, 1 stop bit and no parity.

SCP/BIOS now configures the system PS/2 keyboard, by resetting it and setting the correct scan-code set, either through native support or using keyboard controller level translation, and ensures the device is working properly. On some machines, the user may be prompted to strike a key at this stage to proceed. Once this is complete, SCP/BIOS then will attempt to communicate with system hard drives. This feature is not yet complete and will be implemented in the next release of SCP/BIOS.

Finally SCP/BIOS will initialise the USB EHCI for each EHCI controller. SCP/BIOS supports up to four separate EHCI controllers. SCP/BIOS will enumerate all supported USB devices on the EHCI controllers and will initialise them one by one. SCP/BIOS will then enter the Bootloader procedure and copy the sector(s) of data that were specified in the SysInit Parameter Table into memory at address 7C00H, and control is then transferred to the loaded program.

SCP/BIOS Memory Map and Memory Paging

SCP/BIOS occupies a data segment starting 16 bytes after the High Memory Area (HMA), at address 110000H. SCP/BIOS also reserves the 16 bytes between HMA and the start of the SCP/BIOS segment and this region must not be used.

The size of the SCP/BIOS segment is variable as the amount of memory SCP/BIOS allocates for itself during system initialisation depends on the hardware present on the machine and how much information SCP/BIOS can get about the hardware it is running on from the system BIOS before SCP/BIOS completes its initialisation. The programmer, however, can get a pointer to the first KB after the end of SCP/BIOS and use that information, along with the system memory map, to begin memory allocation.

An operating system or applications programmer should get and parse the full system memory map from SCP/BIOS as soon as they can as various machine configurations have various memory “holes”; regions reserved for hardware memory mapped I/O. The following table gives a rough outline of what the memory map looks like after SCP/BIOS system initialisation has completed. Note, in the following table, the regions marked as reserved must not be used under any circumstances, even if the system memory map of your system marks those regions as clear.

SCP/BIOS broadly divides system memory into four “arenas”.

1. The Conventional Memory Arena, 00000000H - 000FFFFFH
2. The Low Extended Memory Arena, 00100000H - 00EFFFFFH
3. The High Extended Memory Arena, 00F00000H - 0FFFFFFFFH
4. The Long Memory Arena, 100000000H - END_OF_MEMORY

The conventional memory arena is the entire memory space that is accessible with 20-bit addressing. This is the space that is accessible to the CPU in all operating modes. This mode is reserved for future use by SCP/BIOS and must be left complete preserved by the programmer, with the exception of the region between 7C00H and D000H which is used for loading the operating system or bootable program and the space between D000H and DFFFH which is given to the operating system or bootable program by SCP/BIOS to use as a 512 qword stack. The space between 7C00H and DFFFH is called the bootstrap region, and should only be used during the initial bootstrapping phase of a program running under SCP/BIOS. SCP/BIOS will copy the specified number of sectors from a valid logical block device to address 7C00H, after which the program is expected to either relocate itself and the stack pointer to past the end of SCP/BIOS or read more sectors from Logical Block Devices past the end of SCP/BIOS and transfer control there and move the stack pointer. When control is transferred to a loaded programs’ boot sector by SCP/BIOS, a pointer to the address past the bottom of SCP/BIOS, called USER_BASE, is passed to the application in the RBX register. Therefore, the area between 7C00H and DFFFH should only be used by an application during its initial bootstrap phase. The space between addresses 000A0000H and 00FFFFFH is called the “Conventional Memory Arena Hole”, and is where some devices map their registers and/or memory for MMIO. These regions may be used by an applications programmer to communicate with devices.

The Low Extended Memory arena is the memory space starting at address 00100000H and ending at address 00EFFFFFH. This space is accessible by the CPU in both protected and long modes, provided there is memory and/or device to populate this memory space. This is also the arena where SCP/BIOS resides. The space from the start of the arena to the end of SCP/BIOS is reserved by SCP/BIOS and may not be used. The arena is ended by the "Low Extended Memory hole", another MMIO mapping area where some devices may map their registers and/or memory for MMIO. These regions may be used by an applications programmer to communicate with devices.

The High Extended Memory arena is the memory space starting at address 00F00000H and ending at address 0FFFFFFFFH. This space is accessible by the CPU in both protected and long modes, provided there is memory and/or device to populate this memory space. This space may be used from the start address of the arena up to the "High Extended Memory hole" which ends the High Extended Memory arena. This memory hole is used for devices mapping their registers/memory for MMIO. This memory hole may be used by an applications programmer to communicate with these devices. /par

The Long Memory arena is the memory space starting at address 100000000H and ending at the END_OF_MEMORY mark. This space is accessible by the CPU only in long mode (and a part of it may be accessible by a CPU in protected mode with Physical Address Extensions enabled). Depending on the system configuration, this region may contain the largest contiguous chunk of free memory.

These descriptions form a broad generalisation and are not a definitive guide to the system memory map. A systems programmer must always attempt to parse the system memory map, or at least get one of the system memory size determination counts before making assumptions about availability of system memory or it's memory layout. SCP/BIOS guarantees that the space from USER_BASE to at least address 00200000H is free and available for use by a programmer, so as to give a programmer a region of guaranteed availability during the programs initialisation. Therefore, if a programmer chooses not to call any memory size determination functions or does not attempt to parse the system memory map, then they must assume the system their program is running on has at most 2MB of memory and may only use the space from USER_BASE.

If your system has less than 4GB of system memory, than your system will have no "Long memory".

Paging under SCP/BIOS

The philosophy of SCP/BIOS is such that the hardware of the system is presented to a systems programmer "as is". Therefore, SCP/BIOS identity maps the first 4GB of system memory for use by a systems programmer. For this revision of SCP/BIOS, the Long Memory arena remains unmapped by default and thus it remains inaccessible to an applications programmer. Attempting to access this memory will trigger an exception. If an applications programmer wishes to use this memory and continue using SCP/BIOS, they must create a new page table which identity maps all the memory they from address 0 to the memory address they wish to end at. This limitation will be removed in future versions. The programmer is permitted to set up this extended identity paging scheme by copying the SCP/BIOS page tables from their location in the SCP/BIOS system data table area. In the future, applications programmers will be permitted to set up their own non-identity paging scheme.

Interrupts

SCP/BIOS functions are accessed using the system software interrupts. Each SCP/BIOS function is accessed via its own interrupt handler with the subfunction being passed to the handler in the AH register. In some cases, such at the INT 35H dispatch functions, a further value is placed in the AL register to specify which dispatcher function needs to be called.

Software interrupts 30H to 3FH are used to access various BIOS routines. For example, INT 36H manages the system keyboard, and INT 30H manages the system video output device.

Parameter Passing

All registers that are not used to pass parameters to the SCP/BIOS functions are preserved. Registers that are used to pass parameters, may be preserved, unless they are used to return values from the function. Most SCP/BIOS functions pass values to the interrupts in several registers and the function responds by placing values in certain registers in response.

If a function presents many subfunctions then the subfunction value is specified in AH. For example, to read a number of logical blocks (sectors) from a block device into to a memory buffer, the following code may be used:

```
mov ah, 82h            ;Read sectors into memory using LBA
mov al, NUM_SECTORS    ;Establish the number of sectors to be read
mov rbx, BUF_PTR       ;Establish the buffer address
mov rcx, STRT_LBA      ;Establish the starting block to read from
mov dl, DEV_NUM        ;Establish which Logical Block Device to use
int 33h                ;
jnc NO_ERROR           ;If an error occurred the carry flag will be set
cmp ah, ERROR_MASK     ;On return, AH contains the error code
```

To await a keypress and write it on screen using the teletype interface, a programmer may use the following code:

```
mov ah, 00h     ;Await a keypress function
int 36h         ;INT36h returns the ASCII code in AL
                ; and the keyboard scan-code in AH
mov ah, 0Eh     ;Select the write ASCII code in AL to TTY function
int 30h         ;Call the video display routine
```

The register AH is usually used to pass back the main return information, or a return code. The Carry Flag is frequently also used to indicate an error has occurred. When calling a function where the Carry Flag is used to indicate an error, please clear the carry flag before entering the function using the CPU instruction CLC.

If a programmer attempts to use a function number that doesn't exist, then, upon returning from the interrupt the Carry Flag will be set and the register AH will contain the value 80H for “function doesn't exist” or 86H for “reserved function not yet implemented”, depending on which subfunction of which interrupt routine was called.

Warning! SCP/BIOS routines may modify any register used to pass arguments to an SCP/BIOS function, even if it is not a register in which a value will be returned in. After a function call, the state of all registers in which arguments were passed should be considered as undefined and should be reloaded with data for subsequent function calls if need be.

Guide to BIOS Interrupts

The interrupts which provide an application interface to a systems programmer are outlined in the table below:

Interrupts 00H - 1FH, Reserved CPU Exceptions

These interrupts are described in detail in Appendix F. Should one of these interrupts occur, the user will be presented with a Blue screen with an error code and the user will be given three options. If the user continues to receive blue screens thereafter, they should reboot the system.

Of note however is Interrupt 02H, which is connected to the Non-Maskable Interrupt (NMI) line. If a Non-Maskable Interrupt should be raised, the programmer should shut down the system as soon as possible as a Non-Maskable interrupt usually signals critical failure of some aspect of the hardware, which could permanently damage your system.

Interrupts 20H - 2FH, Hardware Interrupts and IRQs

These interrupts are reserved for hardware interrupt handlers. Each of the 16 system IRQ levels are connected to one interrupt vector. The interrupt vectors, hardware IRQ level and the device the IRQ may be reserved for, are demonstrated in Table 27.

Interrupt Vector	IRQ Level	Device/Status
Interrupt 20H	IRQ 0	PIT Channel 0
Interrupt 21H	IRQ 1	PS/2 Port 1 (Keyboard)
Interrupt 22H	IRQ 2	Reserved
Interrupt 23H	IRQ 3	Serial Ports 2 and 4
Interrupt 24H	IRQ 4	Serial Ports 1 and 3
Interrupt 25H	IRQ 5	Free
Interrupt 26H	IRQ 6	Reserved
Interrupt 27H	IRQ 7	Free
Interrupt 28H	IRQ 8	RTC
Interrupt 29H	IRQ 9	Reserved
Interrupt 2AH	IRQ 10	Free
Interrupt 2BH	IRQ 11	Free
Interrupt 2CH	IRQ 12	Free
Interrupt 2DH	IRQ 13	Free
Interrupt 2EH	IRQ 14	Reserved
Interrupt 2FH	IRQ 15	Free

Table 27: IRQs, their associated Interrupt vectors and their reserved status

IRQ levels 0 and 1 are reserved by the PIT and the Keyboard respectively and should not be re-purposed. All other may be used by any hardware device, and may sometimes be shared by multiple devices. IRQ levels marked as reserved *should* not be used by any device. A reserved status means that that IRQ is envisaged as being used in a future version of SCP/BIOS for supporting additional hardware.

SCP/BIOS recommends that devices which require the use of an IRQ should be built with IRQ sharing in mind. Interrupt handlers too should be written with IRQ sharing in mind.

For IRQ sharing to work properly, device interrupt handlers must be written in such a way that each interrupt handler must save pointer to the current interrupt handler before installing itself. Each interrupt handler should have a mechanism to detect if the device it was written to manage is the device which triggered the interrupt. If it was, the handler should handle the interrupt as usual and exit. If the device is not the device which triggered the interrupt, the handler should just exit.

When the interrupt handler is exiting, rather than executing an IRETQ instruction, the new interrupt handler should do a jump to the previous interrupt handler. SCP/BIOS installs default interrupt handlers for each device which send an End of Interrupt signal to the appropriate PICs on the system board. Therefore, an interrupt handler that has been written to support IRQ sharing should not issue an End of Interrupt signal to either PIC.

Finally, SCP/BIOS will be moving onto using APIC hardware in future releases. Therefore any program which utilises SCP/BIOS and installs hardware interrupt handlers should be written with this in mind.

Interrupt 20H (IRQ0) - System Timer Interrupt Handler

SCP/BIOS enters this interrupt on every rising edge of the output line of Channel 0 of the PIT. That amounts to once every 55ms, or 18.2 times a second if Channel 0 is left with its default divisor. This interrupt increments a timer tick qword in the SCP/BIOS data area and then calls interrupt 3CH.

Interrupt 21H (IRQ1) - Keyboard Interrupt Handler

SCP/BIOS enters this interrupt on every keystroke entered by a user. This handler will use the scan-code and the current shift state of the keyboard (whether any modifier keys are being held) to then look up the appropriate ASCII code to place in the keyboard buffer. Every scan-code sent will update the state of the keyboard data, including the set/reset status of each of the modifier keys.

Certain key combinations will result in no data being put into the keyboard buffer. These combinations are reserved for special purposes. These include but are not limited to:

- Ctrl+Alt+Del - For rebooting the system
- Ctrl+BREAK - For triggering the Keyboard break interrupt

In the event of a Ctrl+Alt+Del key combination, the systems keyboard controller will pulse the CPU's RESET line, thus resetting the CPU. This is akin to doing a full system restart and doing so will force the system to reboot. The behaviour of the system on CPU reset is beyond the scope of this document. However, on some machines, the keyboard controller is not connected to the CPU's RESET line. In these machines, if the user presses Ctrl+Alt+Delete, the keyboard interrupt handler will ignore the keypress and a delete scan-code will not be saved in the buffer.

Interrupts 30H - FFH, Software Interrupts

The following section is a complete summary of all SCP/BIOS functions and how to call them. A user application can use these functions to carry out their processing in a manner that abstracts the underlying hardware from their program.

INT 30H	- Video Services
INT 31H	- Equipment Determination Service
INT 32H	- Conventional Memory Size Determination Service
INT 33H	- Block Storage Services
INT 34H	- Asynchronous Communications Services
INT 35H	- System Services
INT 36H	- Keyboard Services
INT 37H	- SCP/BIOS Reserved
INT 38H	- SYSDEBUG
INT 39H	- Warm Reboot Service
INT 3AH	- System-Timer and Real-Time Clock Services
INT 3BH	- Keyboard Break Handler
INT 3CH	- System Clock Handler
INT 3DH	- Screen Mode Parameters
INT 3EH	- Block Storage Device Parameters
INT 3FH	- Reserved, Video Extension Parameters
INT 6AH	- RTC Alarm Handler

Table 28: SCP/BIOS Reserved Software Interrupts

Interrupts 3DH, 3DH and 3FH are to be considered reserved by a programmer and should only be used in such a way that is implementation independent. This is because their implementation in this version of SCP/BIOS is under consideration. It is expected that by the next version, any issues will have been resolved.

Interrupt 30H - Video Services

The following is a summary of the SCP/BIOS video services:

(AH) = 00H	- Reserved, Set Screen Mode
(AH) = 01H	- Set Cursor Type
(AH) = 02H	- Set Cursor Position
(AH) = 03H	- Read Cursor Position
(AH) = 04H	- Write Byte to Active Page
(AH) = 05H	- Select Active Display Page
(AH) = 06H	- Scroll Active Page Up
(AH) = 07H	- Scroll Active Page Down
(AH) = 08H	- Read Attribute/Character at Current Cursor Position
(AH) = 09H	- Write Attribute/Character at Current Cursor Position
(AH) = 0AH	- Write Character at Current Cursor Position
(AH) = 0BH	- Reserved, Set Colour Palette
(AH) = 0CH	- Reserved, Write Dot
(AH) = 0DH	- Reserved, Read Dot
(AH) = 0EH	- Write Teletype to Active Page
(AH) = 0FH	- Read Current Video State
(AH) = 10H	- Reserved, Set Palette Registers
(AH) = 11H	- Reserved
(AH) = 12H	- Reserved
(AH) = 13H	- Write String
(AH) = 14H to FFH	- Reserved

Table 29: INT 30H - Video Functions

A programmer using INT 30H should not hard code any information about the screen geometry. They should instead call INT 30H, (AH)=0FH to get the current screen mode (which at this time is only VGA mode 3), the current active page and the number of columns on screen. The programmer should also know that mode 3 has 25 rows of character cells. The programmer should then save this information to variables to use in their application.

(AH) = 00H - Reserved, Set Screen Mode

This function is reserved and will be used to set the various screen modes, once additional screen modes are implemented.

(AH) = 01H - Set Cursor Type

(CH)	- Bits 4-0, Top line for cursor. Bits 7-5 Reserved.
(CL)	- Bits 4-0, Bottom line for cursor. Bits 7-5 Reserved.

Notes

1. The BIOS maintains one cursor type for all pages.
2. Setting reserved bits may cause erratic blinking or disable the cursor. Must be set to zero.

(AH) = 02H - Set Cursor Position

(DH, DL)	- Row, Column (0,0 is upper left)
(BH)	- Page Number (0-based)

Notes

- Each page has an independent cursor, and this function can be used to set cursor positions for all pages, and not just for the active page.

(AH) = 03H - Read Cursor Position

(BH)	- Page Number (0-based)

On Return:

(DH, DL)	- Row, Column (0,0 is upper left)
(CH, CL)	- Cursor type currently set

(AH) = 04H - Write Byte to Active Page

(AL)	- 8-bit value to write to active page current cursor position

Notes

- This function can be used to write an 8 bit value as ASCII characters. For example, if AL = 3CH then the ASCII characters 3 and C will be written to the current cursor position of the current active page.

(AH) = 05H - Select Active Display Page

(AL)	- New Page Number (0-based)

(AH) = 06H - Scroll Active Page Up

(AL)	- Number of lines blanked at bottom of window
	= 00H - Blank entire window
(CH, CL)	- Row, Column of upper left corner of scroll
(DH, DL)	- Row, Column of lower right corner of scroll
(BH)	- Attribute to use on blank line

(AH) = 07H - Scroll Active Page Down

(AL)	- Number of lines blanked at top of window
	= 00H - Blank entire window
(CH, CL)	- Row, Column of upper left corner of scroll
(DH, DL)	- Row, Column of lower right corner of scroll
(BH)	- Attribute to use on blank line

(AH) = 08H - Read Attribute/Character at Current Cursor Position

(BH)	- Page Number (0-based)

On Return:

(AL)	- Character read
(AH)	- Attribute of character read

(AH) = 09H - Write Attribute/Character at Current Cursor Position

(BH)	- Page number (0-based)
(CX)	- Count of characters to write
(AL)	- Character to write
(BL)	- Attribute of character

(AH) = 0AH - Write Character at Current Cursor Position

(BH)	- Page number (0-based)
(CX)	- Count of characters to write
(AL)	- Character to write

(AH) = 0BH - Reserved, Set Colour Palette

This function is reserved and will be used to set the colour palette colours of the VGA.

(AH) = 0CH - Reserved, Write Dot

This function is reserved and will be used to draw dots on the screen when in a graphical, All Pixels Addressible (APA) mode.

(AH) = 0DH - Reserved, Read Dot

This function is reserved and will be used to draw dots on the screen when in a graphical, All Pixels Addressible (APA) mode.

(AH) = 0EH - Write Teletype to Active Page

(AL)	- Character to write

(AH) = 0FH - Read Current Video State

On Return:

(AL)	- Screen mode currently set (at this time, this will return 3)
(AH)	- Number of character columns on screen
(BH)	- Current active page number (0-based)

(AH) = 10H - Reserved, Set Palette Registers

This function is reserved and will be used to set the VGA palette register values.

(AH) = 11H - Reserved

This function is reserved.

(AH) = 12H - Reserved

This function is reserved.

(AH) = 13H - Write String

(AL) = 00H - Write String with Attribute, Cursor not moved

- (RBP) - Pointer to string to write
- (CX) - Count of chars to print
- (DH, DL) - Row, Column to write string at (0,0 is upper left)
- (BH) - Page Number
- (BL) - Attribute

(AL) = 01H - Write String with Attribute, Cursor moved

- (RBP) - Pointer to string to write
- (CX) - Count of chars to print
- (DH, DL) - Row, Column to write string at (0,0 is upper left)
- (BH) - Page Number
- (BL) - Attribute

(AL) = 02H - Write String without Attribute, Cursor not moved

- (RBP) - Pointer to string to write
- (CX) - Count of chars to print
- (DH, DL) - Row, Column to write string at (0,0 is upper left)
- (BH) - Page Number

(AL) = 03H - Write String without Attribute, Cursor moved

- (RBP) - Pointer to string to write
- (CX) - Count of chars to print
- (DH, DL) - Row, Column to write string at (0,0 is upper left)
- (BH) - Page Number

(AL) = 04H - Write zero-terminated String in Teletype Mode to Active Page

- (RBP) - Pointer to zero-terminated string to write

Notes

- Subfunctions (AL) = 02H and 03H will use the attributes already stored in the video buffer for those character cells whose characters are being replaced by the string data.

Interrupt 31H - Equipment Determination

This routine returns data about the system configuration to the caller. Any field marked as reserved must not contain data that needs to be preserved by the function call, as these are reserved for use by SCP/BIOS. Not saving these fields before the function call may result in incompatibility of your software with future versions of SCP/BIOS. Note that the legacy bitfield returned may be changed by SCP/BIOS in a future update and should not be depended upon by a systems programmer.

On Return:

(AX)	- Reserved, Legacy Bitfield	
(R8)	- Configuration qword 1	
	(Byte 0)	- Number of Int 33H visible devices
	(Byte 1)	- Number of USB MSD devices on a EHCI bus
	(Byte 2)	- Reserved
	(Byte 3)	- Number of detected Asynchronous Serial Adapters
	(Bytes 4-7)	- Reserved
(R9)	- Reserved	
(R10)	- Reserved	
(R11)	- Reserved	
(R12)	- Reserved	
(R13)	- Reserved	
(R14)	- Reserved	
(R15)	- Reserved	

Interrupt 32H - Conventional Memory Size Determination

This routine returns to the caller the number of free contiguous 1KB blocks in the reserved conventional memory area.

On Return:	
(AX)	- Number of free contiguous 1KB blocks of conventional memory
(R8)	- Reserved
(R9)	- Reserved
(R10)	- Reserved

Notes

- The fields marked as reserved must be saved before calling this function as these fields may be used in future versions of SCP/BIOS.

Interrupt 33H - Block Storage Services

The following is a summary of the SCP/BIOS Block Storage device related services:

(AH) = 00H	- Reset Media System
(AH) = 01H	- Read Status of Last Operation
(AH) = 02H	- Read Desired Sectors into Memory (CHS)
(AH) = 03H	- Write Desired Sectors from Memory (CHS)
(AH) = 04H	- Verify Desired Sectors (CHS)
(AH) = 05H	- Format Desired Track (CHS)
(AH) = xxH	- Reserved for 05 < xxH < 82H
(AH) = 82H	- Read Desired Sectors into Memory (LBA)
(AH) = 83H	- Write Desired Sectors from Memory (LBA)
(AH) = 84H	- Verify Sectors (LBA)
(AH) = 85H	- Format Desired Sectors (LBA)
(AH) = 86H	- Reserved
(AH) = 87H	- Reserved
(AH) = 88H	- Read Device Parameters (LBA)

Table 30: INT 33H - Block Storage Functions

SCP/BIOS has support for Block Storage devices such as USB Bulk-Only Mass Storage Devices. Future supported devices will include Floppy and Hard Disk Drives, SCSI drives and other subclasses of USB Mass Storage Devices.

At the time of writing, Hard Disk drives are not supported, however a fixed CHS addressing scheme has been implemented for both Removable Media and Fixed Media for future proofing. CHS functions for removable media can only access the first 1.44Mb of a Removable Medium, using a disk geometry of 80 Cylinders, 2 Heads and 9 Sectors per Track, with a sector size of 512 bytes.

(AH) = 00H - Reset Media System

(DL) - Device Number (0-based)
Bit 7 = 0 - Removable Medium/Device
Bit 7 = 1 - Fixed Disk
On Return:
CF = CY - Status of reset is non-zero
CF = NC - Status of reset is zero

(AH) - Status of Operation
= FFH - Sense Operation Failed
= BBH - Undefined Error
= 80H - Timeout, Device not ready
= 40H - Seek Failure
= 21H - Device/Controller Stall Error
= 20H - Controller Error (EHCI for EHCI devices)
= 10H - ECC/CRC error on device read
= 06H - Media changed or removed
= 05H - Reset failed
= 04H - Requested sector not found
= 01H - Invalid diskette parameter
= 00H - No error

(AH) = 01H - Read Status of Last Operation

(DL) - Device Number (0-based)
Bit 7 = 0 - Removable Medium/Device
Bit 7 = 1 - Fixed Disk
On Return:
CF = CY - Status of last operation is non-zero
CF = NC - Status of last operation is zero

(AH) - Status of Operation
(R8) - Response qword if device supports SCSI, 0 otherwise
(Byte 0) - SCSI Request Sense Key
(Byte 1) - SCSI Additional Sense Code (ASC)
(Byte 2) - SCSI Additional Sense Code Qualifier (ASCQ)
(Bytes 3-7) - Reserved

(AH) = 02H - Read Desired Sectors into Memory (CHS)

Reads a desired number of sectors from a device into system memory.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(DH)	- Head Number (not value checked, 0-based)
(CH)	- Track Number (not value checked, 0-based)
(CL)	- Sector Number (not value checked, 0-based)
(AL)	- Number of Sectors to read (not value checked)
(RBX)	- Address of buffer

On Return:

CF = CY	- Status of read is non-zero
CF = NC	- Status of read is zero
(AL)	- Number of sectors transferred
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 03H - Write Desired Sectors from Memory (CHS)

Writes a desired number of sectors of system memory to a device.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(DH)	- Head Number (not value checked, 0-based)
(CH)	- Track Number (not value checked, 0-based)
(CL)	- Sector Number (not value checked, 0-based)
(AL)	- Number of Sectors to write (not value checked)
(RBX)	- Address of buffer

On Return:

CF = CY	- Status of write is non-zero
CF = NC	- Status of write is zero
(AL)	- Number of sectors transferred
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 04H - Verify Desired Sectors (CHS)

Verifies the desired number of sectors to ensure their data integrity.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(DH)	- Head Number (not value checked, 0-based)
(CH)	- Track Number (not value checked, 0-based)
(CL)	- Sector Number (not value checked, 0-based)
(AL)	- Number of Sectors to verify (not value checked)
(RBX)	- Address of buffer

On Return:

CF = CY	- Status of verify is non-zero
CF = NC	- Status of verify is zero
(AL)	- Number of sectors verified
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 05H - Format Desired Track (CHS)

Formats track to have 9, 512 byte sectors, initialised with fill byte 0FFH.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(DH)	- Head Number (not value checked, 0-based)
(CH)	- Track Number (not value checked, 0-based)
(RBX)	- Reserved

On Return:

CF = CY	- Status of format is non-zero
CF = NC	- Status of format is zero
(AL)	- Reserved
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 06H to 81H - Reserved

These functions are reserved.

(AH) = 82H - Read Desired Sectors into Memory (LBA)

Reads a desired number of sectors from a device into system memory.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(AL)	- Number of Sectors to read (not value checked)
(RBX)	- Address of buffer
(RCX)	- LBA of Sector to begin reading from

On Return:

CF = CY	- Status of read is non-zero
CF = NC	- Status of read is zero
(AL)	- Number of sectors transferred
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 83H - Write Desired Sectors from Memory (LBA)

Writes a desired number of sectors of system memory to a device.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(AL)	- Number of Sectors to write (not value checked)
(RBX)	- Address of buffer
(RCX)	- LBA of Sector to begin writing to

On Return:

CF = CY	- Status of write is non-zero
CF = NC	- Status of write is zero
(AL)	- Number of sectors transferred
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 84H - Verify Desired Sectors (LBA)

Verifies the desired number of sectors to ensure their data integrity.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(AL)	- Number of Sectors to verify (not value checked)
(RBX)	- Address of buffer
(RCX)	- LBA of Sector to begin verifying from

On Return:

CF = CY	- Status of verify is non-zero
CF = NC	- Status of verify is zero
(AL)	- Number of sectors verified
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 85H - Format Desired Sectors (LBA)

Formats sectors to be 512 byte sectors, initialised with fill byte 0FFH.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk
(AL)	- Number of Sectors to format (not value checked)
(RCX)	- LBA of Sector to begin formatting from

On Return:

CF = CY	- Status of format is non-zero
CF = NC	- Status of format is zero
(AL)	- Reserved
(AH)	- Status of operation

Note: If an error is reported by SCP/BIOS, reset the media system then retry.

(AH) = 88H - Read Device Parameters (LBA)

Returns INT 33H device LBA based parameters.

(DL)	- Device Number (0-based)
	Bit 7 = 0 - Removable Medium/Device
	Bit 7 = 1 - Fixed Disk

On Return:

CF = NC	- Device parameters found
(RAX)	- Device block (sector) size in bytes (Dword)
(RCX)	- Last device LBA block (Qword)

(AH) = 89H to FFH - Reserved

These functions are reserved.

Interrupt 34H - Asynchronous Communications Services

These routines provide Serial communications support. The following is a summary of the SCP/BIOS Asynchronous Communications services:

(AH) = 00H	- Initialise the Communications Port
(AH) = 01H	- Send Character
(AH) = 02H	- Receive Character
(AH) = 03H	- Read Status
(AH) = 04H	- Reserved, Extended Initialise
(AH) = 05H	- Reserved, Extended Communications Port Control
(AH) = 06H to FFH	- Reserved

Table 31: INT 34H - Asynchronous Communications Functions

(AH) = 00H - Initialise the Communications Port

(AL) - Parameters for initialisation
Bits 7, 6, 5 - Baud rate (values are binary)
= 000b - 110 Baud
= 001b - 150 Baud
= 010b - 300 Baud
= 011b - 600 Baud
= 100b - 1200 Baud
= 101b - 2400 Baud
= 110b - 4800 Baud
= 111b - 9600 Baud
For Baud Rates above 9600 Baud use INT 34H, (AH) = 04H or 05H

Bits 4, 3 - Parity (values are binary)
= 00b - None
= 01b - Odd
= 10b - None
= 11b - Even

Bit 2 - Stop bit
= 0 - 1
= 1 - 2

Bits 1, 0 - Word length (values are binary)
= 10b - 7 Bits
= 11b - 8 Bits

(DX) - Serial communications line to use (0, 1, 2, 3)

On Return:

(AL) - MODEM Status

Bit 7	- Data carrier detect
Bit 6	- Ring Indicator
Bit 5	- Data set ready
Bit 4	- Clear to send
Bit 3	- Delta data carrier detect
Bit 2	- Trailing edge ring detector
Bit 1	- Delta data set ready
Bit 0	- Delta clear to send

(AH) - Line Status

Bit 7	- Time-out
Bit 6	- Transmit shift register empty
Bit 5	- Transmit holding register empty
Bit 4	- Break interrupt detect
Bit 3	- Framing error
Bit 2	- Parity error
Bit 1	- Overrun error
Bit 0	- Data ready

Note: If bit 7 of the line status byte is set to 1, the validity of the other bits becomes unpredictable.

(AH) = 01H - Send Character

(AL)	- Character to send
(DX)	- Serial communications line to use (0, 1, 2, 3)

On Return:

(AL)	- Preserved
(AH)	- Line status

(AH) = 02H - Send Character

(DX)	- Serial communications line to use (0, 1, 2, 3)

On Return:

(AL)	- Character received
(AH)	- Line status

Note: This routine is a blocking wait that waits for a character to arrive.

(AH) = 03H - Read Status

(DX)	- Serial communications line to use (0, 1, 2, 3)

On Return:

(AL)	- MODEM status
(AH)	- Line status

(AH) = 04H - Reserved, Extended Initialise

This function is reserved for use in future versions of SCP/BIOS.

(AH) = 05H - Reserved, Extended Communication Port Control

This function is reserved for use in future versions of SCP/BIOS.

(AH) = 06H to FFH - Reserved

These functions are reserved by SCP/BIOS.

Interrupt 35H - System Services

The following is a summary of the system services provided by SCP/BIOS accessible by Interrupt 35H:

(AH) = 00H to 82H	- Reserved
(AH) = 83H	- Reserved, Event Wait
(AH) = 84H	- Reserved
(AH) = 85H	- Reserved
(AH) = 86H	- Wait
(AH) = 87H	- Reserved
(AH) = 88H	- Simple extended memory size determination
(AH) = 89H to C4H	- Reserved
(AH) = C5H	- Miscellaneous system function dispatcher
(AH) = C6H to E7H	- Reserved
(AH) = E8H	- Advanced memory management system dispatcher
(AH) = E9H to EFH	- Reserved
(AH) = F0H	- System data table dispatcher
(AH) = F1H	- EHCI system dispatcher
(AH) = F2H to FFH	- Reserved

Table 32: INT 35H - System Services Functions

The SCP/BIOS system services function utilises the notion of dispatchers. Each dispatcher function groups together functions that are related and presents them as subfunctions of the dispatcher. Subfunctions are specified by placing the subfunction number in register (AL) to specify the desired subfunction.

(AH) = 00H to 82H - Reserved

These functions are reserved by SCP/BIOS.

(AH) = 83H - Reserved, Event Wait

This function is reserved by SCP/BIOS for a non-blocking wait.

(AH) = 84H - Reserved

This function is reserved by SCP/BIOS.

(AH) = 85H - Reserved

This function is reserved by SCP/BIOS.

(AH) = 86H - Wait

This function will cause the caller to wait for a specified number of milliseconds (multiples of 976 microseconds respectively). This function will not return control back to the caller until the specified number of milliseconds have elapsed.

(RCX)	- Number of milliseconds to wait (multiples of 976 microseconds respectively)

(AH) = 87H - Reserved

This function is reserved by SCP/BIOS.

(AH) = 88H - Simple extended memory size determination

On Return:

(AX)	- Number of contiguous 1KB blocks starting at the first KB after the end of SCP/BIOS and ending at 15MB.

(AH) = 89H to C4H - Reserved

These functions are reserved by SCP/BIOS.

(AH) = C5H - Miscellaneous system function dispatcher

Reserved entries are reserved for future use by SCP/BIOS.

(AL) = 00H - Sound PC Speaker

(BX) - Frequency Divisor Value (16-bit value, must not be 1)

(RCX) - Duration to sound speaker for in milliseconds
(multiples of 976 microseconds respectively)

(AL) = 01H - Connect System Debugger

Hooks Interrupts 01H, 03H and 3BH to allow single-stepping, breaking into software breakpoints and entering through Ctrl+BREAK. Overrides any previous interrupt descriptors for those interrupts.

(AL) = 02H - Disconnect System Debugger

Disconnects Interrupts 01H, 03H and 3BH, returning them to the SCP/BIOS default interrupt handlers.

(AL) = 03H to FFH - Reserved

(AH) = E8H - Advanced memory management system dispatcher

Reserved entries are reserved for future use by SCP/BIOS.

(AL) = 00H - Get pointer to the first KB after the end of SCP/BIOS

On Return:

(RAX)	- Pointer to first KB after the end of SCP/BIOS

(AL) = 01H - Extended memory size determination

On Return:

(AX)	- Number of contiguous 1KB blocks from the first KB after the end of SCP/BIOS to 16MB
(BX)	- Number of contiguous 64KB blocks from 16MB to 64MB
(CX)	- See (AX)
(DX)	- See (BX)

(AL) = 02H - SCP/BIOS Segment Information

On Return:

(RAX)	- Start of SCP/BIOS segment
(BL)	- Reserved
(RCX)	- Total sum of all SCP/BIOS segments
(RDX)	- Total free system memory

All registers except (RSP), (RBP) and (R15) are reserved.

(AL) = 03H to 1FH - Reserved

(AL) = 20H - Get System Memory Map

On Return:

(RAX)	- Pointer to the first KB after the end of SCP/BIOS
(CX)	- Number of 24-byte entries in the memory map
(RSI)	- Pointer to the memory map

(AL) = 21H to FFH - Reserved

(AH) = F0H - System data table dispatcher

Reserved entries are reserved for future use by SCP/BIOS.

(AL) = 00H - Register New GDT Pointer
Updates SCP/BIOS with the location of a new GDT to be used.

(RBX)	- Pointer to new GDT
(CX)	- Limit of new GDT
(EDX)	- Number of entries in GDT

(AL) = 01H - Register New IDT Pointer
Updates SCP/BIOS with the location of a new IDT to be used.

(RBX)	- Pointer to new IDT
(CX)	- Limit of new IDT
(EDX)	- Number of entries in IDT

(AL) = 03H - Get Current GDT Pointer
Returns the current, in use, GDT pointer.

On Return:

(RBX)	- Pointer to GDT base
(CX)	- GDT Limit
(EDX)	- Number of entries in GDT

(AL) = 04H - Get Current IDT Pointer
Returns the current, in use, IDT pointer.

On Return:

(RBX)	- Pointer to IDT base
(CX)	- IDT Limit
(EDX)	- Number of entries in IDT

(AL) = 05H - Register New Page Table Base Pointer
Updates SCP/BIOS with the location of a new set of page tables.

(RBX)	- Pointer to new system page table base

(AL) = 06H - Get Current Page Table Base Pointer

Returns a pointer to the current, in use, page table base.

On Return:

(RBX) - Pointer to current system page table base

(AL) = 07H - Set IDT Descriptor

Installs a new interrupt descriptor in the current IDT.

(RBX) - Pointer to new interrupt handler.
(CX) - Interrupt descriptor number (0-number of IDT entries).
(DX) - Attribute word of interrupt descriptor.
(SI) - GDT segment selector of interrupt descriptor.

(AL) = 08H - Get IDT Descriptor

Returns an interrupt descriptor from the current IDT.

On return

(AX) - GDT segment selector of interrupt descriptor.
(RBX) - Pointer to interrupt handler.
(DX) - Attribute word of interrupt descriptor.

(AL) = 09H - Register new Removable Device CHS Translation Table

(RBX) - Pointer to valid new CHS Translation Table.

(AL) = 0AH - Get current Removable Device CHS Translation Table

On return

(RBX) - Pointer to CHS Translation Table.

(AL) = 0BH - Reserved

(AL) = 0CH - Reserved

(AL) = 0DH - Register new SysInit Parameters

(RBX) - New first logical block to copy
(DX) - New number of contiguous blocks to copy. DH must be 0

(AL) = 0EH - Get current SysInit Parameters

On return

(RBX) - First logical block that will be copied
(DX) - Number of contiguous that will be copied

(AL) = 0FH to FFH - Reserved

(AH) = F1H - EHCI system dispatcher

Reserved entries are for future use by SCP/BIOS.

(AL) = 00H - Get pointer to EHCI Critical Error Handler

Returns a pointer to the current EHCI critical error handler.

On return

(RBX) - Pointer to EHCI critical error handler.

(AL) = 01H - Set pointer to EHCI Critical Error Handler

Sets a pointer to a new EHCI critical error handler.

(RBX) - Pointer to new EHCI critical error handler.

(AL) = 02H to FFH - Reserved

Interrupt 36H - Keyboard Services

These routines provide keyboard support. The following is a summary of the keyboard functions provided by SCP/BIOS accessible via Interrupt 36H:

(AH) = 00H	- Keyboard Read
(AH) = 01H	- Keystroke Status
(AH) = 02H	- Shift Status
(AH) = 03H to FFH	- Reserved

Table 33: INT 36H - Keyboard Functions

The keyboard functions of SCP/BIOS provide support for any 101/102/104/105-key PS/2 keyboard and utilises IBM scan-code set 1. IF the keyboard natively supports a different scan-code set, the BIOS ensures they are translated before they are put into the keyboard buffer. The keyboard buffer is large enough to hold up to 16 keystrokes at any one time and stores each keystroke as a scan-code/character pair. If the key does not have a standard ASCII character code (such as the function keys) then an ASCII NUL is placed in the buffer.

(AH) = 00H - Keyboard Read

The scan-code/character pair is extracted from the buffer. If the buffer is empty, the function enters a (blocking) keyboard loop, waiting for a key to be pressed before returning.

On Return:	
(AL)	- ASCII character code
(AH)	- Scan-code

Note: Control is returned only when a keystroke is available.

(AH) = 01H - Keyboard Status

This is a non-blocking function and allows the programmer to detect if a new keystroke is available. If there is, (AX) will contain the new keystroke however, the programmer should then call Keyboard Read to prevent the user from filling the keyboard buffer.

On Return:		
ZF = 1	- No keystroke available	
ZF = 0	- Keystroke available	
	(AL)	- ASCII character code
	(AH)	- Scan-code

Note: Control is returned immediately returned to the caller.

(AH) = 02H - Shift Status

On Return:		
(AL)	- Current Shift Status	
	Bit 7	= 1 - Insert locked
	Bit 6	= 1 - Caps Lock locked
	Bit 5	= 1 - Num Lock locked
	Bit 4	= 1 - Scroll Lock locked
	Bit 3	= 1 - (Either) Alt key pressed
	Bit 2	= 1 - (Either) Ctrl key pressed
	Bit 1	= 1 - Left Shift key pressed
	Bit 0	= 1 - Right Shift key pressed
(AH)	- Reserved	

Interrupt 37H - SCP/BIOS Reserved

This interrupt vector is reserved for future use by SCP/BIOS and may not be used for any purpose. Using this interrupt vector in software may result in that software becoming incompatible with future versions of SCP/BIOS.

Interrupt 38H - SYSDEBUG

Calling this interrupt vector allows a programmer to enter the BIOS debugger SYSDEBUG. The SYSDEBUG user guide is outlined in Appendix C.

SYSDEBUG acts as a basic monitor and debugger for SCP/BIOS and allows a programmer to manually manipulate system registers, memory and I/O addresses, dump system and I/O memory, insert and remove software and hardware breakpoints, load and write sectors from and to Block Storage Devices, view the system memory map, single step through procedures and jump and continue program execution from any location in system memory. If SCP/BIOS fails to detect a valid boot device, a programmer can use SCP/BIOS to try and manually boot. This procedure is outlined in Appendix C.

If a programmer wishes to test software that runs under SCP/BIOS, they may do so by using the debugger. It is recommended that a programmer connects the debugger to SCP/BIOS before debugging any code as this will allow the program to break into SYSDEBUG upon hitting breakpoints or if the programmer presses Ctrl+BREAK. A programmer can do so programmatically using INT 35H, AX = C501H.

Once the programmer is done debugging, it is recommended the programmer disconnects the debugger from SCP/BIOS. They may do so programmatically using INT 35H, AX = C502H.

Warning! SYSDEBUG is non-reentrant. You must not enter SYSDEBUG from within SYSDEBUG as doing so will erase the original saved program state. Doing so is easy (simply connect the debugger and press Ctrl+BREAK twice) so care must be taken to not do so.

Interrupt 39H - Warm Reboot Service

Calling this interrupt vector allows a programmer to reboot the machine. The machine will first attempt to read a valid boot sector from INT 33H device 0 to memory location 7C00H.

If a valid boot sector is found, SCP/BIOS will jump to that sector. The following registers will be initialised to the following values.

(RBX) - The first logical block after the end of SCP/BIOS on boot device
(DX) - The number of the boot device
(RSP) - 0DFF8H, a 512 qword stack is set up from D000H-DFFFH
(FS) - The FS MSR will contain a pointer to the first KB after the end of SCP/BIOS
(CS) - 0008H, Default Code Segment Selector
(RIP) - 07C02H

If no valid boot sector is found, then INT 39H will return with the carry flag set.

Note:

- Calling INT 39H will not reinitialise SCP/BIOS

Interrupt 3AH - System-Timer and Real-Time Clock Services

The following is a summary of the system-timer and real-time clock services of Interrupt 3AH.

(AH) = 00H	- Read System-Timer Time Counter
(AH) = 01H	- Set System-Timer Time Counter
(AH) = 02H	- Read Real-Time Clock Time
(AH) = 03H	- Set Real-Time Clock Time
(AH) = 04H	- Read Real-Time Clock Date
(AH) = 05H	- Set Real-Time Clock Date
(AH) = 06H	- Set Real-Time Clock Alarm
(AH) = 07H	- Reset Real-Time Clock Time
(AH) = 08H to FFH	- Reserved

Table 34: INT 3AH - System-Timer and Real-Time Clock Services

(AH) = 00H - Read System-Timer Time Counter

On Return:

(AL) - System 24-hour flag
= 00H - System has not passed 24 hours since power-on, system reset, reset or last count update
> 00H - System has passed 24 hours since power-on, system reset, or last count update
(DX) - Low portion of count
(CX) - High portion of count

(AH) = 01H - Read System-Timer Time Counter

(DX) - Low portion of count
(CX) - High portion of count

(AH) = 02H - Read Real-Time Clock Time

On Return:

(CH)	- Hours, in BCD
(CL)	- Minutes, in BCD
(DH)	- Seconds, in BCD
(DL)	- Daylight saving flag
	= 00H - Daylight savings mode deactivated
	= 01H - Daylight saving mode activated
CF	- Real-Time Clock operating status
	= NC - Clock operating
	= CY - Clock not operating

(AH) = 03H - Set Real-Time Clock Time

(CH)	- Hours, in BCD
(CL)	- Minutes, in BCD
(DH)	- Seconds, in BCD
(DL)	- Daylight saving flag
	= 00H - Do not set daylight savings mode
	= 01H - Set daylight saving mode

On Return:

CF	- Real-Time Clock status
	= NC - Clock operating
	= CY - Clock not operating

(AH) = 04H - Read Real-Time Clock Date

On Return:

(CH)	- Reserved
(CL)	- Year, in BCD
(DH)	- Month, in BCD
(DL)	- Day, in BCD
CF	- Real-Time Clock operating status
	= NC - Clock operating
	= CY - Clock not operating

(AH) = 05H - Set Real-Time Clock Date

(CH)	- Reserved
(CL)	- Year, in BCD
(DH)	- Months, in BCD
(DL)	- Day, in BCD

On Return:

CF	- Real-Time Clock status
	= NC - Clock operating
	= CY - Clock not operating

(AH) = 06H - Set Real-Time Clock Alarm

(CH)	- Hours, in BCD
(CL)	- Minutes, in BCD
(DH)	- Seconds, in BCD

On Return:

CF	- Real-Time Clock status
	= NC - Alarm set successfully
	= CY - Alarm already set or clock not operating.

Note: The alarm interrupt occurs at the specified hour, minute and second that are passed in (CH), (CL) and (DH) respectively. When the alarm interrupt occurs Interrupt 6AH is triggered. The programmer using the alarm must first install an interrupt handler for INT 6AH before setting the alarm using INT 3AH. Only one alarm function can be active at any one time. The alarm interrupt will occur once at the time specified and every 24-hours thereafter until the alarm is reset.

(AH) = 07H - Reset Real-Time Clock Alarm

Calling this function will stop the Real-Time Clock alarm from occurring.

(AH) = 08H to FFH - Reserved

These functions are reserved for future use by SCP/BIOS.

Interrupt 3BH - Keyboard Break Handler

This interrupt is reserved by SCP/BIOS for use as an intercept vector, to allow a programmer to intercept a Ctrl + BREAK keypress. A programmer is free to hook their own interrupt handlers for this vector to intercept a Ctrl + BREAK keypress. SCP/BIOS installs a default handler on system initialisation. The installed interrupt handler is entered every time a Ctrl + BREAK keypress occurs.

Interrupt 3CH - System Clock Handler

This interrupt is reserved by SCP/BIOS for use as an intercept vector, to allow a programmer to intercept an a system timer tick. A programmer is free to hook their own interrupt handlers for this vector to intercept a system timer tick. SCP/BIOS installs a default handler on system initialisation. The installed interrupt handler is entered every time an IRQ 0 event occurs.

Interrupt 3DH - Screen Mode Parameters

On Return:
(R8) - Pointer to the screen mode parameter table

Interrupt 3EH - Block Storage Device Parameters

On Return:
(R8) - Pointer to the removable block storage device geometry table
(R9) - Reserved

Interrupt 3FH - Video Extension Parameters

This interrupt is reserved for future use by SCP/BIOS.

Interrupts 40H - 5Ah, User Definable Interrupts

These interrupt vectors are user definable and may be used a programmer for any purpose.

Interrupts 60H - 69h, SCP/BIOS Reserved

These interrupt vectors are reserved for future use by SCP/BIOS and may not be used for any purpose. Using these interrupt vectors in software may result in that software becoming incompatible with future versions of SCP/BIOS.

Interrupt 6AH - RTC Alarm Handler Interrupt

This interrupt vector is reserved for use by the RTC Alarm. When an RTC Alarm interrupt occurs this interrupt is triggered by the RTC Alarm handler. By default, this interrupt simply exits. This interrupt vector may be used by a programmer who may need to run a procedure when the RTC Alarm is triggered.

Interrupts 6BH - FEh, User Definable Interrupts

These interrupts vectors are user definable and may be used a programmer for any purpose.

Interrupt FFH - SCP/BIOS Reserved

This interrupt vector is reserved for future use by SCP/BIOS and may not be used for any purpose. Using this interrupt vector in software may result in that software becoming incompatible with future versions of SCP/BIOS.

Real-Mode Interrupt Vector FFH - SCP/BIOS Reserved

This real mode interrupt vector is reserved for future use by SCP/BIOS and may not be used for any purpose by system software. Using this interrupt vector in software may result in that software becoming incompatible with future versions of SCP/BIOS.

Data Areas and Tables

SCP/BIOS uses three main data segments:

- The system data table area
- The BIOS data area
- The dynamic transaction area

Each of these areas are undocumented except for those tables which are explicitly listed herein. Attempting to access the undocumented areas violates SCP/BIOS and changing the values within may cause system instability or system crashes. Furthermore, the areas that are undocumented may change their structure and/or location in system memory in a future version of SCP/BIOS without warning, which could lead to future application incompatibilities with SCP/BIOS

The system data table is used to save the CPU data about the system, such as the BIOS GDT, IDT and paging tables. Changing any information in this area without using an SCP/BIOS function may cause system instability and crashes.

The BIOS data area is the main data segment for SCP/BIOS. It is used by SCP/BIOS to store variables used for various internal purposes. It may be tempting to write an application that accesses the variables in this area directly as opposed to getting them by making a call to the appropriate SCP/BIOS function, but doing so may lead to future system incompatibility. This area contains the device tables for the various system devices such as asynchronous serial communication devices, block storage devices, as well as character buffers for the system keyboard and asynchronous serial communications devices.

The dynamic transaction area is used to initiate data transfers with block storage devices and store objects of indeterminate size. A programmer must not under any circumstances attempt to write to this area as doing so can lead to system instability and failure.

Each of these areas uses numerous internal data tables and data structures, each of which are described in the SCP/BIOS listing. A programmer should not attempt to make use of these tables and/or modify these tables in any way whilst using SCP/BIOS as doing so could cause the system to misbehave and crash.

The only documented table that can modified by a programmer is the "Removable Device CHS Translation Table". This translation table provides the INT 33H functions that rely on CHS addressing for Block Storage devices, a disk geometry which can be used to emulate a CHS interface even if the device has no physical platters. The default system translation table emulates a 1.44Mb double-sided, double-density floppy disk drive (C=80, H=2, S=9). Therefore, a programmer using the CHS INT 33H functions will only be able to access the first 1.44Mb of the device they are attempting to access.

These geometries are prototypical and a programmer may replace these data tables with their own geometries, using the relevant documented INT 35H dispatcher function. The table must adhere to the structure outlined in the example table below:

3 Bytes	- Reserved, must be set to 0
1 Byte	- Reserved, must be set to 2
1 Byte	- Sectors per track
3 Bytes	- Reserved, must be set to 0
1 Byte	- Fill byte to fill sector with during format
1 Byte	- Reserved, must be set to 0
1 Byte	- Reserved, must be set to 1

Table 35: Removable Device CHS Translation Table Structure

This table has a limitation that enforces a 2 head and 80 cylinder structure, however, a programmer may attempt to program a disk geometry with up to 255 sectors. Byte 4, in future versions will be used to allow a programmer to make requests with "large" sectors, though for now, the sector size is fixed at 512 bytes and this field must be set to 2.

Appendix A: BIOS Listing

SCP/BIOS Listing

LINE LOC OBJ SOURCE

```
[map all scpio64.map]
;--------------------------------SCPIO.SYS--------------------------------
;---------------------------------Equates---------------------------------
permissionflags equ 003h    ;Page table Permission flags
codedescriptor      equ 0008h

BIOSStartAddr   equ 00110000h   ;Start just after HMA + 16 bytes
BIOSInitAddr    equ 800h

e820Seg         equ 1000h
e820SizeOff     equ 0000h     ;First word is # of entries
e820BaseOff     equ e820SizeOff + 2
e820SizeAddr    equ (e820Seg<<4) + e820SizeOff
;-----------------------PIC Chip IO values------------------------
pic1command     equ 020h          ;Command port
pic2command     equ 0A0h          ;Command port
pic1data        equ 021h          ;Data port
pic2data        equ 0A1h          ;Data port
;------------------------------------------------------------------
;--------------------PS/2 IO port commands-------------------------
ps2command      equ 64h             ;Command Port (write)
ps2status       equ 64h             ;Status Port  (read)
ps2data         equ 60h            ;Data Port     (read/write)
;------------------------------------------------------------------
;----------------------Serial port equates-------------------------
com1_base       equ 03F8h
com2_base       equ 02F8h
com3_base       equ 03E8h
com4_base       equ 02E8h
;------------------------------------------------------------------
;-----------------------PIT port equates---------------------------
PITbase         equ 40h
PIT0            equ PITbase
PIT1            equ PITbase + 1
PIT2            equ PITbase + 2
PITcommand      equ PITbase + 3
;------------------------------------------------------------------
;-----------------------CMOS port equates--------------------------
cmos_base        equ 70h
cmos_data        equ 71h
;------------------------------------------------------------------
;----------------------Keyboard equates----------------------------
kb_flag_rshift   equ     01h     ;Right Shift is being held
kb_flag_lshift   equ     02h     ;Left Shift is being held
kb_flag_ctrl     equ     04h     ;Ctrl is being held
kb_flag_alt      equ     08h     ;Alt is being held
kb_flag_scrlset  equ     10h     ;Scroll lock is set
kb_flag_numset   equ     20h     ;Num lock is set
kb_flag_capsset  equ     40h     ;Caps lock is set
kb_flag_insset   equ     80h     ;Insert mode is set

kb_flag2_e1      equ     01h     ;0E1h scancode procedure being
                                    processed
kb_flag2_e0      equ     02h     ;0E0h scancode procedure being
                                    processed
;------------------------------------------------------------------
;----------------------Screen equates------------------------------
vga_index        equ     03D4h
```

```
vga_data        equ     03D5h
;---------------- New Equates ----------------
vga_aindex      equ     03B4h     ;Alt (MDA) IO Base
vga_adata       equ     03B5h
;These equates are SEGMENTS, need to be SHL 4 to become addrs
vga_bpage0      equ     0A0000h
vga_bpage1      equ     0B0000h
vga_bpage2      equ     0B8000h
;-----------------------------------------------------------------
;----------------------PCI equates--------------------------
pci_index       equ     0CF8h
pci_data        equ     0CFCh
;-----------------------------------------------------------------
;----------------------USB equates--------------------------
usb_class       equ     0Ch     ;pci class
usb_subclass    equ     03h     ;pci subclase
uhci_interface  equ     00h     ;usb 1.0
uhcimask        equ     10h
ohci_interface  equ     10h     ;usb 1.0 alt
ohcimask        equ     20h
ehci_interface  equ     20h     ;usb 2.0
ehcimask        equ     40h
xhci_interface  equ     30h     ;usb 3.0
xhcimask        equ     80h
lousbtablesize  equ     0000E000h     ;Location of the table size, uword
lousbtablebase  equ     lousbtablesize + 2 ;base of the table, tword entries
debounceperiod  equ 200 ;double 200ms as per Windows, for inaccuracies
;               -------EHCI equates-------
ehcicaplength   equ     00h     ;Add this to base addr in table to find opparams
ehciversion     equ     02h     ;Interface Version number
ehcistrucparams equ     04h     ;Structural Parameters
ehcihccparams   equ     08h     ;Capability Parameters
ehciportroute   equ     0Ch     ;Companion Port Route Description (v1 ignore)

;Operational registers below

ehcicmd         equ     00h     ;USB command register
ehcists         equ     04h     ;USB status register
ehciintr        equ     08h     ;USB Interrupt Enable
ehcifrindex     equ     0Ch     ;USB Frame Index
ehcictrlseg     equ     10h     ;4Gb Segment Selector
ehciperiodbase  equ     14h     ;Frame List Base Address
ehciasyncaddr   equ     18h     ;Next Asynchronus List Address
ehciconfigflag  equ     40h     ;Configured Flag Register
ehciportsc      equ     44h     ;Read = 1 - # of ports, Write = port ctrl
;               -------------------------
;               --------MSD equates-------
setupReset      equ 0FFh
setupGetMaxLUN  equ 0FEh
;               -------------------------
;               -------Bulk Storage equates-------
CBWSig          equ     043425355h
CSWSig          equ 053425355h
CBWFlagOut      equ     00h     ;Switch to send to device
CBWFlagIn       equ     80h     ;Switch to recieve from
bCSWPassed      equ     00h
bCSWFailed      equ     01h
bCSWPhase       equ     02h
;               -------------------------
;-----------------USB Device table entry sizes-----------------
msdDevTblEntrySize      equ 10h
hubDevTblEntrySize      equ 8h
usbDevTblEntrySize      equ 3h
usbMaxDevices           equ 10
;               -------------------------
;-----------------EHCI Transfer Descriptor size-----------------
ehciSizeOfQH    equ 60h
ehciSizeOfTD    equ 40h
;-----------------------------------------------------------------
;----------------------ATA equates--------------------------
ata0_base       equ     1F0h
ata0_ctrl       equ     3F6h
ata1_base       equ     170h
ata1_ctrl       equ     376h

msd_class       equ     01h
ide_subclass    equ     01h
```

```
                                   sata_subclass   equ     06h
                                   ;--------------------------------------------------------------
                                   ;-------------------------IDE equates--------------------------
                                   ideTableEntrySize     equ 10h
                                   ;------------------------FDD equates---------------------------
                                   fdd_base        equ     3F0h
                                   ;-------------------------Int 33h Equates----------------------
                                   fdiskTableEntrySize   equ 10h
                                   int33TblEntrySize     equ 10h
                                   ;--------------------------------------------------------------
                                   ;--------------------------------------------------------------
                                   ;---------------------------Misc-------------------------------
                                   port61h         equ 61h         ;I/O port 61h
                                   EOI             equ 20h         ;End of interrupt signal
                                   waitp           equ 80h         ;debug port used to wait for io
                                                                    cycles
                                   bochsout        equ 0E9h        ;Emulator debug port
                                   BREAKPOINT      equ 0CCh        ;Use to manually encode breakpoints
                                                                    in program
                                   sizeOfMCPAlloc  equ 800h        ;2Kb allocated space
                                   ;--------------------------------------------------------------
                                   ;------------------------------------------------------------------
                                   ;                     BIOS SYSTEM TABLE AREA                     :
                                   ;------------------------------------------------------------------
                                   Segment BIOSTables nobits start=BIOSStartAddr align=1
00000000 <res 1000h>               BIOSIDTable     resq 2*256  ;256 paragraph entries reserved for IDT
00001000 <res 6000h>               BIOSPageTbl     resq 0C00h  ;6000 bytes for page tables
00007000 <res 18h>                 BIOSGDTable     resq 3      ;3 entries in basic GDT
00007018 ????????????????                          resq 1      ;Alignment qword
                                   ;------------------------------------------------------------------
                                   ;                     BIOS DATA AREA STARTS HERE                 :
                                   ;------------------------------------------------------------------
                                   Segment data nobits follows=BIOSTables align=1
                                   ;Refer to MEMMAP.TXT for memory address reference!
                                   ;If Interrupt call is faulty, Carry will be set AND either:
                                   ;                   ah=80h => Invalid function.
                                   ;                   ah=86h => Not (yet) supported.
                                   ;--------------------------------Data Area---------------------------
00000000 ????                      IDTlength       resw 1  ;Maximum number of Interrupts is 256
                                   IDTpointer:
00000002 ????                      .Limit          resw 1
00000004 ????????????????          .Base           resq 1

0000000C ????                      GDTlength       resw 1
                                   GDTpointer:
0000000E ????                      .Limit          resw 1
00000010 ????????????????          .Base           resq 1

00000018 ????????????????          pageTablePtr:   resq 1
                                   ;------------------------------------------
                                   ;        Spurious Interrupt counter       :
                                   ;------------------------------------------
00000020 ??                        spurint1        resb 1     ;Keep track of how many spur ints on pic1
00000021 ??                        spurint2        resb 1     ;pic 2
                                   ;------------------------------------------
                                   ;              Keyboard Data Area          :
                                   ;------------------------------------------
00000022 <res 20h>                 kb_buffer       resw 10h
00000042 ????????????????          kb_buf_head     resq 1     ;Pointer to Keyboard buffer head
0000004A ????????????????          kb_buf_tail     resq 1     ;Pointer to Keyboard buffer tail
00000052 ????????????????          kb_buf_start    resq 1     ;Pointer for circular buffer start
0000005A ????????????????          kb_buf_end      resq 1     ;Ditto..., for end
00000062 ??                        kb_flags        resb 1     ;Keyboard state flags
00000063 ??                        kb_flags_1      resb 1     ;Extended flags, empty for now
00000064 ??                        kb_flags_2      resb 1     ;Bit 0 = E1 present, Bit 1 = E0 present
00000065 ??                        break_flag      resb 1     ;Well, its not for the Print Screen key
                                   ;------------------------------------------
                                   ;               Serial Data Area           :
                                   ;------------------------------------------
00000066 ??                        numCOM          resb 1   ;Number of Serial Ports
00000067 ????????????????          com_addresses   resw 4       ;Space for 4 IO addresses

                                   comX_buffer:
0000006F <res 10h>                 com1_buffer     resb 10h
0000007F <res 10h>                 com2_buffer     resb 10h
0000008F <res 10h>                 com3_buffer     resb 10h
0000009F <res 10h>                 com4_buffer     resb 10h

                                   comX_buf_head:
000000AF ????????????????          com1_buf_head   resq 1
000000B7 ????????????????          com2_buf_head   resq 1
000000BF ????????????????          com3_buf_head   resq 1
000000C7 ????????????????          com4_buf_head   resq 1
```

```

                                   comX_buf_tail:
000000CF ????????????????          com1_buf_tail    resq 1
000000D7 ????????????????          com2_buf_tail    resq 1
000000DF ????????????????          com3_buf_tail    resq 1
000000E7 ????????????????          com4_buf_tail    resq 1

                                   comX_buf_start:
000000EF ????????????????          com1_buf_start   resq 1
000000F7 ????????????????          com2_buf_start   resq 1
000000FF ????????????????          com3_buf_start   resq 1
00000107 ????????????????          com4_buf_start   resq 1

                                   comX_buf_end:
0000010F ????????????????          com1_buf_end     resq 1
00000117 ????????????????          com2_buf_end     resq 1
0000011F ????????????????          com3_buf_end     resq 1
00000127 ????????????????          com4_buf_end     resq 1

                                   ;------------------------------------------
                                   ;             Printer Data Area           :
                                   ;------------------------------------------
0000012F ????????????              prt_addresses    resw 3    ;Space for 3 IO addresses
                                   ;------------------------------------------
                                   ;             Timer Data Area             :
                                   ;------------------------------------------
00000135 ????                      pit_divisor      resw 1
00000137 ????????                  pit_ticks        resd 1    ;Similar to IBM PC, only with default
                                                                    divisor
                                   ;[31]=OF cnt, [30:21]=Res [20:16]=Hi cnt, [15,0]=Lo cnt
0000013B ????????????????          rtc_ticks        resq 1
                                   ;------------------------------------------
                                   ;             Screen Data Area            :
                                   ;------------------------------------------
00000143 <res 10h>                 scr_curs_pos     resw 8    ;Cursor pos, hi byte = row / lo byte =
                                                                    column
00000153 ??                        scr_cols         resb 1    ;80 Cols
00000154 ??                        scr_rows         resb 1    ;25 Rows
00000155 ????                      scr_curs_shape   resw 1    ;Packed start/end scan line
00000157 ??                        scr_char_attr    resb 1    ;Grey text on black background
00000158 ??                        scr_mode         resb 1    ;80x25, 16 colours default
00000159 ??                        scr_active_page  resb 1    ;Mode dependent
0000015A ????                      scr_crtc_base    resw 1    ;03D4h for Graphics, 03B4h for MDA
0000015C ????????                  scr_page_addr    resd 1    ;CRTC Register 12 changes base address
                                                                    accessed
00000160 ????????????????          scr_mode_params  resq 1    ;Stub pointer location for future mode
                                                                    parameters
00000168 <res 40h>                 scr_vga_ptrs     resq 8  ;VGA pointers
                                   ;------------------------------------------
                                   ;        Mass storage Data Area           :
                                   ;------------------------------------------
000001A8 ??                        i33Devices       resb 1  ;Number of devices Int 33h is aware of
000001A9 ??                        msdStatus        resb 1  ;Status byte. Used by BIOS for all
                                                                    transfers with MSD.
000001AA ??                        fdiskNum         resb 1  ;Number of fixed disks
000001AB ??                        ir14_mutex       resb 1
000001AC ??                        ir14_status      resb 1
000001AD ??                        ir15_mutex       resb 1
000001AE ??                        ir15_status      resb 1
000001AF ????????????????          diskDptPtr       resq 1
000001B7 ????????????????          fdiskDptPtr      resq 1
                                   ;------------------------------------------
                                   ;             SysInit Data Area           :
                                   ;------------------------------------------
000001BF ????????????????          nextFilePtr      resq 1  ;Pointer to next file to load
000001C7 ????                      numSectors       resw 1  ;Number of sectors to copy
                                   ;------------------------------------------
                                   ;             Memory Data Area            :
                                   ;------------------------------------------
000001C9 ????                      MachineWord      resw 1    ;Really Legacy Hardware Bitfield
000001CB ????                      convRAM          resw 1  ;Conventional memory word
000001CD ????????????????          userBase         resq 1    ;Start address of the user space
000001D5 ??                        bigmapSize       resb 1    ;First byte, in units of 24 bytes
000001D6 ????????????????          srData           resw 4  ;4 words for memory64MB word 0 is ax word 1
                                                                    is bx etc.
000001DE ????                      srData1          resw 1  ;Reserve 1 word for memory16MB
000001E0 ????????????????          sysMem           resq 1  ;Size of usable system RAM (without
                                                                    SCP/BIOS)
000001E8 ????????                  scpSize          resd 1  ;Size of SCP/BIOS allocation
                                   ;------------------------------------------
                                   ;             MCP Data Area               :
                                   ;------------------------------------------
000001EC ????????????????          mcpUserBase      resq 1  ;Pointer to register save space
```

```
000001F4 ????????????????          mcpUserRip       resq 1  ;Save the custom user RIP for new jumps
000001FC ????????????????          mcpUserkeybf     resq 1  ;Pointer to the keyboard buffer
00000204 ????????????????          mcpUserRaxStore  resq 1  ;Temp rax save space
0000020C ????????????????          mcpStackPtr      resq 1  ;Address of base of user Stack Pointer
                                   ;------------------------------------------------------------
                                   ;              USB Data Area                              :
                                   ;------------------------------------------------------------
00000214 ??                        eControllers     resb 1     ;Number of EHCI controllers
00000215 <res 20h>                 eControllerList  resq 4     ;Entry = PCI space addr|MMIO addrs
00000235 ??                        usbDevices       resb 1     ;Max value, 10 for now!
00000236 ????????????????          eHCErrorHandler  resq 1  ;Address of default error handler
                                   ;------------------------------------------------------------
                                   ;              EHCI Async Area                            :
                                   ;------------------------------------------------------------
0000023E ????????????????          eCurrAsyncHead   resq 1       ;Point to the current head of the async
                                                                      list
00000246 ??                        eNewBus          resb 1       ;Default to 0, if 1, a new bus was
                                                                      selected
00000247 ??                        eActiveCtrlr     resb 1         ;Current working controller (default
                                                                      -1)
00000248 ??                        eActiveInt       resb 1         ;Gives a copy of the usbsts intr bits
00000249 ??                        eAsyncMutex      resb 1
                                        ;Mutex, x1b=data NOT ready, wait. x0b=ready, data ready to
                                                                      access.
                                        ;         1xb=Internal buffer. 0xb=user provided buffer.
                                        ;         bits [7:2], number of interrupts to ignore (if any)
                                        ;              a value of 0 means dont ignore
                                   ;------------------------------------------------------------
                                   ;              MSD Data Area                              :
                                   ;------------------------------------------------------------
0000024A ??                        cbwTag           resb 1         ;cbw transaction unique id (inc post
                                                                      use)
0000024B ??                        numMSD           resb 1         ;Number of MSD devices
                                   ;------------------------------------------------------------
                                   ;              USB Tables                                 :
                                   ;------------------------------------------------------------
0000024C <res 1Eh>                 usbDevTbl        resb 10*usbDevTblEntrySize
                                   usbDevTblEnd     equ $
                                   usbDevTblE       equ ($ - usbDevTbl)/usbDevTblEntrySize ;Number of
                                                                      Entries
                                   ;Byte 0 = Dev Addr, Byte 1 = Root hub, Byte 2 = Class Code (USB
                                                                      standard)
                                   ; i.e. 08h=MSD, 09h=Hub
0000026A <res 50h>                 hubDevTbl        resb 10*hubDevTblEntrySize
                                   hubDevTblEnd     equ $
                                   hubDevTblE       equ ($ - hubDevTbl)/hubDevTblEntrySize
                                   ;bAddress - The assigned device address
                                   ;bBus - Host Bus [Root hub]
                                   ;bHostHub - Address of Hub we are attached to or 0 for Root
                                   ;bHubPort - Port number we are inserted in
                                   ;bMaxPacketSize0 - Max packet size to endpoint 0
                                   ;bNumPorts - Number of downstream ports on hub
                                   ;bPowerOn2PowerGood - Time in units of 2ms for device on port to
                                                                      turn on
                                   ;bRes- Endpoint address, for when we add interrupt eps
                                   ;    If bNumPorts=0 => Hub needs to undergo Hub Config
000002BA <res A0h>                 msdDevTbl        resb 10*msdDevTblEntrySize
                                   msdDevTblEnd     equ $
                                   msdDevTblE       equ     ($ - msdDevTbl)/msdDevTblEntrySize
                                   ;bAddress - The assigned device address [+ 0]
                                   ;bBus - Host Bus [Root hub] [+ 1]
                                   ;bHostHub - Address of Hub we are attached to or 0 for Root [+ 2]
                                   ;bHubPort - Port number we are inserted in  [+ 3]
                                   ;bInerfaceNumber - Interface number being used  [+ 4]
                                   ;bInterfaceSubclass - 00h (defacto SCSI), 06h (SCSI), 04h (UFI)
                                                                     [+ 5]
                                   ;bInterfaceProtocol - 50h (BBB), 00h (CBI), 01h (CBI w/o interrupt)
                                                                     [+ 6]
                                   ;bMaxPacketSize0 - Max packet size to endpoint 0
                                                                     [+ 7]
                                   ;bEndpointInAddress - 4 bit address of IN EP
                                                                     [+ 8]
                                   ;wMaxPacketSizeIn - Max packet size to chosen In endpoint
                                                                     [+ 9]
                                   ;bEndpointOutAddress - 4 bit address of OUT EP
                                                                     [+ 11]
                                   ;wMaxPacketSizeOut - Max packet size to OUT endpoint
                                                                     [+ 12]
                                   ;bInEPdt - In Endpoints' dt bit
                                                                     [+ 14]
                                   ;bOutEPdt - Out Endpoints' dt bit
                                                                     [+ 15]
                                   ;These past two bytes are temporarily kept separate! Will bitstuff
```

```
                                           later
                                ;------------------------------------------
                                ;          IDE Tables                     :
                                ;------------------------------------------
                                ;Support up to two IDE controllers
0000035A ??                     ideNumberOfControllers: resb 1
0000035B <res 20h>              ideControllerTable:     resb 2*ideTableEntrySize ;Max 2 controllers
                                ;dPCIAddress   - PCI IO address of controller   [+0]
                                ;dPCIBAR4 - PCI BAR4, the Bus Mastery address [+4]
                                ; Note that this address is given with the bottom nybble indicating
                                ; if the address is IO or MMIO. Bit set => IO
                                ;------------------------------------------
                                ;          ATA Tables                     :
                                ;------------------------------------------
0000037B <res 40h>              fdiskTable:     resb 4*fdiskTableEntrySize  ;Max 4 fixed disks
                                ; - BIOS address of device
                                ;------------------------------------------
                                ;          Int33h Table Area              :
                                ;------------------------------------------
000003BB <res A0h>              diskDevices:    resb 10*int33TblEntrySize
                                diskDevicesE    equ ($ - diskDevices)/int33TblEntrySize
                                ;bDevType - 0 = Unasigned, 1 = MSD EHCI, 2 = MSD xHCI, 3 = Floppy
                                           Physical,
                                ;           4 = ATA device, 5 = ATAPI device   [+ 0]
                                ;wDeviceAddress - USB Address/Bus pair OR local device table
                                           address  [+ 1]
                                ;dBlockSize - Dword size of LBA block (should be 512 for remdev) [+
                                           3]
                                ;qLastLBANum - Last LBA address (OS MAY minus 1 to avoid crashing
                                           device) [+ 7]
                                ;bEPSize - 1 = 64 byte, 2 = 512 byte (EP size for sector transfer)
                                           [+ 15]
                                ;NOTE: LBA SECTOR 0 IS CHS SECTOR 0,0,1 !!
                                ;------------------------------------------------------------
                                ;------------------------------------------------------------
                                ;                 MCP Transaction area                       :
                                ;------------------------------------------------------------
                                Segment MCPseg nobits follows=codeResident align=1
00000000 <res 800h>                             resb sizeOfMCPAlloc    ;2KB space
                                MCPsegEnd:  ;Pointer to the end of the segment
                                ;------------------------------------------------------------
                                ;                 BIOS Transaction area                      :
                                ;                                                            :
                                ;                  Must be the last segment                  :
                                ;------------------------------------------------------------
                                Segment xdata nobits follows=MCPseg align=40h     ;eXtra data seg
                                ;This segment comes after the resident code and is the transaction
                                ;area. The ehci async schedule (and eventually periodic) live here.
                                ;They are BOTH always postfixed by the big memory map.
                                ehciAschedule:                       ;Static label for head of the
                                           asyncschedule
00000000 <res 60h>              ehciQHead0      resb ehciSizeOfQH ;96 bytes, for address 0 device
                                           only
00000060 <res 20h>                   alignb 40h
00000080 <res 60h>              ehciQHead1      resb ehciSizeOfQH ;Used for cmds with an addressed
                                           usb device
000000E0 <res 20h>                   alignb 40h
00000100 <res 280h>             ehciTDSpace     resb 10*ehciSizeOfTD    ;640 bytes of transfer space
                                     alignb 40h
00000380 <res 20h>              ehciDataOut     resb 20h                ;32 bytes
000003A0 <res 20h>                   alignb 40h
                                sectorbuffer:                           ;Same buffer for multiple
                                           purposes
000003C0 <res 200h>             ehciDataIn      resb 200h               ;512 bytes, to get as much data
                                           as needed
                                     alignb 40h
000005C0 <res 10h>              msdCSW          resb 10h
                                ;13 bytes, special, to be saved after each transfer
000005D0 <res 10h>                   alignb 20h
000005E0 <res 10h>              prdt:           resq 2      ;2 entries in the prdt
                                bigmapptr:                            ;Pointer to big mem map
                                ;------------------------------------------------------------
                                ;                      SysInit Table                         :
                                ;------------------------------------------------------------
                                Segment SysInitParams    nobits start=600h
                                ;Use the bootsector reload space (600h-800h) as a temporary stack
                                ; and a storage space for the SysInit table
                                SysInitTable:
00000000 ????                   .numSecW        resw 1
00000002 ????????????????       .FileLBA        resq 1
0000000A ????                   loMachineWord   resw 1
                                ;------------------------------------------------------------
                                ;                      Real Mode Stack                       :
```

```
                              ;------------------------------------------------------------------
                              Segment lowStack    nobits   start=700h
00000000 <res 100h>                       resb 100h
                              lowStackPtr:
                              ;------------------------------------------------------------------
                              ORG 800h
                              ;------------------------------------------------------------------
                              ;                      INIT CODE STARTS HERE                       :
                              ;------------------------------------------------------------------
                              Segment codeInit start=BIOSInitAddr align=1
                              BITS 16
                              ;First set stack and save the SysInit Ptr, then set A20, check CPUID and
                              ; exended features. Then tell BIOS that we are going long and perhaps
                              ; protected then get the Int 11h word, store at 0:800h
                              realInit:
                              ;The Caller Far Jumps to set cs to 0
00000000 FA                       cli     ;Stop interrupts as we dont know where the stack is
00000001 31C0                     xor ax, ax
00000003 8ED8                     mov ds, ax
00000005 8ED0                     mov ss, ax
00000007 BC[0001]                 mov sp, lowStackPtr ;Set up stack pointer
0000000A FB                       sti
0000000B 26803F0C                 cmp byte [es:bx], 0Ch   ;Check length
0000000F 0F85ED00                 jne .fail   ;If thats not it, error 0
00000013 268B4701                 mov ax, word [es:bx + 1]    ;Get number of sectors into ax
00000017 B92A00                   mov cx, 42  ;42 sectors maximum
0000001A 39C8                     cmp ax, cx
0000001C 0F43C1                   cmovnb ax, cx
0000001F A3[0000]                 mov word [SysInitTable.numSecW], ax
00000022 26668B4704               mov eax, dword [es:bx + 4]       ;Get low dword
00000027 66A3[0200]               mov dword [SysInitTable.FileLBA], eax
0000002B 26668B4708               mov eax, dword [es:bx + 8]       ;Get high dword
00000030 66A3[0600]               mov dword [SysInitTable.FileLBA + 4], eax
00000034 06                       push es
                              .a20Proc:
00000035 50                       push ax
00000036 51                       push cx ;preserve ax and cx
00000037 31C9                     xor cx, cx ;clear to use as a timeout counter

                              .a20FastEnable:
00000039 E492                     in al, 92h
0000003B A802                     test al, 2
0000003D 750B                     jnz .no92
0000003F 0C02                     or al, 2
00000041 24FE                     and al, 0FEh
00000043 E692                     out 92h, al

00000045 FEC1                     inc cl     ;increments the time out counter
00000047 E94900                   jmp .a20Check

                              .no92:
0000004A B104                     mov cl, 4
0000004C E96A00                   jmp .a20Fail

                              .a20KeybEnable: ;communicating with the keyboard controller
0000004F FA                       cli

00000050 E83200                   call .a20wait
00000053 B0AD                     mov al,0ADh
00000055 E664                     out 64h,al ;disable the keyboard
00000057 E82B00                   call .a20wait
0000005A B0D0                     mov al,0D0h
0000005C E664                     out 64h,al ;read from the keyboard input
0000005E E82B00                   call .a20wait2
00000061 E460                     in al,60h
00000063 6650                     push eax     ;get the keyboard data and push it to the stack
00000065 E81D00                   call .a20wait
00000068 B0D1                     mov al,0D1h
0000006A E664                     out 64h,al     ;output the command to prep to go a20
0000006C E81600                   call .a20wait
0000006F 6658                     pop eax     ;need this be eax and not just ax?
00000071 0C02                     or  al,2
00000073 E660                     out 60h,al     ;output to go a20
00000075 E80D00                   call .a20wait
00000078 B0AE                     mov al,0AEh
0000007A E664                     out 64h,al     ;reenable keyboard
0000007C E80600                   call .a20wait     ;done!
0000007F FB                       sti

00000080 FEC1                     inc cl     ;increments the time out counter
00000082 E90E00                   jmp .a20Check
```

```

                                .a20wait:
00000085 E464                       in al,64h
00000087 A802                       test al,2
00000089 75FA                       jnz .a20wait
0000008B C3                         ret

                                .a20wait2:
0000008C E464                       in al,64h
0000008E A801                       test al,1
00000090 74FA                       jz .a20wait2
00000092 C3                         ret

                                .a20Check:
00000093 B8FFFF                     mov ax, 0FFFFh
00000096 50                         push ax
00000097 07                         pop es ;es to FFFF
00000098 BF1000                     mov di, 0010h ;FFFF:0010 == 0000:0000
0000009B 31F6                       xor si, si      ;remember ds = 0000
0000009D 268A05                     mov al, byte [es:di]
000000A0 3E3804                     cmp byte [ds:si], al
000000A3 7414                       je .a20Fail
000000A5 FEC0                       inc al      ;make change to al
000000A7 3E8804                     mov byte [ds:si], al ;al is now incremented and saved at
                                                                   address 0000:0000
000000AA 263805                     cmp byte [es:di], al ;check against overflown version
000000AD 740A                       je .a20Fail

                                .a20Pass:
000000AF FEC8                       dec al      ;return al to its original value
000000B1 3E8804                     mov byte [ds:si], al ;return to original position

000000B4 59                         pop cx
000000B5 58                         pop ax
000000B6 07                         pop es
000000B7 EB11                       jmp short .a20Exit

                                .a20Fail:
000000B9 80F903                     cmp cl, 3
000000BC 0F8E79FF                   jle .a20FastEnable
000000C0 80F906                     cmp cl, 6
000000C3 7E8A                       jle .a20KeybEnable

000000C5 59                         pop cx
000000C6 58                         pop ax
000000C7 07                         pop es
000000C8 EB2E                       jmp short .noa20

                                .a20Exit:
000000CA 669C                       pushfd
000000CC 6658                       pop eax
000000CE 6689C1                     mov ecx, eax ;save original flag state for later
000000D1 663500002000               xor eax, 00200000h ;21st bit - CPUID bit, switch it!!
000000D7 6650                       push eax
000000D9 669D                       popfd

000000DB 669C                       pushfd
000000DD 6658                       pop eax
000000DF 6685C8                     test eax, ecx ; compare the registers. If they are the same
000000E2 7416                       je .noCPUID
000000E4 6651                       push ecx
000000E6 669D                       popfd

                                .extCheck:
000000E8 66B800000080               mov eax, 80000000h
000000EE 0FA2                       cpuid
000000F0 663D01000080               cmp eax, 80000001h ;If this is true, CPU supports extended
                                                                   functionality
000000F6 733C                       jae tellBIOS
                                .noa20:
000000F8 B401                       mov ah, 1      ;noa20 error code
                                .noCPUID:
000000FA B402                       mov ah, 2      ;noCPUID error code
000000FC EB02                       jmp short .fail
000000FE B403                       mov ah, 3      ;no Extended functionality error code
                                .fail:
00000100 88E2                       mov dl, ah      ;store ax to get error code printed
00000102 BE[2801]                   mov si, .msg
00000105 E81100                     call .write
00000108 88D0                       mov al, dl
0000010A BB0700                     mov bx, 0007h       ;Attribs
0000010D B40E                       mov ah, 0Eh           ;TTY print char
0000010F 0430                       add al, 30h           ;add '0' to digit
```

```
00000111 CD10                           int 10h
00000113 31C0                           xor ax, ax
00000115 CD16                           int 16h     ;await keystroke
00000117 CD18                           int 18h
                                    ;Error codes:
                                    ;    00h - Bad SysInit Data
                                    ;    01h - No A20 Line
                                    ;    02h - No CPUID
                                    ;    03h - No Extended Functionality
                                    .write: ;destroys registers ax and bx
00000119 AC                             lodsb
0000011A 3C00                           cmp al, 0 ;check for zero
0000011C 7409                           je .return
0000011E B40E                           mov ah, 0Eh     ;TTY output
00000120 BB0700                         mov bx, 0007h ;colour
00000123 CD10                           int 10h
00000125 EBF2                           jmp short .write
                                    .return:
00000127 C3                             ret
00000128 426F6F74206572726F-        .msg: db 'Boot error:',0
00000131 723A00
                                    tellBIOS:
00000134 66B800EC0000                   mov eax, 0EC00h ;Tell BIOS we are going long
0000013A B303                           mov bl, 03h     ;Both Long and Protected modes
0000013C CD15                           int 15h         ;Ignore response
0000013E CD11                           int 11h
00000140 A3[0A00]                       mov word [loMachineWord], ax
                                    ;Getting Memory Map
                                    rmE820Map:
00000143 06                             push es
00000144 1E                             push ds
00000145 B80010                         mov ax, e820Seg
00000148 8ED8                           mov ds, ax
0000014A 8EC0                           mov es, ax
0000014C BF0200                         mov di,     e820BaseOff
0000014F 6631DB                         xor ebx, ebx
00000152 31ED                           xor bp,bp
00000154 66BA50414D53                   mov edx, 0534D4150h     ;Magic dword
0000015A 66B820E80000                   mov eax, 0E820h
00000160 2666C7451401000000             mov dword [es:di + 20], 1
00000169 66B918000000                   mov ecx, 24             ;Get 24 bytes
0000016F CD15                           int 15h
00000171 7257                           jc .mapfail             ;Carry set => Fail
00000173 66BA50414D53                   mov edx, 0534D4150h     ;Magic dword
00000179 6639D0                         cmp eax, edx        ;Must be equal on success
0000017C 754C                           jne .mapfail
0000017E 6685DB                         test ebx, ebx           ;One table entry, bad
00000181 7447                           jz .mapfail
00000183 EB1F                           jmp short .map1
                                    .map0:
00000185 66B820E80000                   mov eax, 0E820h
0000018B 2666C7451401000000             mov dword  [es:di + 20], 1
00000194 66B918000000                   mov ecx, 24
0000019A CD15                           int 15h
0000019C 722C                           jc .mapexit
0000019E 66BA50414D53                   mov edx, 0534D4150h
                                    .map1:
000001A4 E31D                           jcxz .map3
000001A6 80F914                         cmp cl, 20
000001A9 7607                           jbe .map2
000001AB 26F6451401                     test byte [es:di + 20], 1
000001B0 7411                           je .map3
                                    .map2:
000001B2 26668B4D08                     mov ecx, dword [es:di + 8]
000001B7 26660B4D0C                     or ecx, [es:di + 12]
000001BC 7405                           jz .map3
000001BE 45                             inc bp
000001BF 81C71800                       add di, 24
                                    .map3:
000001C3 6685DB                         test ebx, ebx
000001C6 75BD                           jne .map0
000001C8 EB00                           jmp short .mapexit
                                    .mapfail:
                                    .mapexit:
000001CA 26892E0000                     mov word [es:e820SizeOff], bp  ;Num entries in var space (3
                                                                         qwords/entry)
                                    ;Second memory test
000001CF 31C9                           xor cx, cx
000001D1 31D2                           xor dx, dx
000001D3 B801E8                         mov ax, 0E801h
000001D6 CD15                           int 15h
000001D8 7216                           jc .badmem2
000001DA 80FC86                         cmp ah, 86h     ;unsupported command
```

```
000001DD 7411              je .badmem2
000001DF 3D8000            cmp ax, 80h      ;invalid command
000001E2 740C              je .badmem2
                       .mem2write:
000001E4 AB                stosw
000001E5 89D8              mov ax, bx
000001E7 AB                stosw
000001E8 89C8              mov ax, cx
000001EA AB                stosw
000001EB 89D0              mov ax, dx
000001ED AB                stosw
000001EE EB0B              jmp short .mem3test
                       .badmem2:
000001F0 31C0              xor ax, ax
000001F2 31DB              xor bx, bx
000001F4 31C9              xor cx, cx
000001F6 31D2              xor dx, dx
000001F8 E9E9FF            jmp .mem2write
                       .mem3test:
000001FB F8                clc
000001FC B488              mov ah, 88h
000001FE CD15              int 15h
00000200 31DB              xor bx, bx
00000202 0F42C3            cmovc ax, bx     ;if error, store zero
00000205 3D8600            cmp ax, 86h
00000208 0F44C3            cmovz ax, bx
0000020B 3D8000            cmp ax, 80h
0000020E 0F44C3            cmovz ax, bx
00000211 AB                stosw
                       .finalmemtest:
00000212 F8                clc
00000213 CD12              int 12h
00000215 0F42C3            cmovc ax, bx     ;If carry on, store a zero
00000218 AB                stosw     ;Store the word
                       rmGetFontPointers:
                       ;Get ROM Font Pointers, immediately after Memory map
                       ;Each entry is 8 bytes long: es=Seg, bp=Off, cx=bytes/char, dx=# of rows - 1
00000219 31DB              xor bx, bx           ;Clear bh
                       .gfp1:
                       ;Over protective routine in the event that the BIOS routine clobbers registers
0000021B BE0010            mov si, 1000h     ;Save segment loader
0000021E 31C9              xor cx, cx
00000220 31D2              xor dx, dx
00000222 31ED              xor bp, bp
00000224 53                push bx              ;Save bx

00000225 B83011            mov ax, 1130h     ;Get font pointer function
00000228 CD10              int 10h

0000022A 8CC0              mov ax, es         ;Get segment into ax to store
0000022C 8EC6              mov es, si         ;Reload segment for stos to work
0000022E AB                stosw
0000022F 89E8              mov ax, bp         ;Get offset
00000231 AB                stosw
00000232 89C8              mov ax, cx         ;bytes/char
00000234 AB                stosw
00000235 88D0              mov al, dl         ;dl contains # of rows, but zero extended for alignment
00000237 30E4              xor ah, ah
00000239 AB                stosw
0000023A 5B                pop bx             ;Get the count back
0000023B FEC7              inc bh
0000023D 80FF07            cmp bh, 7
00000240 76D9              jbe .gfp1         ;Once above 7, fall through

00000242 1F                pop ds
00000243 07                pop es     ;Bring back original es value
                       rmSetTables:
                       ;Memory tables live in 0:8000h - 0:E000h range
00000244 66BF00800000      mov edi, 8000h
0000024A 0F22DF            mov cr3, edi     ;Cannot lsh cr3
0000024D B90030            mov cx, 3000h     ;6000h bytes (6x4Kb) of zero to clear table area
00000250 57                push di
00000251 31C0              xor ax, ax
00000253 F3AB              rep stosw         ;Store 3000h words of zero

00000255 5F                pop di             ;Return zero to the head of the table, at 08000h
00000256 B80390            mov ax, 9000h|permissionflags     ;9000h is the low word of the address.
```

```
00000259 AB                          stosw     ;store the low word of the address
0000025A 81C7FE0F                    add di, 0FFEh
0000025E B90400                      mov cx, 4
                                  rmUtables:               ;di should point to 8000h
00000261 050010                      add ax, 1000h
00000264 AB                          stosw     ;ax is now A003h,B003h,C003h,D003h
00000265 81C70600                    add di, 6     ;qword alignment
00000269 49                          dec cx
0000026A 75F5                        jnz rmUtables

0000026C B90008                      mov cx, 800h     ;4x512 consecutive entries
0000026F 31C0                        xor ax, ax
00000271 50                          push ax              ;push for algorithm to work
00000272 BF00A0                      mov di, 0A000h
                                  rmPDTentries:
00000275 B88300                      mov ax, 83h          ;bit 7/permission flags
00000278 AB                          stosw              ;di incremented twice
00000279 58                          pop ax              ;get current address
0000027A AB                          stosw              ;di incremented twice. store the address
0000027B 052000                      add ax, 20h          ;add the offset to the next page
0000027E 50                          push ax              ;push current address into memory
0000027F 81C70400                    add di, 4          ;qword Align
00000283 49                          dec cx
00000284 75EF                        jnz rmPDTentries

00000286 0F20E0                      mov eax, cr4
00000289 660DA0000000                or eax, 0A0h ;Set PAE and PGE, for glbl page and physical page
                                                   extensions
0000028F 0F22E0                      mov cr4, eax

00000292 66B9800000C0                mov ecx, 0C0000080h     ;Read EFER MSD into EDX:EAX
00000298 0F32                        rdmsr     ; Read information from the msr.
0000029A 660D00010000                or eax, 00000100h ; Set the Long mode bit!
000002A0 0F30                        wrmsr   ; Write the data back

000002A2 FA                          cli
000002A3 B0FF                        mov al, 0FFh              ; Out 0xFF to 0xA1 and 0x21 to disable
                                                   all IRQs.
000002A5 E6A1                        out 0A1h, al
000002A7 E621                        out 21h, al

000002A9 0F0116[DA02]                lgdt [GDT.Pointer] ;Load the Global Descriptor Table pointer

000002AE 0F20C0                      mov eax, cr0
000002B1 660D01000080                or eax, 80000001h ;Set the Paging and Protected Mode bits (Bits
                                                   31 and 0)
000002B7 0F22C0                      mov cr0, eax  ;write it back!
000002BA EA[E402]0800                jmp GDT.Code:longmode_ep

                                  GDT:                     ;Global Descriptor Table (64-bit).
                                  .Null: equ $ - GDT       ;The null descriptor.
000002BF 0000000000000000            dq 0
                                  .Code: equ $ - GDT       ;The 32-bit code descriptor. Limit =
                                                   FFFFFh, Base=0
000002C7 FFFF                        dw 0FFFFh             ;Limit 0:15
000002C9 0000                        dw 00000h             ;Base 0:15
000002CB 00                          db 00h                ;Base 16:23
000002CC 9A                          db 09Ah               ;Access Byte
000002CD 3F                          db 03Fh               ;Limit 16:19
000002CE 00                          db 00b                ;Base 24:31

                                  .Data: equ $ - GDT       ;The 32-bit data descriptor.
000002CF FFFF                        dw 0FFFFh             ;Limit 0:15
000002D1 0000                        dw 00000h             ;Base 0:15
000002D3 00                          db 0h                 ;Base 16:23
000002D4 92                          db 092h               ;Access Byte
000002D5 1F                          db 01Fh               ;Limit 16:19 then Flags
000002D6 00                          db 00h                ;Base 24:31
000002D7 90                       ALIGN 4
000002D8 0000                        dw 0
000002DA 1A00                     .Pointer     dw $ - GDT - 1        ; GDT pointer.
000002DC [BF02000000000000]       .Base        dq GDT                    ; GDT offset.
                                  ;-------------------------------------------------------------
                                  BITS 64
                                  ;-------------------------------------------------------------
                                  ;                  Long Mode Initialisation                 :
                                  ;-------------------------------------------------------------
                                  ; Sets up Segment registers, copies the resident portion of SCPBIOS
                                  ; high, initialises the BDA, copies data from real mode BIOS to
                                  ; SCPBIOS internal area, Identity maps the first 4 Gb, creates
                                  ; an IVT and moves the GDT to its final resting place,
                                  ; and directs cr3, gdtr and idtr to the BDA vars and reinits the
                                                   video
```

```
                                       ; to VGA Mode 3. Finish by printing boot message and memory sizes.
                                       ;--------------------------------------------------------------------
                                       longmode_ep:
000002E4 66B81000                          mov ax, 10h
000002E8 668ED8                            mov ds, ax
000002EB 668EC0                            mov es, ax
000002EE 668EE0                            mov fs, ax
000002F1 668EE8                            mov gs, ax
000002F4 668ED0                            mov ss, ax
                                       ;------------------Write BDA constants------------------
000002F7 48BF-                             mov rdi, section.data.start
000002F9 [0000000000000000]
00000301 66B80001                          mov ax, 100h
00000305 66AB                              stosw                ;IDT Length
00000307 66B8FF0F                          mov ax, (100h*10h) - 1      ;IDT Limit
0000030B 66AB                              stosw
0000030D 48B8-                             mov rax, BIOSIDTable      ;IDT Base
0000030F [0000000000000000]
00000317 48AB                              stosq
00000319 66B80300                          mov ax, 3h
0000031D 66AB                              stosw
0000031F 66B81700                          mov ax, (3h*8h)-1
00000323 66AB                              stosw
00000325 48B8-                             mov rax, BIOSGDTable
00000327 [0070000000000000]
0000032F 48AB                              stosq
00000331 48B8-                             mov rax, BIOSPageTbl
00000333 [0010000000000000]
0000033B 48AB                              stosq
0000033D 31C0                              xor eax, eax      ;Clears upper dword too
                                       ;Clear spur int counters
0000033F 66AB                              stosw
                                       ;Keyboard area
00000341 B904000000                        mov ecx, 4h
00000346 F348AB                            rep stosq      ;Clear kb buffer for 16 words
00000349 48B8-                             mov rax, kb_buffer
0000034B [2200000000000000]
00000353 66B90300                          mov cx, 3h      ;Circular pointers
00000357 F348AB                            rep stosq
0000035A 480520000000                      add rax, 20h      ;End of buffer pointer
00000360 48AB                              stosq
00000362 31C0                              xor eax, eax
00000364 AB                                stosd      ;Store keyboard flags bytes
                                       ;Serial Area
00000365 AA                                stosb    ;Clear number of COM devices byte
00000366 48AB                              stosq      ;Clear com_addresses (4 words)
00000368 66B90800                          mov cx, 8
0000036C F348AB                            rep stosq      ;Store 8 qwords for COM buffers
                                       ;Buffer heads
0000036F 48B8-                             mov rax, com1_buffer
00000371 [6F00000000000000]
00000379 48AB                              stosq
0000037B 480510000000                      add rax, 10h      ;Com2
00000381 48AB                              stosq
00000383 480510000000                      add rax, 10h      ;Com3
00000389 48AB                              stosq
0000038B 480510000000                      add rax, 10h      ;Com4
00000391 48AB                              stosq
                                       ;Buffer Tails
00000393 482D30000000                      sub rax, 30h
00000399 48AB                              stosq
0000039B 480510000000                      add rax, 10h      ;Com2
000003A1 48AB                              stosq
000003A3 480510000000                      add rax, 10h      ;Com3
000003A9 48AB                              stosq
000003AB 480510000000                      add rax, 10h      ;Com4
000003B1 48AB                              stosq
                                       ;Buffer start
000003B3 482D30000000                      sub rax, 30h
000003B9 48AB                              stosq
000003BB 480510000000                      add rax, 10h      ;Com2
000003C1 48AB                              stosq
000003C3 480510000000                      add rax, 10h      ;Com3
000003C9 48AB                              stosq
000003CB 480510000000                      add rax, 10h      ;Com4
000003D1 48AB                              stosq
                                       ;Buffer end
000003D3 482D20000000                      sub rax, 20h
000003D9 48AB                              stosq
000003DB 480510000000                      add rax, 10h      ;Com2
000003E1 48AB                              stosq
000003E3 480510000000                      add rax, 10h      ;Com3
000003E9 48AB                              stosq
```

```
904 000003EB 480510000000           add rax, 10h     ;Com4
905 000003F1 48AB                   stosq
906                               ;Printer area
907 000003F3 31C0                   xor eax, eax
908 000003F5 66B90300               mov cx, 3h
909 000003F9 F366AB                 rep stosw
910                               ;Timers area
911 000003FC 66AB                   stosw   ;Default pit_divisor, 0 = 65536
912 000003FE AB                     stosd    ;pit_ticks
913 000003FF 48AB                   stosq    ;rtc_ticks
914                               ;Screen area
915 00000401 66B90200               mov cx, 2h
916 00000405 F348AB                 rep stosq    ;rax, is 0
917 00000408 66B85000               mov ax, 50h
918 0000040C AA                     stosb
919 0000040D 66B81900               mov ax, 19h
920 00000411 AA                     stosb
921 00000412 6631C0                 xor ax, ax
922 00000415 66AB                   stosw
923 00000417 66B80700               mov ax, 07
924 0000041B AA                     stosb
925 0000041C 66B80300               mov ax, 03
926 00000420 AA                     stosb
927 00000421 6631C0                 xor ax, ax
928 00000424 AA                     stosb
929 00000425 66B8D403               mov ax, vga_index
930 00000429 66AB                   stosw
931 0000042B B800800B00             mov eax, vga_bpage2
932 00000430 AB                     stosd
933 00000431 31C0                   xor eax, eax     ;zero rax
934                               ;Store scr_mode_params and scr_vga_ptrs
935 00000433 B909000000             mov ecx, 9
936 00000438 F348AB                 rep stosq
937                               ;HDD/FDD data area
938 0000043B 31C0                   xor eax, eax
939 0000043D 66AB                   stosw   ;Int 33h entries and msdStatus
940 0000043F AA                     stosb   ;Fixed disk entries
941 00000440 AB                     stosd   ;Hard drive status entries
942 00000441 48B8-                  mov rax, diskdpt
942 00000443 [8D19000000000000]
943 0000044B 48AB                   stosq   ;Store the address of the default remdev format table
944 0000044D 48B8-                  mov rax, fdiskdpt
944 0000044F [9819000000000000]
945 00000457 48AB                   stosq
946 00000459 31C0                   xor eax, eax
947                               ;SysInit area
948 0000045B 488B0425[02000000]     mov rax, qword [SysInitTable.FileLBA]
949 00000463 48AB                   stosq    ;NextFileLBA
950 00000465 0FB70425[00000000]     movzx eax, word [SysInitTable.numSecW]
951 0000046D 66AB                   stosw    ;numSectors Word
952 0000046F 31C0                   xor eax, eax
953                               ;Memory Data area
954 00000471 AB                     stosd     ;0 MachineWord and convRAM
955 00000472 48AB                   stosq    ;0 userBase
956 00000474 AA                     stosb     ;0 bigmapSize
957 00000475 48AB                   stosq    ;0 srData, 4 words
958 00000477 66AB                   stosw    ;0 srData1, 1 word
959 00000479 48AB                   stosq    ;0 sysMem, 1 qword
960 0000047B AB                     stosd    ;0 scpSize, 1 dword
961                               ;MCP data area
962 0000047C 48C70425[EC010000]-    mov qword [mcpUserBase], section.MCPseg.start
962 00000484 [00000000]
963 00000488 48C70425[F4010000]-    mov qword [mcpUserRip], section.MCPseg.start + 180h
963 00000490 [80010000]
964 00000494 48C70425[FC010000]-    mov qword [mcpUserkeybf], section.MCPseg.start + 100h
964 0000049C [00010000]
965 000004A0 48C70425[0C020000]-    mov qword [mcpStackPtr], MCPsegEnd
965 000004A8 [00080000]
966 000004AC 48C70425[04020000]-    mov qword [mcpUserRaxStore], 0
966 000004B4 00000000
967 000004B8 4881C728000000         add rdi, 5*8     ;Go forwards by 5 entries
968                               ;USB Area
969 000004BF AA                     stosb
970 000004C0 66B90400               mov cx, 4
971 000004C4 F348AB                 rep stosq      ;eControllerList
972 000004C7 AA                     stosb
973 000004C8 48B8-                  mov rax, USB.ehciCriticalErrorHandler ;Get the critical error
                                                                    handler ptr
973 000004CA [A531000000000000]
974 000004D2 48AB                   stosq         ;Install eHCErrorHandler
975 000004D4 31C0                   xor eax, eax     ;Rezero rax
976 000004D6 66FFC8                 dec ax
977 000004D9 48AB                   stosq         ;eCurrAsyncHead
```

```
000004DB AA                         stosb          ;eActiveAddr
000004DC AA                         stosb           ;eActiveCtrlr
000004DD 66FFC0                     inc ax
000004E0 AB                         stosd
                                 ;USB Tables
000004E1 66B91E00                   mov cx, 10*usbDevTblEntrySize
000004E5 F3AA                       rep stosb
000004E7 66B95000                   mov cx, 10*hubDevTblEntrySize
000004EB F3AA                       rep stosb
000004ED 66B9A000                   mov cx, 10*msdDevTblEntrySize
000004F1 F3AA                       rep stosb
                                 ;IDE and Int 33h stuff
000004F3 AA                         stosb          ;ideNumberOfControllers
000004F4 66B92000                   mov cx, 2*ideTableEntrySize ;ideControllerTable
000004F8 F3AA                       rep stosb
000004FA 66B94000                   mov cx, 4*fdiskTableEntrySize
000004FE F3AA                       rep stosb
00000500 66B9A000                   mov cx, 10*int33TblEntrySize
00000504 F3AA                       rep stosb
                                 ;End of BDA variable init


                                 ;Copy the resident portion of SCPBIOS.SYS to its offset
                                 Relocate:
00000506 48BE-                      mov rsi, section.codeResident.start
00000508 [0000000000000000]
00000510 48BF-                      mov rdi, section.codeResident.vstart     ;address for the end of
                                                                   the section
00000512 [0000000000000000]
0000051A 48B9ED090000000000-        mov rcx, (residentLength/8) + 1
00000523 00
00000524 F348A5                     rep movsq     ;Copy resident portion high

                                 ;Copy machine word into var from 600h
00000527 668B0425[0A000000]         mov ax, word [loMachineWord]
0000052F 66890425[C9010000]         mov word [MachineWord], ax

                                 ;Copy Memory Maps DIRECTLY after USB dynamic space.
00000537 48BF-                      mov rdi, bigmapptr
00000539 [F005000000000000]
                                 .move820_0:      ;Add to the end
00000541 48BE00000100000000-        mov rsi, e820SizeAddr
0000054A 00
0000054B 66AD                       lodsw     ;Get number of entries for big map
0000054D 480FB6C0                   movzx rax, al     ;zero extend
00000551 488D0C40                   lea rcx, qword [rax + 2*rax]     ;Save 3*#of entries for
                                                                   countdown loop
                                 .mv0:
00000555 F348A5                     rep movsq     ;Transfer 3*al qwords
00000558 0402                       add al, 2     ;Two more entries for BIOS
0000055A 880425[D5010000]           mov byte [bigmapSize], al     ;Save entries in al
                                 ;Compute the size of BIOS allocation + space for two more entries
                                                                   up to next KB
00000561 4881C718000000             add rdi, 3*8 ;rdi now points to start of last allocated entry
                                                                   (added)
00000568 4889FB                     mov rbx, rdi
0000056B 4881C318000000             add rbx, 3*8h    ;Add size of last new entry
                                 ;Round to nearest KB
00000572 4881E300FCFFFF             and rbx, ~3FFh
00000579 4881C300040000             add rbx, 400h
00000580 48891C25[CD010000]         mov qword [userBase], rbx     ;Save userbase
00000588 4881EB00001100             sub rbx, BIOSStartAddr
0000058F 891C25[E8010000]           mov dword [scpSize], ebx     ;Save Size
                                 ;Calculate amount of system RAM available
                                 .readSystemSize:
00000596 48BB-                      mov rbx, bigmapptr
00000598 [F005000000000000]
000005A0 48BA01000000010000-        mov rdx, 0000000100000001h        ;Valid entry signature
000005A9 00
000005AA 0FB6C8                     movzx ecx, al         ;Get the number of 24 byte entries
000005AD 81E902000000               sub ecx, 2            ;Remove the allocated entries from the count
000005B3 31C0                       xor eax, eax                      ;Zero rax, use to hold
                                                                   cumulative sum
                                 .rss1:
000005B5 48395310                   cmp qword [rbx + 2*8], rdx    ;Check valid entry
000005B9 7504                       jnz .rss2
000005BB 48034308                   add rax, qword [rbx + 8]      ;Add size to rax
                                 .rss2:
000005BF 4881C318000000             add rbx, 3*8                  ;Goto next entry
000005C6 FFC9                       dec ecx                       ;Decrement count
000005C8 75EB                       jnz .rss1                     ;Not at zero, keep going
000005CA 48890425[E0010000]         mov qword [sysMem], rax
                                 ;Create and insert new entry. If no space found for new, just add
                                                                   to end
```

```
1048                                    .addEntry:
1049 000005D2 0FB60C25[D5010000]            movzx ecx, byte [bigmapSize]
1050 000005DA 81E902000000                  sub ecx, 2           ;Remove the allocated entries from the count
1051 000005E0 31D2                          xor edx, edx     ;Use as index pointer
1052                                    .ae0:
1053 000005E2 4881BA[F0050000]00-           cmp qword [bigmapptr+rdx], 100000h     ;Start of extended memory
1053 000005EA 001000
1054 000005ED 7438                          je .ae1
1055 000005EF 4881C218000000                add rdx, 18h     ;Go to next entry
1056 000005F6 FFC9                          dec ecx
1057 000005F8 75E8                          jnz .ae0
1058                                    ;If address not found, just add it to the end, deal with that here
1059                                    ;Ignore the extra calculated allocated entry
1060                                    ;rdi points to last new entry, so sub rdi to point to second to
                                                                          last entry
1061 000005FA 4881EF18000000                sub rdi, 3*8h
1062 00000601 48C70700001100                mov qword [rdi], BIOSStartAddr
1063 00000608 488B0425[E8010000]            mov rax, qword [scpSize]
1064 00000610 48894708                      mov qword [rdi + 8h], rax
1065 00000614 48B802000000010000-           mov rax, 100000002h
1065 0000061D 00
1066 0000061E 48894708                      mov qword [rdi + 8h], rax
1067 00000622 E99F000000                    jmp .altRAM
1068                                    .ae1:
1069                                    ;Address found, add new entry
1070                                    ;ecx contains number of entries that need to be shifted + 1
1071 00000627 56                            push rsi
1072 00000628 57                            push rdi
1073 00000629 4889FE                        mov rsi, rdi
1074 0000062C 4881EE30000000                sub rsi, 2*18h
1075 00000633 FFC9                          dec ecx
1076 00000635 89C8                          mov eax, ecx     ;Use eax as row counter
1077                                    .ae2:
1078 00000637 B903000000                    mov ecx, 3       ;3 8 byte entries
1079 0000063C F348A5                        rep movsq
1080 0000063F 4881EE30000000                sub rsi, 2*18h
1081 00000646 4881EF30000000                sub rdi, 2*18h
1082 0000064D FFC8                          dec eax
1083 0000064F 75E6                          jnz .ae2
1084 00000651 5F                            pop rdi
1085 00000652 5E                            pop rsi
1086                                    ;Values copied, time to change values
1087                                    ;Change HMA entry
1088 00000653 4881C2[F0050000]              add rdx, bigmapptr     ;Add offset into table to rdx
1089 0000065A 488B4A08                      mov rcx, qword [rdx + 8h]        ;Save size from entry into rax
1090 0000065E 48C7420800000100              mov qword [rdx + 8h], 10000h     ;Free 64Kb entry (HMA)
1091 00000666 4881C218000000                add rdx, 3*8h    ;Move to new SCP reserved entry
1092                                    ;Now Create the SCPBIOS Space Entry
1093 0000066D 48C70200001100                mov qword [rdx], BIOSStartAddr
1094 00000674 31DB                          xor ebx, ebx
1095 00000676 8B1C25[E8010000]              mov ebx, dword [scpSize]
1096 0000067D 48895A08                      mov qword [rdx + 8h], rbx
1097 00000681 48BB02000000010000-           mov rbx, 100000002h
1097 0000068A 00
1098 0000068B 48895A10                      mov qword [rdx + 10h], rbx  ;Reserved flags
1099 0000068F 4881C218000000                add rdx, 3*8h
1100                                    ;Now modify the Free space entry
1101 00000696 488B0425[CD010000]            mov rax, qword [userBase]
1102 0000069E 488902                        mov qword [rdx], rax
1103 000006A1 31C0                          xor eax, eax
1104 000006A3 8B0425[E8010000]              mov eax, dword [scpSize]
1105 000006AA 4829C1                        sub rcx, rax
1106 000006AD 4881E900000100                sub rcx, 10000h ;Sub HMA size
1107 000006B4 48894A08                      mov qword [rdx + 8h], rcx     ;Put entry back
1108 000006B8 48BB01000000010000-           mov rbx, 100000001h
1108 000006C1 00
1109 000006C2 48895A10                      mov qword [rdx + 10h], rbx  ;Free flags
1110                                    .altRAM:
1111                                    ;Copy Alt RAM values
1112 000006C6 8B1C25[E8010000]              mov ebx, dword [scpSize]
1113 000006CD C1EB0A                        shr ebx, 0Ah     ;Rescale from byts to KB
1114 000006D0 81C340000000                  add ebx, 40h     ;Add the HMA (64Kb)
1115 000006D6 48BF-                         mov rdi, srData     ;Save qword in srData ah=E801h
1115 000006D8 [D601000000000000]
1116 000006E0 48AD                          lodsq     ;Get into rax, inc rsi
1117 000006E2 6629D8                        sub ax, bx       ;bx preserved, contains number of KB's plus 1
1118 000006E5 48C1C820                      ror rax, 20h     ;Rotate over 32 bits
1119 000006E9 6629D8                        sub ax, bx
1120 000006EC 48C1C820                      ror rax, 20h     ;Rotate over 32 bits again
1121 000006F0 48AB                          stosq                ;Save, inc rdi
1122 000006F2 48BF-                         mov rdi, srData1     ;Save word for ah=88h
1122 000006F4 [DE01000000000000]
1123 000006FC 66A5                          movsw     ;Save value, then reduce by BIOS size
```

```
000006FE 66295FFE                   sub word [rdi - 2], bx      ;Reduce the size of the previous
                                                                 stored val
00000702 48BF-                      mov rdi, convRAM       ;Int 12h value
00000704 [CB01000000000000]
0000070C 66A5                       movsw
                                  ;Copy VGA fonts to Internal Int 30h area
0000070E 48BF-                      mov rdi, scr_vga_ptrs
00000710 [6801000000000000]
00000718 48B908000000000000-        mov rcx, 8
00000721 00
00000722 F348A5                     rep movsq
                                  ;————————————Write Long Mode Page Tables————————————
                                  ;Creates a 4Gb ID mapped page
00000725 48BF-                      mov rdi, BIOSPageTbl
00000727 [0010000000000000]
0000072F 57                         push rdi
                                  Ptablefinal:
00000730 48B9000C0000000000-        mov rcx, 6000h/8;6000h bytes (6x4Kb) of zero to clear table area
00000739 00
0000073A 57                         push rdi
0000073B 4831C0                     xor rax, rax
0000073E F348AB                     rep stosq            ;Clear the space

00000741 5F                         pop rdi                ;Return zero to the head of the table, at
                                                                 08000h
00000742 4889F8                     mov rax, rdi     ;Load rax with the PML4 table location
00000745 480500100000               add rax, 1000h  ;Move rax to point to PDPT
0000074B 480D03000000               or rax, permissionflags     ;Write the PDPT entry as present and
                                                                 r/w
00000751 48AB                       stosq     ;store the low word of the address
00000753 4881C7F80F0000             add rdi, 0FF8h
0000075A B904000000                 mov ecx, 4
                                  .utables:
0000075F 480500100000               add rax, 1000h  ;Write four entries in PDPT for each GB range
00000765 48AB                       stosq
00000767 FFC9                       dec ecx
00000769 75F4                       jnz .utables

0000076B 4881C7E00F0000             add rdi, 0FE0h  ;rdi points to the new page tables, copy!
00000772 48BE00A0000000000000-      mov rsi, 0A000h ;Get the first Page table
0000077B 00
0000077C B900080000                 mov ecx, 4000h/8  ;Number of bytes to copy
00000781 F348A5                     rep movsq          ;Get the 4Gb tables into place
00000784 5F                         pop rdi                ;Bring back Table base
00000785 0F22DF                     mov cr3, rdi      ;Finalise change in paging address


                                  ;——————————————Write Interrupts——————————————
00000788 48B900010000000000-        mov rcx, 0100h      ;256 entries
00000791 00
00000792 48B8-                      mov rax, dummy_return_64
00000794 [314F000000000000]
0000079C BB08000000                 mov ebx, codedescriptor
000007A1 31F6                       xor esi, esi
000007A3 66BA008F                   mov dx, 8F00h
                                    ;Toggle attribs. 8F = Interrupt Present, accessable from ring 0
                                                                 and greater,
                                    ;0 (so collectively 08h) and gate type 0Fh (64-bit trap gate
                                                                 (gate which
                                    ;leaves interrupts on))
                                  idtFillDummy:
000007A7 E8(CB000000)               call idtWriteEntry
000007AC 66FFC9                     dec cx
000007AF 75F6                       jnz idtFillDummy

000007B1 31F6                       xor esi, esi
000007B3 48B940000000000000-        mov rcx, ((IDT_TABLE_Length >> 3))
000007BC 00
000007BD 48BD-                      mov rbp, IDT_TABLE
000007BF [2F18000000000000]
                                  idtLoop:
000007C7 488B44F500                 mov rax, qword [rbp+(rsi*8)]
000007CC E8(CB000000)               call idtWriteEntry
000007D1 48FFC9                     dec rcx
000007D4 75F1                       jnz idtLoop

000007D6 48BC00000800000000-        mov rsp, 80000h       ;Realign stack pointer
000007DF 00
                                  ;Reload the interrupt table
000007E0 0F011C25[02000000]         lidt [IDTpointer]
                                  ;Write GDT to its final High location
000007E8 48BE-                      mov rsi, GDT
000007EA [BF02000000000000]
000007F2 48BF-                      mov rdi, BIOSGDTable
```

```
000007F4 [0070000000000000]
000007FC 48B903000000000000–            mov rcx, 3
00000805 00
00000806 F348A5                         rep movsq      ;copy the three descriptors high
                                    ;Reload the GDT Pointer
00000809 0F011425[0E000000]             lgdt [GDTpointer]

                                    ;Video Initialisation: VGA mode, CRTC at 3D4h, Mode 03h, 128k VRAM
                                    ;For now, only unlock upper WO CRTC registers, by using undocumented
                                    ; CRTC register 11h.
00000811 668B1425[5A010000]             mov dx, word [scr__crtc__base]      ;Get current set CRTC index register
00000819 B011                           mov al, 11h       ;Register 11
0000081B 88D8                           mov al, bl
0000081D EE                             out dx, al
0000081E E680                           out waitp, al     ;Wait an I/O cycle
00000820 66FFC2                         inc dx  ;Point to data register
00000823 EC                             in al, dx    ;get register 11h
00000824 247F                           and al, 7Fh ;Clear upper bit
00000826 86C3                           xchg al, bl ;Get address back into al, save new register value in bl
00000828 66FFCA                         dec dx  ;Return to index
0000082B EE                             out dx, al
0000082C FEC2                           inc dl
0000082E 86C3                           xchg al, bl
00000830 EE                             out dx, al  ;Output new byte, unlock upper WO CRTC registers for use!
                                    ;Boot message/Verification of successful VGA card reset!
                                    ;Print Boot Message
00000831 66B80413                       mov ax, 1304h
00000835 48BD–                          mov rbp, startboot
00000837 [8415000000000000]
0000083F CD30                           int 30h

00000841 E8170E0000                     call memprint      ;Print Memory status

                                    ;-----------------------------------------------------------------
                                    ;                      End of Initialisation                     :
                                    ;-----------------------------------------------------------------
                                    ;-----------------------------------------------------------------
                                    ;                  PIC Initialisation procedure                  :
                                    ;-----------------------------------------------------------------
                                    ;Remapping the IO ports to Interrupt 0x40
                                    PICremap:
00000846 B011                           mov al, 11h          ;bit 10h and 1h = Start initialisation
00000848 E620                           out pic1command, al
0000084A E680                           out waitp, al
0000084C E6A0                           out pic2command, al
0000084E E680                           out waitp, al

00000850 B020                           mov al, 20h         ;PIC1 to take Int 20h - 27h
00000852 E621                           out pic1data, al
00000854 E680                           out waitp, al
00000856 0408                           add al, 8           ;PIC2 to take Int 28h - 2Fh
00000858 E6A1                           out pic2data, al
0000085A E680                           out waitp, al

0000085C B004                           mov al, 4
0000085E E621                           out pic1data, al      ;Tell PIC 1 that there is a PIC 2 at IRQ2 (00000100)
00000860 E680                           out waitp, al
00000862 FEC8                           dec al
00000864 FEC8                           dec al
00000866 E6A1                           out pic2data, al      ;Tell PIC 2 its cascade identity (00000010)
00000868 E680                           out waitp, al

0000086A B001                           mov al, 01h          ;Initialise in 8086 mode
0000086C E621                           out pic1data, al
0000086E E680                           out waitp, al
00000870 E6A1                           out pic2data, al
00000872 E680                           out waitp, al

00000874 B0FF                           mov al, 0FFh      ;Mask all interrupts
00000876 E621                           out pic1data, al
00000878 E6A1                           out pic2data, al

                                    ;Ensure that interrupts are still masked
                                    ;-----------------------------------------------------------------
                                    ;                      End of Initialisation                     :
                                    ;-----------------------------------------------------------------
                                    ;-----------------------------------------------------------------
                                    ;                         PCI Enumeration                        :
                                    ;-----------------------------------------------------------------
```

```
                                   ; This proc enumerates only the PCI devices we care for
                                   ;————————————————————————————————————————————————————————
0000087A 4831ED                        xor rbp, rbp
0000087D 66892C2500E00000              mov word [lousbtablesize], bp
00000885 4889E9                        mov rcx, rbp     ;reset cx now too, for below
                                   pci_scan:    ;Enumerate PCI devices (formerly, USB devices)
00000888 4831DB                        xor rbx, rbx        ;Used to save the value of eax temporarily
0000088B 48B808000081000000-           mov rax, 81000008h ;Set bit 31 and lower byte to 2, for
                                                               register 2/offset 8
00000894 00
                                                          ;also make it the largest register so that
                                                               we enumerate
                                                          ;backwards and set up USB controllers in
                                                               order from
                                                          ;newest to oldest.
                                   .u1:
00000895 2D00010000                    sub eax, 100h       ;mov eax into valid PCI range, go to next
                                                               device
0000089A 66BAF80C                      mov dx, pci_index  ;PCI index register
0000089E EF                            out dx, eax    ;output the next packed
                                                               bus,device,function,register combo

0000089F 89C3                          mov ebx, eax        ;save to be used later, to access PCI BARS

000008A1 66BAFC0C                      mov dx, pci_data   ;PCI data register
000008A5 ED                            in eax, dx     ;Get Class, subclass and interface value in upper
                                                               three bytes

000008A6 C1E808                        shr eax, 8                  ;shift down the details by a byte
                                   ;IF any of these are satisfied, remember ebx has the device index
000008A9 3D00030C00                    cmp eax, ((usb_class << 16) +(usb_subclass << 8)+uhci_interface)
000008AE 0F8446010000                  je .uhci_found
000008B4 3D10030C00                    cmp eax, ((usb_class << 16) +(usb_subclass << 8)+ohci_interface)
000008B9 0F847D010000                  je .ohci_found
000008BF 3D20030C00                    cmp eax, ((usb_class << 16) +(usb_subclass << 8)+ehci_interface)
000008C4 0F847C010000                  je .ehci_found
000008CA 3D30030C00                    cmp eax, ((usb_class << 16) +(usb_subclass << 8)+xhci_interface)
000008CF 0F84D0010000                  je .xhci_found
000008D5 50                            push rax
000008D6 C1E808                        shr eax, 8                 ;roll over rid of function number
000008D9 3D01010000                    cmp eax, (msd_class << 8) + (ide_subclass)
000008DE 7452                          je .idePCIEnum
000008E0 3D06010000                    cmp eax, (msd_class << 8) + (sata_subclass)
000008E5 7414                          je .sataPCIEnum
000008E7 58                            pop rax
                                   .u11:    ;After a device found, jump here to continue enumeration
000008E8 6681E50F00                    and bp, 000Fh        ;Zero the upper nybble again.
000008ED 89D8                          mov eax, ebx         ;Return pci value into eax
000008EF 3D08000080                    cmp eax, 80000008h  ;The lowest value
000008F4 7F9F                          jg .u1
000008F6 E9A9040000                    jmp pciExit
                                   .sataPCIEnum:
000008FB 58                            pop rax
000008FC 50                            push rax
000008FD 55                            push rbp
000008FE 66B80413                      mov ax, 1304h
00000902 48BD-                         mov rbp, .spemsg
00000904 [1509000000000000]
0000090C CD30                          int 30h
0000090E 5D                            pop rbp
0000090F 58                            pop rax
00000910 E9D3FFFFFF                    jmp .u11
00000915 0A0D41484349205341-       .spemsg: db 0Ah, 0Dh, "AHCI SATA controller found", 0
0000091E 544120636F6E74726F-
00000927 6C6C657220666F756E-
00000930 6400

                                   .idePCIEnum:
00000932 58                            pop rax
00000933 50                            push rax
00000934 55                            push rbp
00000935 66B80413                      mov ax, 1304h
00000939 48BD-                         mov rbp, .ipemsg
0000093B [A809000000000000]
00000943 CD30                          int 30h
00000945 5D                            pop rbp
00000946 58                            pop rax
00000947 50                            push rax
00000948 B404                          mov ah, 04h
0000094A CD30                          int 30h
0000094C 58                            pop rax
                                   ;If function is 80h, then it will respond to default IO addresses
0000094D A880                          test al, 80h ;Check if bus mastery is enabled. Only support DMA
                                                               transfers
```

```
1336 0000094F 7497                      jz .u11        ;Exit if not enabled
1337 00000951 3C80                      cmp al, 80h   ;If 80h, device hardwired bus master legacy mode,
                                                                  all good.
1338 00000953 742B                      je .ipeWriteTable
1339                                  ;Bit bash, and reread, if it works, yay, if not, fail cancel
1340 00000955 66BAF80C                  mov dx, pci_index
1341 00000959 89D8                      mov eax, ebx
1342 0000095B EF                        out dx, eax       ;Register offset 8
1343 0000095C 6681C20400                add dx, 4         ;Point to pci_data
1344 00000961 25FFFAFFFF                and eax, 0FFFFFAFFh      ;Zero bits 0 and 2 of nybble 3
1345 00000966 EF                        out dx, eax
1346 00000967 6681EA0400                sub dx, 4
1347 0000096C 89D8                      mov eax, ebx
1348 0000096E EF                        out dx, eax
1349 0000096F 6681C20400                add dx, 4
1350 00000974 ED                        in eax, dx
1351 00000975 A900050000                test eax, 00000500h  ;Test bits 0 and 2 of nybble 3 have been
                                                                  zeroed
1352 0000097A 0F8568FFFFFF              jnz .u11      ;IF not, fail
1353                                  .ipeWriteTable:
1354                                  ;Now the controller and devices have been set to legacy, they should
1355                                  ; respond to the default IO addresses and IRQ. Save BAR 5 for Bus
                                                                  mastering.
1356 00000980 50                        push rax
1357 00000981 55                        push rbp
1358 00000982 48BD-                     mov rbp, .ipemsg2
1358 00000984 [CB09000000000000]
1359 0000098C 66B80413                  mov ax, 1304h
1360 00000990 CD30                      int 30h
1361 00000992 5D                        pop rbp
1362 00000993 58                        pop rax
1363 00000994 89D8                      mov eax, ebx
1364 00000996 B020                      mov al, 20h ;BAR4 Address
1365 00000998 66BAF80C                  mov dx, pci_index
1366 0000099C EF                        out dx, eax
1367 0000099D 6681C20400                add dx, 4
1368 000009A2 ED                        in eax, dx  ;Get BAR 4 address
1369                                    ;call IDE.addControllerTable
1370                                    ;jc .u11       ;If this fails, exit gracefully
1371 000009A3 E940FFFFFF                jmp .u11
1372 000009A8 0A0D49444520415441-     .ipemsg:     db 0Ah, 0Dh,"IDE ATA Controller found. Type: ", 0
1372 000009B1 20436F6E74726F6C6C-
1372 000009BA 657220666F756E642E-
1372 000009C3 20547970653A2000
1373 000009CB 0A0D49444520415441-     .ipemsg2:    db 0Ah, 0Dh, "IDE ATA Controller set to compatibility
                                                                  mode",0
1373 000009D4 20436F6E74726F6C6C-
1373 000009DD 65722073657420746F-
1373 000009E6 20636F6D7061746962-
1373 000009EF 696C697479206D6F64-
1373 000009F8 6500
1374                                  ;bp lo = status register,
1375                                  ;bp hi = controller being serviced (ie 1000xxxx => xHCI being
                                                                  serviced)
1376                                  .uhci_found:
1377 000009FA 6681CD1100                or bp, 00010001b      ;set bit 0/mask = 1
1378 000009FF 55                        push rbp
1379 00000A00 50                        push rax
1380 00000A01 53                        push rbx
1381 00000A02 66B80413                  mov ax, 1304h
1382 00000A06 30FF                      xor bh, bh
1383 00000A08 48BD-                     mov rbp, .uhci_succ
1383 00000A0A [1C0A000000000000]
1384 00000A12 CD30                      int 30h
1385 00000A14 5B                        pop rbx
1386 00000A15 58                        pop rax
1387 00000A16 5D                        pop rbp
1388 00000A17 E9A6000000                jmp .controlController
1389 00000A1C 0A0D5548434920636F-     .uhci_succ:     db     0Ah, 0Dh, 'UHCI controller found on IRQ ', 0
1389 00000A25 6E74726F6C6C657220-
1389 00000A2E 666F756E64206F6E20-
1389 00000A37 4952512000
1390                                  .ohci_found:
1391 00000A3C 6681CD2200                or bp, 00100010b      ;set bit 1/mask = 2
1392 00000A41 E9A2FEFFFF                jmp .u11
1393                                  .ehci_found:
1394 00000A46 6681CD4400                or bp, 01000100b      ;set bit 2/mask = 4
1395 00000A4B 55                        push rbp
1396 00000A4C 50                        push rax
1397 00000A4D 53                        push rbx
1398 00000A4E 66B80413                  mov ax, 1304h
1399 00000A52 30FF                      xor bh, bh
1400 00000A54 48BD-                     mov rbp, .ehci_succ
```

```
1400 00000A56 [650A000000000000]
1401 00000A5E CD30                        int 30h
1402 00000A60 5B                          pop rbx
1403 00000A61 58                          pop rax
1404 00000A62 5D                          pop rbp
1405 00000A63 EB5D                        jmp short .controlController
1406 00000A65 0A0D4548434920636F-     .ehci_succ:    db    0Ah, 0Dh, 'EHCI controller found on IRQ ', 0
1406 00000A6E 6E74726F6C6C657220-
1406 00000A77 666F756E64206F6E20-
1406 00000A80 4952512000
1407 00000A85 0A0D7848434920636F-     .xhci_succ:    db    0Ah, 0Dh, 'xHCI controller found on IRQ ', 0
1407 00000A8E 6E74726F6C6C657220-
1407 00000A97 666F756E64206F6E20-
1407 00000AA0 4952512000
1408                                  .xhci_found:
1409 00000AA5 55                          push rbp
1410 00000AA6 50                          push rax
1411 00000AA7 53                          push rbx
1412 00000AA8 66B80413                    mov ax, 1304h
1413 00000AAC 30FF                        xor bh, bh
1414 00000AAE 48BD-                       mov rbp, .xhci_succ
1414 00000AB0 [850A000000000000]
1415 00000AB8 CD30                        int 30h
1416 00000ABA 5B                          pop rbx
1417 00000ABB 58                          pop rax
1418 00000ABC 5D                          pop rbp
1419 00000ABD 6681CD8800                  or bp, 10001000b     ;set bit 3/mask = 8
1420
1421                                  .controlController:
1422                                  ;This for now will get the IRQ line for all controllers,
1423                                  ;and install a USB handler there, then disabling the HC rather than
                                                                just the
1424                                  ;legacy support.
1425                                  ;EAX doesnt need to be saved since the first instruction of .u11 is
                                                                to move the
1426                                  ;value of ebx back into eax.
1427                                  ;EDX doesnt need to be saved since the port data gets loaded in the
                                                                proc above
1428                                  ;DO NOT MODIFY EBX
1429 00000AC2 31D2                        xor edx, edx
1430 00000AC4 89D8                        mov eax, ebx     ;Move a copy of ebx, the PCI config space
                                                                device address
1431 00000AC6 B03C                        mov al, 3Ch      ;offset 3C has interrupt masks in lower word
1432 00000AC8 66BAF80C                    mov dx, pci_index
1433 00000ACC EF                          out dx, eax        ;set to give interrupt masks
1434 00000ACD 66BAFC0C                    mov dx, pci_data
1435 00000AD1 ED                          in eax, dx         ;Get info into eax (formally, al)
1436 00000AD2 50                          push rax
1437 00000AD3 240F                        and al, 0Fh
1438 00000AD5 B404                        mov ah, 04h
1439 00000AD7 CD30                        int 30h
1440 00000AD9 58                          pop rax
1441 00000ADA 66F7C54000                  test bp, 40h       ;Check if EHCI
1442 00000ADF 0F8490000000                jz .cc1            ;Skip mapping
1443 00000AE5 240F                        and al, 0Fh        ;Clear upper nybble for good measure
1444 00000AE7 3C10                        cmp al, 10h
1445 00000AE9 0F8786000000                ja .cc1            ;Cant map it
1446 00000AEF 3C08                        cmp al, 08h
1447 00000AF1 733E                        jae .cc0
1448 00000AF3 56                          push rsi
1449 00000AF4 52                          push rdx
1450 00000AF5 50                          push rax
1451 00000AF6 53                          push rbx
1452 00000AF7 480FB6F0                    movzx rsi, al
1453 00000AFB 81C620000000                add esi, 20h
1454 00000B01 66BA008F                    mov dx, 8F00h
1455 00000B05 48B8-                       mov rax, ehci_IRQ.pic1      ;PIC1 ep
1455 00000B07 [500B000000000000]
1456 00000B0F BB08000000                  mov ebx, codedescriptor
1457 00000B14 E8(CB000000)                call idtWriteEntry
1458 00000B19 5B                          pop rbx
1459 00000B1A 58                          pop rax
1460 00000B1B 5A                          pop rdx
1461 00000B1C 5E                          pop rsi
1462 00000B1D 51                          push rcx
1463 00000B1E 88C1                        mov cl, al
1464 00000B20 B001                        mov al, 1
1465 00000B22 D2E0                        shl al, cl           ;Shift bit to appropriate position
1466 00000B24 F6D0                        not al               ;Turn into a bitmask
1467 00000B26 88C4                        mov ah, al           ;Save in ah
1468 00000B28 E421                        in al, pic1data
1469 00000B2A 20E0                        and al, ah           ;Add bitmask to current mask
1470 00000B2C E621                        out pic1data, al     ;Unmask this line
```

```
1471 00000B2E 59                         pop rcx
1472 00000B2F EB44                       jmp short .cc1
1473                                  .cc0:
1474 00000B31 56                         push rsi
1475 00000B32 52                         push rdx
1476 00000B33 50                         push rax
1477 00000B34 53                         push rbx
1478 00000B35 480FB6F0                   movzx rsi, al
1479 00000B39 81C620000000               add esi, 20h     ;Start of PIC range
1480 00000B3F 66BA008F                   mov dx, 8F00h
1481 00000B43 48B8-                      mov rax, ehci_IRQ
1481 00000B45 [490B000000000000]
1482 00000B4D BB08000000                 mov ebx, codedescriptor
1483 00000B52 E8(CB000000)               call idtWriteEntry
1484 00000B57 5B                         pop rbx
1485 00000B58 58                         pop rax
1486 00000B59 5A                         pop rdx
1487 00000B5A 5E                         pop rsi
1488 00000B5B 51                         push rcx
1489 00000B5C 2C08                       sub al, 8
1490 00000B5E 88C1                       mov cl, al
1491 00000B60 E421                       in al, pic1data
1492 00000B62 24FB                       and al, 0FBh  ;Clear Cascade bit
1493 00000B64 E621                       out pic1data, al
1494 00000B66 B001                       mov al, 1
1495 00000B68 D2E0                       shl al, cl     ;Shift bit to appropriate position
1496 00000B6A F6D0                       not al         ;Turn into a bitmask
1497 00000B6C 88C4                       mov ah, al     ;Save in ah
1498 00000B6E E4A1                       in al, pic2data
1499 00000B70 20E0                       and al, ah     ;Add bitmask to current mask
1500 00000B72 E6A1                       out pic2data, al     ;Unmask this line
1501 00000B74 59                         pop rcx
1502                                  .cc1:
1503 00000B75 89D8                       mov eax, ebx     ;Bring back a copy of ebx, the PCI config space
                                                                 addr to eax
1504 00000B77 B010                       mov al, 10h      ;Change the register from Class code to BAR0
1505
1506 00000B79 66BAF80C                   mov dx, pci_index
1507 00000B7D EF                         out dx, eax          ;Set to give BAR0
1508 00000B7E 66BAFC0C                   mov dx, pci_data
1509 00000B82 ED                         in eax, dx          ;get unrefined BAR0/BASE pointer into eax
1510
1511 00000B83 2500FFFFFF                 and eax, 0FFFFFF00h     ;refine eax into an mmio register
1512 00000B88 50                         push rax     ;push BASE pointer onto stack
1513
1514                                  ;Write USB controller table:
1515                                  ;Each table entry (tword), as follows:
1516                                  ;Offset:
1517                                  ; 00h - hci type (bp) [word]
1518                                  ; 02h - PCI address (ebx) [dword]
1519                                  ; 06h - MMIO address (eax) [dword]
1520                                  ;ALL REGISTERS PRESERVED, data stored at usbtablebase, size at
                                                                 usbtablesize
1521 00000B89 56                         push rsi
1522 00000B8A 51                         push rcx
1523 00000B8B 0FB70C2500E00000           movzx ecx, word [lousbtablesize]     ;get number of table entries
1524 00000B93 89CE                       mov esi, ecx
1525 00000B95 D1E1                       shl ecx, 1     ;Multiply by 2
1526 00000B97 678DB4F102E00000           lea esi, [8*esi + ecx + lousbtablebase]
1527                                     ;multiply esi by 10 to get table offset & add to table base
1528                                     ;store table offset back in esi
1529 00000B9F 6667892E                   mov word [esi], bp     ;Store controller type
1530 00000BA3 81C602000000               add esi, 2
1531 00000BA9 67891E                     mov dword [esi], ebx
1532                                         ;Store PCI device config space address (set to register 2)
1533 00000BAC 81C604000000               add esi, 4
1534 00000BB2 678906                     mov dword [esi], eax     ;Store device MMIO Address (refined
                                                                 BAR0 value)
1535 00000BB5 59                         pop rcx
1536 00000BB6 5E                         pop rsi
1537 00000BB7 66FF042500E00000           inc word [lousbtablesize]
1538
1539 00000BBF 6681FD8000                 cmp bp, 80h     ;Are we servicing xHCI, EHCI or UHCI?
1540 00000BC4 7D7A                       jge .controlxHCI
1541 00000BC6 6681FD4000                 cmp bp, 40h     ;Are we servicing EHCI or UHCI?
1542 00000BCB 0F8DC9000000               jge .controlEHCI
1543                                  ;If neither of these, collapse into UHCI
1544                                  .controlUHCI:
1545                                  ;eax points to the refined base pointer
1546 00000BD1 53                         push rbx                       ;temp stack save
1547 00000BD2 89D8                       mov eax, ebx     ;get the current packed
                                                                 bus,device,function,register combo
1548 00000BD4 2500F8FFFF                 and eax, 0FFFFF800h           ;Clear bottom 10 bytes.
```

```
00000BD9 0DC0020000              or eax, 2C0h               ;Function 2, register offset C0h

00000BDE 50                      push rax                   ;temp save address value on stack

00000BDF 66BAF80C                mov dx, pci_index
00000BE3 EF                      out dx, eax
00000BE4 80C204                  add dl, 4                  ;dx now points to pci_index
00000BE7 ED                      in eax, dx                 ;Bring register value into eax

00000BE8 66B8008F                mov ax, 8F00h              ;Clear all SMI bits (no SMI pls)
00000BEC 89C3                    mov ebx, eax               ;save temporarily in ebx

00000BEE 58                      pop rax                    ;bring back address value from stack

00000BEF 80EA04                  sub dl, 4                  ;put dx back to pci_index
00000BF2 EF                      out dx, eax                ;select legsup register

00000BF3 80C204                  add dl, 4                  ;aim dx back to pci_data
00000BF6 89D8                    mov eax, ebx               ;bring back new legsup value
00000BF8 EF                      out dx, eax                ;send it back!

                             ;Now set bit 6 of the command register to 1 (semaphore)
00000BF9 5B                      pop rbx                    ;Return original ebx value
00000BFA 89D8                    mov eax, ebx   ;Move a copy of ebx, PCI config space device
                                                            address (index)
00000BFC B020                    mov al, 20h                ;Change the register from Class
                                                            code to BAR4
00000BFE 6681EA0400              sub dx, 4                  ;Point dx back to pci_index
00000C03 EF                      out dx, eax                ;Get the data we want!
00000C04 6681C20400              add dx, 4
00000C09 ED                      in eax, dx             ;Bring the value of BAR4 into eax, to
                                                            add to BASE
00000C0A 25FCFFFFFF              and eax, 0FFFFFFFCh        ;Refine the IO address that we got
00000C0F 6689C2                  mov dx, ax                 ;Mov the base IO address into dx
                             ;dx contains the base io address!
00000C12 66B80200                mov ax, 0002h              ;Reset the HC
00000C16 66EF                    out dx, ax
00000C18 51                      push rcx

                             .cu0:
00000C19 4831C9                  xor rcx, rcx
00000C1C FEC9                    dec cl

                             .cu1:
00000C1E E2FE                    loop .cu1     ;wait

00000C20 66ED                    in ax, dx     ;Bring value in
00000C22 66250200                and ax, 0002h
00000C26 75F1                    jnz .cu0      ;Reset still in progress, loop again
00000C28 59                      pop rcx

00000C29 6631C0                  xor ax, ax
00000C2C 6681C20400              add dx, 4    ;point to USBINTR
00000C31 66EF                    out dx, ax
00000C33 6681EA0400              sub dx, 4    ;return to cmd
00000C38 66EF                    out dx, ax   ;zero everything.

00000C3A 58                      pop rax      ;Get BASE (dereferenced BAR0) value back (stack
                                                            align)
00000C3B E9A8FCFFFF              jmp .u11                     ;return
                             ;End UHCI


                             .controlxHCI:
                             ;mov HCCPARAMS1 into edx, eax contains BASE pointer from BAR0
                                                            (offset 10h for
                             ; register)
00000C40 678B5010                mov edx, dword [eax + 10h]
00000C44 81E20000FFFF            and edx, 0FFFF0000h
                             ;mov hi word into lo word and shl by 2 to adjust that we are in
                                                            units of DWORDS
00000C4A C1EA0E                  shr edx, 0Eh
00000C4D 01D0                    add eax, edx               ;add offset from base onto base
                                                            ;eax now pointing at USBLEGSUP
                             .suohoc0:
00000C4F 678B10                  mov edx, dword [eax]       ;store upper byte of USBLEGSUP into dl
00000C52 81CA00000001            or edx, (1<<24)            ;Set the HCOSSEM Semaphore
00000C58 678910                  mov dword [eax], edx       ;replace the upper byte with HCOSSEM set

00000C5B 51                      push rcx                   ;push poll counter
00000C5C 4831C9                  xor rcx, rcx
                             .suohoc1:     ;Remove control from BIOS and check for confirmation
00000C5F 66FFC9                  dec cx                     ;drop counter by one
00000C62 0F84DE000000            jz .weirdEHCI1             ;temporary label
00000C68 F390                    pause                      ;wait
00000C6A 678B10                  mov edx, dword [eax]       ;Check if owned by BIOS
```

```
00000C6D 81E200000100               and edx, (1<<16)
00000C73 75EA                       jnz .suohoc1              ;not zero, keep polling

00000C75 66B9FFFF                   mov cx, 0FFFFh
                                  .suohoc2:   ;Check if control to OS has been given
00000C79 66FFC9                     dec cx
00000C7C 740D                       jz .suohoc21              ;timeout, assume it has.
00000C7E F390                       pause
00000C80 678B10                     mov edx, dword [eax]
00000C83 81E200000001               and edx, (1<<24)
00000C89 74EE                       jz .suohoc2               ;if zero, keep polling until bit set =>
                                                                  owned by OS
                                  .suohoc21:   ;Check for legsup being present, assume for now.
00000C8B 59                         pop rcx                   ;return poll counter
                                  .suohoc3:
00000C8C 67C7400400000000           mov dword [eax + 4], 0    ;Set all SMI bytes to 0 so no SMIs
                                                                  will be set.
00000C94 58                         pop rax                   ;Bring back BAR0 into eax
00000C95 E94EFCFFFF                 jmp .u11                  ;return


                                  .controlEHCI:
00000C9A 678B5008                   mov edx, dword [eax + 8h]
00000C9E 81E200FF0000               and edx, 0000FF00h
00000CA4 66C1EA08                   shr dx, 8
00000CA8 81FA40000000               cmp edx, 40h
00000CAE 7C05                       jl .ce0            ;No EECP pointer present, skip BIOS/OS EHCI
                                                                  handover
00000CB0 E81B000000                 call .ehcieecpsetup
                                  .ce0:
00000CB5 31D2                       xor edx, edx       ;clear edx
00000CB7 58                         pop rax            ;Bring back refined base into eax
00000CB8 678B10                     mov edx, dword [eax]
00000CBB 81E2FF000000               and edx, 000000FFh
00000CC1 01D0                       add eax, edx
00000CC3 67816040FEFFFFFF           and dword [eax + 40h], 0FFFFFFFEh
                                                              ;located at offset 40 of the opregs.

00000CCB E918FCFFFF                 jmp .u11                  ;return
                                  .ehcieecpsetup:
                                  ;eax has hccparams
                                  ;ebx has pci register, to get class code
00000CD0 50                         push rax
00000CD1 52                         push rdx
00000CD2 53                         push rbx
00000CD3 51                         push rcx
00000CD4 88D3                       mov bl, dl         ;Move EECP pointer into low byte of PCI address
00000CD6 89D8                       mov eax, ebx       ;Move this address to eax
00000CD8 66BAF80C                   mov dx, pci_index
00000CDC EF                         out dx, eax        ;Return EHCI EECP register
00000CDD 66BAFC0C                   mov dx, pci_data
00000CE1 ED                         in eax, dx         ;Get this register into eax
00000CE2 0D00000001                 or eax, 1000000h ;Set bit 24, to tell bios to give up control!
00000CE7 93                         xchg eax, ebx      ;Swap these two temporarily
00000CE8 66BAF80C                   mov dx, pci_index
00000CEC EF                         out dx, eax
00000CED 93                         xchg eax, ebx      ;Bring back out value to eax
00000CEE 66BAFC0C                   mov dx, pci_data
00000CF2 EF                         out dx, eax        ;Tell BIOS who is boss of the EHCI controller

00000CF3 4831C9                     xor rcx, rcx
00000CF6 89D8                       mov eax, ebx       ;Get address back into eax
                                  .ees1:
00000CF8 66FFC9                     dec cx
00000CFB 7449                       jz .weirdEHCI1
00000CFD E680                       out waitp, al      ;Wait a bit, for device to process request

00000CFF 66BAF80C                   mov dx, pci_index
00000D03 EF                         out dx, eax
00000D04 66BAFC0C                   mov dx, pci_data
00000D08 ED                         in eax, dx         ;Get word back into eax
00000D09 2500000100                 and eax, 10000h    ;BIOS should set this bit to zero
00000D0E 75E8                       jnz .ees1          ;Not zero yet, try again!

00000D10 4831C9                     xor rcx, rcx
00000D13 89D8                       mov eax, ebx      ;Get address back into eax
                                  .ees2:
00000D15 66FFC9                     dec cx
00000D18 742C                       jz .weirdEHCI1
00000D1A E680                       out waitp, al      ;Wait a bit, for device to process request

00000D1C 66BAF80C                   mov dx, pci_index
00000D20 EF                         out dx, eax
00000D21 66BAFC0C                   mov dx, pci_data
```

```
00000D25 ED                          in eax, dx          ;Get word back into eax
00000D26 2500000001                  and eax, 1000000h     ;This should set this bit to one now (OS control)
00000D2B 74E8                        jz .ees2          ;Not set yet, try again!
                                 ;Now we have control! :D Finally, now lets clear SMI bits
00000D2D 81C304000000                add ebx, 4h
00000D33 89D8                        mov eax, ebx
00000D35 66BAF80C                    mov dx, pci_index
00000D39 EF                          out dx, eax
00000D3A 31C0                        xor eax, eax
00000D3C 66BAFC0C                    mov dx, pci_data
00000D40 EF                          out dx, eax          ;NO MORE SMI INTERRUPTS

00000D41 59                          pop rcx
00000D42 5B                          pop rbx
00000D43 5A                          pop rdx
00000D44 58                          pop rax
00000D45 C3                          ret

                                 .weirdEHCI1:
00000D46 48B804130000000000-         mov rax, 1304h
00000D4F 00
00000D50 48BB07000000000000-         mov rbx, 0007h
00000D59 00
00000D5A 48B931000000000000-         mov rcx, failmsglen
00000D63 00
00000D64 48BD-                       mov rbp, .failmsg
00000D66 [730D000000000000]
00000D6E CD30                        int 30h     ; write strng
00000D70 F390                        pause
00000D72 F4                          hlt
00000D73 0A0D78484349206F72-     .failmsg: db 0Ah,0Dh,"xHCI or EHCI controller fail, halting system", 0Ah, 0Dh, 0
00000D7C 204548434920636F6E-
00000D85 74726F6C6C65722066-
00000D8E 61696C2C2068616C74-
00000D97 696E67207379737465-
00000DA0 6D0A0D00
                                 failmsglen     equ     $ - .failmsg

                                 pciExit:
                                 ;------------------------------------------------------------------
                                 ;                          End Proc                              :
                                 ;------------------------------------------------------------------
                                 ;------------------------------------------------------------------
                                 ;                  PIT Initialisation procedure                  :
                                 ;------------------------------------------------------------------
                                 PITreset:          ;Set Timer 0 to trigger every 55ms
00000DA4 B036                        mov al, 36h     ;Set bitmap for frequency write to channel 0 of pit
00000DA6 E643                        out PITcommand, al     ;43h = PIT command register
00000DA8 668B0425[35010000]          mov ax, word [pit_divisor]
00000DB0 E640                        out PIT0, al     ;mov low byte into divisor register
00000DB2 88E0                        mov al, ah       ;bring hi byte into low byte
00000DB4 E640                        out PIT0, al     ;mov hi byte into divisor register
                                 ;PIT unmasked below
                                 ;------------------------------------------------------------------
                                 ;                     End of Initialisation                      :
                                 ;------------------------------------------------------------------
                                 ;------------------------------------------------------------------
                                 ;                  RTC Initialisation procedure                  :
                                 ;------------------------------------------------------------------
                                 rtc_init:
                                 ;Set tick rate to 1024Hz and ensure RTC doesnt generate IRQ8
00000DB6 66B88A8A                    mov ax, 8A8Ah     ;Status A register with NMI disable
00000DBA E670                        out cmos_base, al
00000DBC E680                        out waitp, al     ;Latch wait
00000DBE EB00                        jmp short $+2
00000DC0 B026                        mov al, 00100110b ;32KHz timebase, 1024Hz square wave output
00000DC2 E671                        out cmos_data, al
                                 ;Now ensure NO interrupts are cooked
00000DC4 FEC4                        inc ah      ;ah=8Bh
00000DC6 88E0                        mov al, ah
00000DC8 E670                        out cmos_base, al
00000DCA E680                        out waitp, al  ;Latch wait
00000DCC EB00                        jmp short $+2
00000DCE B002                        mov al, 02h     ;Zero all int bits, time: BCD, 24hr, Daylight saving off
00000DD0 E671                        out cmos_data, al
                                 ;Clear any cooked IRQs
00000DD2 FEC4                        inc ah      ;ah=8Ch
00000DD4 88E0                        mov al, ah
00000DD6 E670                        out cmos_base, al
```

```
00000DD8 E680                          out waitp, al      ;Latch wait
00000DDA EB00                          jmp short $+2
00000DDC E471                          in al, cmos_data
                                   ;Get final CMOS RAM status byte
00000DDE B00D                          mov al, 0Dh       ;Status D register with NMI enable
00000DE0 E670                          out cmos_base, al
00000DE2 E680                          out waitp, al      ;Latch wait
00000DE4 EB00                          jmp short $+2
00000DE6 E471                          in al, cmos_data
                                   ;Unmask RTC and PIT here!
00000DE8 E4A1                          in al, pic2data     ;Get current state
00000DEA 24FE                          and al, 0FEh      ;Unmask RTC
00000DEC E6A1                          out pic2data, al
00000DEE E421                          in al, pic1data
00000DF0 24FA                          and al, 0FAh      ;Unmask PIT and Cascade
00000DF2 E621                          out pic1data, al
00000DF4 FB                            sti               ;Enable maskable interrupts
                                   ;-------------------------------------------------------------
                                   ;                 End of Initialisation                      :
                                   ;-------------------------------------------------------------
00000DF5 48B9C8000000000000-           mov rcx, 200      ;Beep for a 200ms
00000DFE 00
00000DFF BBA9040000                    mov ebx, 04A9h    ;Frequency divisor for 1000Hz tone
00000E04 66B800C5                      mov ax, 0C500h
00000E08 CD35                          int 35h
                                   ;-------------------------------------------------------------
                                   ;          Serial Port Initialisation procedure              :
                                   ;-------------------------------------------------------------
                                   ;Initial init procedure, check which ports exist and
                                   ; write the address to Data area
00000E0A 66B85A5A                      mov ax, 5A5Ah
00000E0E 4831C9                        xor rcx, rcx
00000E11 48BD-                         mov rbp, com_addresses
00000E13 [6700000000000000]
                                   checkCOM:
00000E1B 668B9409[C0190000]            mov dx, word [serial_abt + rcx*2]      ;Multiplied by 2 for word
                                                                 offsets
00000E23 6681C20700                    add dx, 7      ;Scratch register
00000E28 EE                            out dx, al      ;Output
00000E29 EB00                          jmp short $ + 2
00000E2B EC                            in al, dx      ;Read the value
00000E2C 38C4                          cmp ah, al     ;Check if theyre the same
00000E2E 7514                          jne COMinitproceed ;Scratch register non-existant, IO registers
                                                                 not present
00000E30 6681EA0700                    sub dx, 7      ;point dx back to base
00000E35 66899409[67000000]            mov word [com_addresses + rcx*2], dx      ;Save dx into data area
                                                                 table
00000E3D FEC1                          inc cl
00000E3F 80F904                        cmp cl, 4
00000E42 75D7                          jne checkCOM      ;Keep looping
                                   COMinitproceed:
                                   ;Sets all active COM ports to 2400,N,8,1, FIFO on, hware handshaking
00000E44 880C25[66000000]              mov byte [numCOM], cl
00000E4B 30C9                          xor cl, cl
                                   serialinit:
00000E4D 668B9409[67000000]            mov dx, word [com_addresses + rcx*2]  ;get the serial port base
                                                                 addr in dx
00000E55 6685D2                        test dx, dx
00000E58 743E                          jz COMinitexit      ;invalid address, port doesnt exist, init
                                                                 complete
                                   ;Disable interrupts
00000E5A 66FFC2                        inc dx           ;point at base + 1
00000E5D 30C0                          xor al, al       ;get zero to out it to the interrupt register
00000E5F EE                            out dx, al       ;Disable all interrupts
                                   ;Set DLAB
00000E60 6681C20200                    add dx, 2       ;point dx to the Line Control register (LCR)
00000E65 EC                            in al, dx       ;get the LCR byte into al
00000E66 0C80                          or al, 10000000b  ;set bit 7, DLAB bit on
00000E68 EE                            out dx, al       ;output the set bit
                                   ;Set baud rate
00000E69 6681EA0300                    sub dx, 3       ;word of baud divisor
00000E6E 66B83000                      mov ax, 0030h     ;the divisor for 2400 baud (cf table below)
00000E72 66EF                          out dx, ax       ;out put the divisor word
                                   ;Clear DLAB, set the parity, break stop and word length
00000E74 6681C20300                    add dx, 3       ;repoint at LCR (base + 3)
00000E79 B003                          mov al, 00000011b  ;DLAB off, 8,n,1, no break, no stick
00000E7B EE                            out dx, al       ;out that byte
                                   ;Clear FIFO
00000E7C 66FFCA                        dec dx           ;base + 2, FIFO register
00000E7F B006                          mov al, 00000110b    ;Clear FIFO, set char mode
00000E81 EE                            out dx, al       ;out that stuff
                                   ;Enable interrupts and RTS/DTR
00000E82 66FFCA                        dec dx           ;base + 1, Interrupt Enable Register
```

```
00000E85 B001                 mov al, 1        ;ONLY set the data receive interrupt, none of the
                                                          other
                                               ; status or transmit type interrupts
00000E87 EE                   out dx, al

00000E88 6681C20300           add dx, 3        ;base + 4, Modem control register
00000E8D EC                   in al, dx        ;preserve reserved upper bits
00000E8E 24E0                 and al, 11100000b
00000E90 0C0B                 or al, 00001011b     ;Set OUT2 (ie IRQ enable), set RTS/DTR.
00000E92 EE                   out dx, al
00000E93 66FFC1               inc cx
00000E96 EBB5                 jmp short serialinit
                          COMinitexit:
                          ;Unmask com ports here!
00000E98 E421                 in al, pic1data
00000E9A 24E7                 and al, 0E7h     ;Unmask Com lines 1 and 2 (bits 3 and 4)
00000E9C E621                 out pic1data, al
                          ;------------------------------------------------------------------
                          ;                      End of Initialisation                       :
                          ;------------------------------------------------------------------

                          ;------------------------------------------------------------------
                          ;                PS/2 Keyboard Initialisation procedure            :
                          ;------------------------------------------------------------------
                          keybsetup:     ;proc near
00000E9E 66B80A0E             mov ax, 0E0Ah
00000EA2 CD30                 int 30h
00000EA4 66B80D0E             mov ax, 0E0Dh
00000EA8 CD30                 int 30h     ;Send a crlf to con

00000EAA 66B80413             mov ax, 1304h
00000EAE 30FF                 xor bh, bh
00000EB0 48BD-                mov rbp, ps2stage.startMsg ;Prompt to strike a key
00000EB2 [DB11000000000000]
00000EBA CD30                 int 30h

00000EBC B05F                 mov al, 05Fh         ;PS/2 Stage signature
00000EBE E680                 out waitp, al
00000EC0 E6E9                 out bochsout, al

00000EC2 4D31C0               xor r8, r8           ;use as an stage counter
00000EC5 E926000000           jmp .step1
                          .kbscdetermine:
00000ECA B0F0                 mov al, 0F0h
00000ECC E8C5020000           call ps2talk.p3
00000ED1 E8AB020000           call ps2talk.p1
00000ED6 3CFA                 cmp al, 0FAh         ;ACK?
00000ED8 75F0                 jne .kbscdetermine   ;Not ack, try again
                          .pt1:
00000EDA 30C0                 xor al, al
00000EDC E8B5020000           call ps2talk.p3
00000EE1 E89B020000           call ps2talk.p1      ;Get ack into al,
00000EE6 3CFA                 cmp al, 0FAh
00000EE8 75F0                 jne .pt1
00000EEA E892020000           call ps2talk.p1      ;Get scancode into al
00000EEF C3                   ret

                          ;------------------------------------------------------------------
                          ;Do all writes using ps2talk:
                          ;      ah = 0 - Read Status port into al
                          ;      ah = 1 - Read Data port into al
                          ;      ah = 2 - Write al into Command port
                          ;      ah = 3 - Write al into Data port
                          ;------------------------------------------------------------------
                          ; Step 1) Disable ps2 port 1 using command word ADh and port 2
                                                          using command
                          ;  word A7h.
                          ; Step 2) Flush buffer and check bit 2 is set (else fail)
                          ; Step 3) Read controller configuration byte (command word 20h)
                          ; Step 4) Disable IRQs bits 0,1 (clear bit 0,1) [and manually
                                                          disable second
                          ;  ps2 port (bit 5 set)]
                          ; Step 5) Write controller config byte back (command word 60h)
                          ; Step 6) Test controller using AAh command word. Return 55h or
                                                          fail.
                          ; Step 7) Test ps2 port 1 using ABh command word. Return 00h or
                                                          fail.
                          ; Step 8) Enable ps2 port 1 using AEh command word. Enable IRQ by
                                                          setting bit 0
                          ;  of the config byte.
                          ; Step 9) Reset ps2 port 1 device using FFh data word. If AAh
                                                          returned,
                          ;  proceed, else if ACK (FAh), await AAh. FCh and FDh indicate
```

```
                                                            fail. FEh =
                                        ;   resend command.
                                        ; Step 10) Reset scan code set to 1 using F0h data word with 01h
                                                            data word. If
                                        ;  ACK (FAh) proceed, if RESEND (FEh), resend 10h tries.
                                        ; Setp 11) Enable scanning (ie keyboard sends scan codes) using
                                                            data word F4h.
                                        ;————————————————————————————————————————————————————————————————
                                        ;Step 1
                                        .step1:
00000EF0 B0AD                               mov al, 0ADh
00000EF2 E897020000                         call ps2talk.p2
00000EF7 B0A7                               mov al, 0A7h           ;Cancel second interface if it exists (DO
                                                            NOT REENABLE)
00000EF9 E890020000                         call ps2talk.p2
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00000EFE 49FFC0                             inc r8             ;Checkpoint 1
00000F01 E8CB020000                         call ps2stage      ;print which stage is complete
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
                                        ;Step 2

00000F06 E460                               in al, ps2data       ;manually flush ps2data port

                                        ;Step 3
                                        keyb0:
00000F08 B020                               mov al, 20h
00000F0A E87F020000                         call ps2talk.p2      ;out ps2command, al
00000F0F E86D020000                         call ps2talk.p1      ;Read config byte into al
                                        ;Step 4
00000F14 88C3                               mov bl, al           ;copy al into bl to check for bit 2
00000F16 80E3BC                             and bl, 10111100b  ;Disable translation, enable later if needed
                                        ;Step 5
00000F19 B060                               mov al, 60h
00000F1B E86E020000                         call ps2talk.p2      ;Write config byte command
00000F20 88D8                               mov al, bl
00000F22 E86F020000                         call ps2talk.p3      ;Out new config byte
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00000F27 49FFC0                             inc r8               ;Checkpoint 2
00000F2A E8A2020000                         call ps2stage        ;print which stage is complete
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
                                        ;Step 6
00000F2F B0AA                               mov al, 0AAh ;Can reset the config byte, out bl to ps2data at
                                                            end of stage
00000F31 E858020000                         call ps2talk.p2
00000F36 E846020000                         call ps2talk.p1
00000F3B 3C55                               cmp al, 55h
00000F3D 0F855B020000                       jne ps2error

00000F43 B060                               mov al, 60h  ;Previous code may have reset our new config byte,
                                                            resend it!
00000F45 E844020000                         call ps2talk.p2              ;Write config byte command
00000F4A 88D8                               mov al, bl
00000F4C E845020000                         call ps2talk.p3              ;Out new config byte
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00000F51 49FFC0                             inc r8               ;Checkpoint 3
00000F54 E878020000                         call ps2stage        ;print which stage is complete
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
                                        ;Step 7
00000F59 B0AB                               mov al, 0ABh              ;Test controller 1
00000F5B E82E020000                         call ps2talk.p2
00000F60 E81C020000                         call ps2talk.p1
00000F65 84C0                               test al, al                  ;Check al is zero
00000F67 0F8531020000                       jnz ps2error
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00000F6D 49FFC0                             inc r8               ;Checkpoint 4
00000F70 E85C020000                         call ps2stage        ;print which stage is complete
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
                                        ;Step 8
00000F75 B0AE                               mov al, 0AEh
00000F77 E812020000                         call ps2talk.p2

                                        ;Set IRQ 1 to connect to port 1
00000F7C B020                               mov al, 20h
00000F7E E80B020000                         call ps2talk.p2           ;Write
00000F83 E8F9010000                         call ps2talk.p1           ;Read
00000F88 0C01                               or al, 00000001b      ;Set bit 0
00000F8A 24EF                               and al, 11101111b      ;Zero bit 4, First port Clock
00000F8C 88C3                               mov bl, al
00000F8E B060                               mov al, 60h
00000F90 E8F9010000                         call ps2talk.p2
00000F95 88D8                               mov al, bl
00000F97 E8FA010000                         call ps2talk.p3
                                        ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
```

```
00000F9C 49FFC0                 inc r8              ;Checkpoint 5
00000F9F E82D020000             call ps2stage       ;print which stage is complete
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
                            ;Step 9
00000FA4 6631C9                 xor cx, cx
                            keyb1:
00000FA7 66FFC9                 dec cx ;timeout counter
00000FAA 0F84EE010000           jz ps2error
00000FB0 B0FF                   mov al, 0FFh
00000FB2 E8DF010000             call ps2talk.p3
                            .k1:
00000FB7 E8C5010000             call ps2talk.p1 ;read from ps2data
00000FBC 3CAA                   cmp al, 0AAh     ;success
00000FBE 7409                   je keyb20
00000FC0 3CFA                   cmp al, 0FAh     ;ACK
00000FC2 74F3                   je .k1           ;Loop if ACK recieved, just read ps2data
00000FC4 E9DEFFFFFF             jmp keyb1        ;Else, loop whole thing (assume fail recieved)
                            ;Step 10
                            keyb20:
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00000FC9 49FFC0                 inc r8              ;Checkpoint 6
00000FCC E800020000             call ps2stage       ;print which stage is complete
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00000FD1 31C9                   xor ecx, ecx
                            keyb2:
00000FD3 FFC9                   dec ecx
00000FD5 0F84C3010000           jz ps2error
                            .k0:
00000FDB B0F0                   mov al, 0F0h
00000FDD E8B4010000             call ps2talk.p3

00000FE2 B401                   mov ah, 01h
00000FE4 E898010000             call ps2talk.p1
00000FE9 3CFE                   cmp al, 0FEh     ;Did we recieve an resend?
00000FEB 74EE                   je .k0           ;Resend the data!
00000FED 3CFA                   cmp al, 0FAh     ;Compare to Ack?
00000FEF 75E2                   jne keyb2        ;If not equal, dec one from the loop counter and try again

00000FF1 B001                   mov al, 01h      ;write 01 to data port (set scan code set 1)
00000FF3 E89E010000             call ps2talk.p3
                            .k1:
00000FF8 E884010000             call ps2talk.p1      ;read data port for ACK or resend response
00000FFD 3CFA                   cmp al, 0FAh
00000FFF 7407                   je keyb30      ;IF ack revieved, scancode set, advance.
00001001 E2F5                   loop .k1       ;Keep polling port
00001003 E9CBFFFFFF             jmp keyb2
                            ;Step 11
                            keyb30:
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00001008 49FFC0                 inc r8              ;Checkpoint 7
0000100B E8C1010000             call ps2stage       ;print which stage is complete
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00001010 31C9                   xor ecx, ecx
                            keyb3:
00001012 66FFC9                 dec cx
00001015 0F8483010000           jz ps2error

0000101B B0F4                   mov al, 0F4h
0000101D E874010000             call ps2talk.p3
                            .k1:
00001022 E85A010000             call ps2talk.p1 ;read data port for ACK or resend response
00001027 3CFA                   cmp al, 0FAh
00001029 7407                   je keyb40
0000102B E2F5                   loop .k1         ;Keep polling port
0000102D E9E0FFFFFF             jmp keyb3        ;Fail, retry the whole process

                            ;Step 12
                            keyb40:
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
00001032 49FFC0                 inc r8              ;Checkpoint 8
00001035 E897010000             call ps2stage       ;print which stage is complete
                            ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
                            keyb4:
0000103A B0ED                   mov al, 0EDh        ;Set lights
0000103C E855010000             call ps2talk.p3
00001041 E83B010000             call ps2talk.p1     ;get response, remember ps2talk does its own timeout
00001046 3CFA                   cmp al, 0FAh
00001048 75F0                   jne keyb4           ;No ack, try again.
                            .k1:
0000104A B000                   mov al, 00h           ;Flash lock on and off
0000104C E845010000             call ps2talk.p3
```

```
2085 00001051 E82B010000                call ps2talk.p1      ;flush, remember ps2talk does its own timeout
2086
2087                                ;End Proc
2088                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2089 00001056 49FFC0                    inc r8             ;Checkpoint 9
2090 00001059 E873010000                call ps2stage      ;print which stage is complete
2091                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2092
2093                                keyb5:
2094 0000105E B0EE                      mov al, 0EEh       ;Echo command
2095 00001060 E831010000                call ps2talk.p3
2096 00001065 30C0                      xor al, al         ;Zero al to ensure that the result is EEh
2097                                .k1:
2098 00001067 E815010000                call ps2talk.p1
2099 0000106C 3CEE                      cmp al, 0EEh
2100 0000106E 7429                      je .k2             ;If equal, continue
2101 00001070 48BD-                     mov rbp, .noecho
2101 00001072 [8610000000000000]
2102 0000107A 66B80413                  mov ax, 1304h
2103 0000107E 30FF                      xor bh, bh
2104 00001080 CD30                      int 30h
2105 00001082 F390                      pause
2106 00001084 EB13                      jmp short .k2
2107 00001086 4E6F204563686F2072-   .noecho:        db    "No Echo recieved", 0Ah, 0Dh, 0
2107 0000108F 656369657665640A0D-
2107 00001098 00
2108                                .k2:
2109                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2110 00001099 49FFC0                    inc r8             ;Checkpoint 0Ah
2111 0000109C E830010000                call ps2stage      ;print which stage is complete
2112                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2113                                keyb6:     ;Set typematic rate/delay, 250ms, 30 reports/second
2114 000010A1 B0F3                      mov al, 0F3h       ;Set typematic rate
2115 000010A3 E8EE000000                call ps2talk.p3
2116 000010A8 30C0                      xor al, al         ;Set rate
2117 000010AA E8E7000000                call ps2talk.p3
2118 000010AF 6631C9                    xor cx, cx
2119                                .k1:
2120 000010B2 66FFC9                    dec cx
2121 000010B5 0F84E3000000              jz ps2error
2122 000010BB E8C1000000                call ps2talk.p1
2123 000010C0 3CFA                      cmp al, 0FAh     ;Ack?
2124 000010C2 75EE                      jnz .k1
2125                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2126 000010C4 49FFC0                    inc r8             ;Checkpoint 0Bh
2127 000010C7 E805010000                call ps2stage      ;print which stage is complete
2128                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2129                                scancode_faff:
2130 000010CC B020                      mov al, 20h        ;Get command byte from command port
2131 000010CE E8BB000000                call ps2talk.p2    ;al should contain command byte
2132 000010D3 88C4                      mov ah, al         ;temp save cmd byte in ah
2133
2134 000010D5 31C9                      xor ecx, ecx
2135                                .p1:
2136 000010D7 66FFC9                    dec cx
2137 000010DA 7439                      jz keybflushe
2138 000010DC E8E9FDFFFF                call keybsetup.kbscdetermine ;Get the current scancode set id
2139 000010E1 80CC01                    or ah, 00000001b     ;Do basic or, ie set IRQ for port 1
2140
2141                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2142 000010E4 49FFC0                    inc r8             ;Checkpoint 0Ch
2143 000010E7 E8E5000000                call ps2stage      ;print which stage is complete
2144                                ;<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
2145
2146 000010EC 3C43                      cmp al, 43h        ;43h is sc1 signature
2147 000010EE 740B                      je .writeback
2148 000010F0 3C01                      cmp al, 01h        ;Untranslated value
2149 000010F2 7407                      je .writeback
2150 000010F4 3CFA                      cmp al, 0FAh       ;Got an ACK for some reason, manually get next
                                                              byte
2151 000010F6 7416                      je .get_next_byte
2152
2153 000010F8 80CC40                    or ah, 01000000b     ;Neither value passed the test, invoke
                                                             translation
2154                                .writeback:
2155 000010FB 4989C7                    mov r15, rax       ;Save the scancode value to print later
2156 000010FE B060                      mov al, 60h
2157 00001100 E889000000                call ps2talk.p2
2158 00001105 88E0                      mov al, ah         ;return command byte
2159 00001107 E88A000000                call ps2talk.p3
2160 0000110C EB0B                      jmp short keybflush
2161                                .get_next_byte:
2162 0000110E E86E000000                call ps2talk.p1    ;Get the byte safely into al!
```

```
00001113 EBC2                        jmp short .p1     ;Recheck the scancode signature

                                 keybflushe:
00001115 4180CFF0                    or r15b,0F0h     ;Add signature to scancode value denoting error
                                 keybflush:     ;Flush internal ram of random bytes before enabling
                                                                  IRQ1
00001119 66B91000                    mov cx, 10h
                                 .kbf1:
0000111D 66FFC9                      dec cx
00001120 7404                        jz keybinitend
00001122 E460                        in al, ps2data          ;Read 16 bytes out (even if empty) and
                                                                    discard
00001124 EBF7                        jmp short .kbf1

                                 keybinitend:
00001126 30FF                        xor bh, bh  ;We are on page 0
00001128 B403                        mov ah, 03h ;Get current cursor row number in dh
0000112A CD30                        int 30h
0000112C B211                        mov dl, 17  ;End of PS/2 Keyboard message at column 17
0000112E 30FF                        xor bh, bh  ;Page 0
00001130 B402                        mov ah, 02h ;Set cursor
00001132 CD30                        int 30h

00001134 52                          push rdx     ;Save row/column in dx on stack
00001135 B91B000000                  mov ecx, 27  ;27 chars in keystrike message
                                 .kbe0:
0000113A B8200E0000                  mov eax, 0E20h
0000113F CD30                        int 30h
00001141 E2F7                        loop .kbe0

00001143 5A                          pop rdx
00001144 30FF                        xor bh, bh  ;Page 0
00001146 B402                        mov ah, 02h ;Set cursor
00001148 CD30                        int 30h

0000114A 48BD-                       mov rbp, ps2stage.okMsg
0000114C [0A12000000000000]
00001154 48B804130000000000-         mov rax, 1304h     ;print 0 terminated string
0000115D 00
0000115E 30FF                        xor bh, bh
00001160 CD30                        int 30h

                                 ;Unmask IRQ1 here
00001162 E421                        in al, pic1data
00001164 24FD                        and al, 0FDh     ;Unmask bit 1
00001166 E621                        out pic1data, al

00001168 E9A0000000                  jmp debuggerInit
                                 ;Relevant Procs for PS/2 keyboard setup
                                 ps2talk:
                                 ;    ah = 0 - Read Status port into al
                                 ;    ah = 1 - Read Data port into al
                                 ;    ah = 2 - Write al into Command port
                                 ;    ah = 3 - Write al into Data port
0000116D 84E4                        test ah, ah
0000116F 740D                        jz .p0
00001171 FECC                        dec ah
00001173 740C                        jz .p1
00001175 FECC                        dec ah
00001177 7415                        jz .p2
00001179 E918000000                  jmp .p3
                                 .p0:
0000117E E464                        in al, ps2status
00001180 C3                          ret
                                 .p1:
00001181 EB00                        jmp short $ + 2
00001183 E464                        in al, ps2status
00001185 A801                        test al, 1     ;Can something be read from KB?
00001187 74F8                        jz .p1         ;Zero if no. Not zero = read.
00001189 EB00                        jmp short $ + 2
0000118B E460                        in al, ps2data  ;Read it in
0000118D C3                          ret
                                 .p2:
0000118E E8(B6000000)                call ps2wait     ;preserves ax
00001193 E664                        out ps2command, al
00001195 C3                          ret
                                 .p3:
00001196 E8(B6000000)                call ps2wait
0000119B E660                        out ps2data, al
0000119D C3                          ret
                                 ps2error:
0000119E 48BD-                       mov rbp, .ps2errormsg
000011A0 [B411000000000000]
```

```
2241 000011A8 66B80413                  mov ax, 1304h
2242 000011AC 30FF                      xor bh, bh
2243 000011AE CD30                      int 30h
2244                                .loop:
2245 000011B0 F390                      pause
2246 000011B2 EBFC                      jmp short .loop
2247 000011B4 0A0D50532F32207374-   .ps2errormsg: db 0Ah, 0Dh,"PS/2 stage init error...", 0Ah, 0Dh, 0
2247 000011BD 61676520696E697420-
2247 000011C6 6572726F722E2E2E0A-
2247 000011CF 0D00
2248
2249                                ps2stage:
2250                                ;Outputs r8b to waitport and Bochs out
2251 000011D1 50                        push rax
2252 000011D2 4488C0                    mov al, r8b
2253 000011D5 E680                      out waitp, al
2254 000011D7 E6E9                      out bochsout, al
2255 000011D9 58                        pop rax
2256 000011DA C3                        ret
2257 000011DB 0A0D50532F32204B65-   .startMsg db 0Ah, 0Dh,'PS/2 Keyboard... Strike a key to
                                                                continue...',0
2257 000011E4 79626F6172642E2E2E-
2257 000011ED 20537472696B652061-
2257 000011F6 206B657920746F2063-
2257 000011FF 6F6E74696E75652E2E-
2257 00001208 2E00
2258 0000120A 4F4B00                .okMsg db 'OK', 0 ;This should go 17 chars in
2259                                ;————————————————————————————————————————————————————————————
2260                                ;                      End of Initialisation                 :
2261                                ;————————————————————————————————————————————————————————————
2262                                ;————————————————————————————————————————————————————————————
2263                                ;                Debugger Initialisation procedures          :
2264                                ;————————————————————————————————————————————————————————————
2265                                debuggerInit:
2266                                ;Int 40h can be used by the Debugger to return to it or if a DOS
                                                                present,
2267                                ; to return to DOS.
2268 0000120D 48B8-                     mov rax, MCP_int ;The application return point
2268 0000120F [FB1F000000000000]
2269 00001217 48BE40000000000000-       mov rsi, 40h
2269 00001220 00
2270 00001221 66BA008F                  mov dx, 8F00h     ;Attribs
2271 00001225 BB08000000                mov ebx, codedescriptor
2272 0000122A E8(CB000000)              call idtWriteEntry
2273                                ;————————————————————————————————————————————————————————————
2274                                ;                Drive Emum and Initialisation procedures    :
2275                                ;————————————————————————————————————————————————————————————
2276                                ideInitialisation:
2277                                ;This is truly read once code
2278                                ;Check primary and secondary bus for master and slave drives
2279                                ; Maximum of 4 "fixed" ATA drives
2280                                ;Use PIO for identification of drives on bus
2281 0000122F EB15                      jmp short hciParse
2282 00001231 B0A0                      mov al, 0A0h
2283 00001233 66BAF001                  mov dx, ata0_base
2284 00001237 48BF-                     mov rdi, sectorbuffer
2284 00001239 [C003000000000000]
2285 00001241 E8(49310000)              call IDE.identifyDevice
2286
2287                                ;           ——————————USB section below——————————
2288                                ;                     ——— PCI table parse ———
2289                                ;Parse the PCI tables for ehci controllers
2290                                hciParse:
2291 00001246 C60425[4B020000]00        mov byte [numMSD], 0
2292 0000124E 4C0FB70C2500E00000        movzx r9, word [lousbtablesize]
2293 00001257 BE02E00000                mov esi, lousbtablebase
2294 0000125C BF[15020000]              mov edi, eControllerList
2295                                .hcip1:
2296 00001261 6667F7064000              test word [esi], ehcimask     ;check if we at a ehci mask
2297 00001267 7418                      jz .hcip2   ;If not, skip adding to ehci table
2298                                    ;First catch all clause (temporary for version 1 of BIOS with
                                                                max 4
2299                                    ; controllers)
2300 00001269 803C25[14020000]04        cmp byte [eControllers], 4
2301 00001271 7430                      je .pr0    ;escape this whole setup proc if at 4 controllers
2302 00001273 67488B4602                mov rax, qword [esi + 2]     ;take pci and mmio address into rax
2303 00001278 48AB                      stosq                          ;store into rdi and inc rdi by 8
                                                                to next entry
2304 0000127A FE0425[14020000]          inc byte [eControllers]     ;increase the number of controllers
                                                                variable
2305                                .hcip2:
2306                                ;Any additional data saving occurs here
2307 00001281 81C60A000000              add esi, 10    ;Goto next table entry
```

```
2308 00001287 41FEC9                        dec r9b      ;Once all table entries exhausted, fall through
2309 0000128A 75D5                          jnz .hcip1
2310
2311                                    ;          ——— EHCI controller enumeration ———
2312                                    ;Enumerate each ehci ctrlr root hub for valid usb devices (hubs and
                                                                 valid MSD)
2313 0000128C 8A0C25[14020000]              mov cl, byte [eControllers]
2314 00001293 66B80413                      mov ax, 1304h
2315 00001297 48BD-                         mov rbp, .echiInitMsg
2315 00001299 [1B13000000000000]
2316 000012A1 CD30                          int 30h
2317                                    .pr0:    ;If ctrlr failure or ports exhausted, ret to here for next
                                                                 ctrlr
2318 000012A3 84C9                          test cl, cl
2319 000012A5 0F8409020000                  jz end      ;No EHCI controllers or last controler? Exit
2320 000012AB FEC9                          dec cl      ;Undo the absolute count from above
2321 000012AD 88C8                          mov al, cl
2322 000012AF E8(D9310000)                  call USB.setupEHCIcontroller
2323 000012B4 72ED                          jc .pr0      ;Continue to next controller
2324 000012B6 E8(54330000)                  call USB.ehciRunCtrlr        ;Activate online controller
2325 000012BB 72E6                          jc .pr0
2326 000012BD E8(B5330000)                  call USB.ehciAdjustAsyncSchedCtrlr ;Start schedule and lock
                                                                 ctrlr as online
2327 000012C2 72DF                          jc .pr0
2328 000012C4 E8(32340000)                  call USB.ehciCtrlrGetNumberOfPorts
2329 000012C9 88C2                          mov dl, al       ;Save the number of ports in dl
2330 000012CB 8A3425[47020000]              mov dh, byte [eActiveCtrlr]     ;Save current active ctrlr in dh
2331 000012D2 4D31D2                        xor r10, r10     ;Host hub 0 [ie Root Hub enum only] (for enum)
2332                                    .pr1:
2333 000012D5 FECA                          dec dl
2334 000012D7 49BC03000000000000-           mov r12, 3       ;Attempt three times to enumerate
2334 000012E0 00
2335                                    .pr11:
2336 000012E1 E8(CE370000)                  call USB.ehciEnumerateRootPort
2337 000012E6 7413                          jz .pr2
2338 000012E8 803C25[A9010000]20            cmp byte [msdStatus], 20h  ;General Controller Failure
2339 000012F0 0F84(9E310000)                je USB.ehciCriticalErrorWrapper
2340 000012F6 49FFCC                        dec r12
2341 000012F9 75E6                          jnz .pr11
2342                                    .pr2:
2343 000012FB 84D2                          test dl, dl
2344 000012FD 75D6                          jnz .pr1
2345 000012FF 84C9                          test cl, cl ;Once cl is zero we have gone through all
                                                                 controllers
2346 00001301 75A0                          jnz .pr0
2347
2348 00001303 B804130000                    mov eax, 1304h
2349 00001308 48BD-                         mov rbp, remDevInit.ok
2349 0000130A [1F14000000000000]
2350 00001312 30FF                          xor bh, bh
2351 00001314 CD30                          int 30h
2352 00001316 E929000000                    jmp remDevInit
2353 0000131B 0A0D496E697469616C-       .echiInitMsg db 0Ah,0Dh,"Initialising USB and EHCI root hubs...",0
2353 00001324 6973696E6720555342-
2353 0000132D 20616E642045484349-
2353 00001336 20726F6F7420687562-
2353 0000133F 732E2E2E00
2354                                    remDevInit:
2355                                    ;Devices on root hubs have been enumerated, and added to tables,
2356                                    ;Now we reset them (in the case of MSD) and enumerate further (on
                                                                 Hubs)
2357 00001344 66B80413                      mov ax, 1304h
2358 00001348 80F70B                        xor bh, 0bh
2359 0000134B 48BD-                         mov rbp, .rmhmsg
2359 0000134D [0314000000000000]
2360 00001355 CD30                          int 30h
2361                                    .hubs_init:
2362 00001357 48BE-                         mov rsi, hubDevTbl
2362 00001359 [6A02000000000000]
2363                                    ;First we scan for hubs only
2364                                    .redi1:
2365 00001361 803E00                        cmp byte [rsi], 0    ;Not an entry
2366 00001364 7417                          jz .hubnextentry
2367 00001366 807E0500                      cmp byte [rsi + 5], 0    ;If number of ports on hub is 0, dev
                                                                 uncofigured
2368 0000136A 7511                          jnz .hubnextentry  ;Device must be already enumerated
2369
2370 0000136C 8A4601                        mov al, byte [rsi + 1]   ;Get bus number into al
2371
2372 0000136F E8(B5330000)                  call USB.ehciAdjustAsyncSchedCtrlr
2373 00001374 7207                          jc .hubnextentry
2374
2375 00001376 E8(1D3E0000)                  call USB.ehciDevSetupHub   ;Only needs a valid device in rsi
```

```
2376 0000137B 7200                        jc .hubnextentry
2377                                  .hubnextentry:
2378 0000137D 4881C608000000              add rsi, hubDevTblEntrySize ;Goto next table entry
2379 00001384 4881FE[BA020000]            cmp rsi, hubDevTbl + 10*hubDevTblEntrySize  ;End of table
                                                                    address
2380 0000138B 72D4                        jb .redi1   ;We are still in table
2381                                  .hub_rescan:
2382                                  ;Now we check that all hubs are initialised
2383 0000138D 48BE-                       mov rsi, hubDevTbl   ;Return to head of table
2383 0000138F [6A02000000000000]
2384                                  ;Leave as a stub for now. Dont support deeper than 1 level of
                                                                    devices
2385                                  ;The specification allows for a maximum of 7 levels of depth.
2386                                  .msds_init:
2387 00001397 66B80413                    mov ax, 1304h
2388 0000139B 80F70B                      xor bh, 0bh
2389 0000139E 48BD-                       mov rbp, .ok
2389 000013A0 [1F14000000000000]
2390 000013A8 CD30                        int 30h
2391 000013AA 66B80413                    mov ax, 1304h
2392 000013AE 80F70B                      xor bh, 0bh
2393 000013B1 48BD-                       mov rbp, .msdmsg
2393 000013B3 [2314000000000000]
2394 000013BB CD30                        int 30h
2395 000013BD 48BE-                       mov rsi, msdDevTbl
2395 000013BF [BA02000000000000]
2396                                  .msd1:
2397 000013C7 803E00                      cmp byte [rsi], 0    ;Not an entry
2398 000013CA 740F                        jz .msdNextEntry
2399 000013CC E8(AC410000)                call USB.ehciMsdInitialise
2400 000013D1 7308                        jnc .msdNextEntry
2401 000013D3 FEC8                        dec al
2402 000013D5 0F84(9E310000)              jz USB.ehciCriticalErrorWrapper ;al = 1 => Host error,
2403                                  ;                                   al = 2 => Bad dev, removed
                                                                    from MSD tables
2404                                  .msdNextEntry:
2405 000013DB 4881C610000000              add rsi, msdDevTblEntrySize ;Goto next entry
2406 000013E2 4881FE[5A030000]            cmp rsi, msdDevTbl + 10*msdDevTblEntrySize
2407 000013E9 75DC                        jne .msd1
2408                                  .rediexit:
2409 000013EB 66B80413                    mov ax, 1304h
2410 000013EF 80F70B                      xor bh, 0bh
2411 000013F2 48BD-                       mov rbp, .ok
2411 000013F4 [1F14000000000000]
2412 000013FC CD30                        int 30h
2413 000013FE E93E000000                  jmp int33hinit
2414 00001403 0A0D496E697469616C-     .rmhmsg db 0Ah,0Dh,"Initialising USB ports...",0
2414 0000140C 6973696E6720555342-
2414 00001415 20706F7274732E2E2E-
2414 0000141E 00
2415 0000141F 204F4B00                .ok db " OK",0
2416 00001423 0A0D496E697469616C-     .msdmsg db 0Ah,0Dh,"Initialising MSD devices...",0
2416 0000142C 6973696E67204D5344-
2416 00001435 20646576696365732E-
2416 0000143E 2E2E00
2417                                  ;------------------------------------------------------------
2418                                  ;                          End of Enum                      :
2419                                  ;------------------------------------------------------------
2420                                  ;------------------------------------------------------------
2421                                  ;                    Int 33h Initialisation                 :
2422                                  ;------------------------------------------------------------
2423                                  int33hinit:
2424                                  ;Create Int 33h data table entry for each MSD/floppy device using
                                                                    steps 1-3.
2425                                  ;Go through MSD table and add devices to diskDevices
2426 00001441 48BD-                       mov rbp, usbDevTbl
2426 00001443 [4C02000000000000]
2427 0000144B 48BF-                       mov rdi, diskDevices
2427 0000144D [BB03000000000000]
2428                                  .i33i1:
2429 00001455 807D0208                    cmp byte [rbp + 2], 08h ;MSD USB Class code
2430 00001459 7525                        jne .i33proceed
2431                                  ;Successfully found a valid MSD device. Talk to it
2432 0000145B 668B4500                    mov ax, word [rbp]   ;Get address/bus pair
2433 0000145F E8(863C0000)                call USB.ehciGetDevicePtr    ;Get pointer to MSD dev in rsi
2434 00001464 E8(F9180000)                call disk_io.deviceInit
2435 00001469 3C01                        cmp al, 1    ;Critical error
2436 0000146B 0F84(9E310000)              je USB.ehciCriticalErrorWrapper
2437 00001471 3C02                        cmp al, 2    ;Device stopped responding, remove from USB data
                                                                    tables
2438 00001473 7420                        je .i33ibad
2439 00001475 3C03                        cmp al, 3    ;Device not added to data tables
2440 00001477 7407                        je .i33proceed
```

```
                                    ;Valid device added, increment rdi to next diskDevices table entry
00001479 4881C710000000                 add rdi, int33TblEntrySize
                                    .i33proceed:
00001480 4881FD[6A020000]               cmp rbp, usbDevTblEnd
00001487 741D                           je .i33iend
00001489 4881C503000000                 add rbp, usbDevTblEntrySize
00001490 E9C0FFFFFF                     jmp .i33i1
                                    .i33ibad:    ;If it goes here, clear table entry
00001495 48C70700000000                 mov qword [rdi], 0   ;Remove from diskDevice table
0000149C 668B06                         mov ax, word [rsi]
0000149F E8(113B0000)                   call USB.ehciRemoveDevFromTables     ;Remove from USB tables
000014A4 EBDA                           jmp short .i33proceed ;Goto next device
                                    .i33iend:
000014A6 8A0425[4B020000]               mov al, byte [numMSD]
000014AD 000425[A8010000]               add byte [i33Devices], al    ;Add the number of MSD devices to
                                                                      Int 33h total

                                    ;————————————————————————————————————————————————————————————————
                                    ;                    End of Enum and Initialisation              :
                                    ;————————————————————————————————————————————————————————————————

                                    end:
000014B4 66B80413                       mov ax, 1304h
000014B8 48BD-                          mov rbp, dbgmsg
000014BA [2A16000000000000]

000014C2 CD30                           int 30h
000014C4 8A0425[4B020000]               mov al, byte [numMSD]
000014CB B404                           mov ah, 04h
000014CD CD30                           int 30h

000014CF 66B80413                       mov ax, 1304h
000014D3 48BD-                          mov rbp, dbgmsg2
000014D5 [3B16000000000000]

000014DD CD30                           int 30h
000014DF 8A0425[A8010000]               mov al, byte [i33Devices]
000014E6 B404                           mov ah, 04h
000014E8 CD30                           int 30h

000014EA 66B80413                       mov ax, 1304h
000014EE 48BD-                          mov rbp, dbgmsg3
000014F0 [4F16000000000000]

000014F8 CD30                           int 30h
000014FA 8A0425[66000000]               mov al, byte [numCOM]
00001501 B404                           mov ah, 04h
00001503 CD30                           int 30h

00001505 803C25[A8010000]00             cmp byte [i33Devices], 0     ;If there are no i33 devices, skip
                                                                      bootstrap
0000150D 740C                           jz endNoDevFound

0000150F 66C70425FE7D000000-            mov word [7DFEh], 0 ;Clear out the old bootloader signature
00001518 00
00001519 CD39                           int 39h               ;Bootstrap loader
                                    endNoDevFound:
0000151B 48BD-                          mov rbp, endboot
0000151D [9A15000000000000]
00001525 66B80413                       mov ax, 1304h
00001529 CD30                           int 30h

0000152B 6631C0                         xor ax, ax  ;Pause for any key
0000152E CD36                           int 36h

00001530 66BB0700                       mov bx, 0007h     ;cls attribs
00001534 E8(F7000000)                   call cls

00001539 6631C9                         xor cx, cx
0000153C 6631D2                         xor dx, dx
0000153F B402                           mov ah, 2
00001541 30FF                           xor bh, bh
00001543 CD30                           int 30h

00001545 66B80413                       mov ax, 1304h
00001549 48BD-                          mov rbp, endboot2
0000154B [0A16000000000000]
00001553 CD30                           int 30h

00001555 4831C0                         xor rax, rax
00001558 4831DB                         xor rbx, rbx
0000155B 4831C9                         xor rcx, rcx
0000155E 4831D2                         xor rdx, rdx
00001561 4831F6                         xor rsi, rsi
00001564 4831FF                         xor rdi, rdi
00001567 4831ED                         xor rbp, rbp
0000156A 4D31C0                         xor r8, r8
0000156D 4D31C9                         xor r9, r9
```

```
2516 00001570 4D31D2                    xor r10, r10
2517 00001573 4D31DB                    xor r11, r11
2518 00001576 4D31E4                    xor r12, r12
2519 00001579 4D31ED                    xor r13, r13
2520 0000157C 4D31F6                    xor r14, r14
2521 0000157F 4D31FF                    xor r15, r15
2522
2523 00001582 CD38                      int 38h
2524
2525
2526 00001584 4C6F6164696E672053-   startboot:  db ”Loading SCP/BIOS...”, 0Ah, 0Dh, 0
2526 0000158D 43502F42494F532E2E-
2526 00001596 2E0A0D00
2527 0000159A 0A0D5343502F42494F-   endboot:    db     0Ah,0Dh,”SCP/BIOS system initialisation
                                                          complete”, 0Ah, 0Dh
2527 000015A3 532073797374656D20-
2527 000015AC 696E697469616C6973-
2527 000015B5 6174696F6E20636F6D-
2527 000015BE 706C6574650A0D
2528 000015C5 4E6F204F7065726174-            db ”No Operating System detected. Strike any key to launch
                                                          SYSDEBUG.”
2528 000015CE 696E67205379737465-
2528 000015D7 6D2064657465637465-
2528 000015E0 642E20537472696B65-
2528 000015E9 20616E79206B657920-
2528 000015F2 746F206C61756E6368-
2528 000015FB 205359534445425547-
2528 00001604 2E
2529 00001605 2E2E0A0D00                        db ”..”,0Ah, 0Dh,0
2530 0000160A 5374617274696E6720-   endboot2:   db ”Starting SCP/BIOS SYSDEBUG...”,0Ah,0Dh,0
2530 00001613 5343502F42494F5320-
2530 0000161C 53595344454255472E-
2530 00001625 2E2E0A0D00
2531 0000162A 0A0A0D4D5344206465-   dbgmsg:     db 0Ah,0Ah,0Dh,”MSD devices: ”,0
2531 00001633 76696365733A2000
2532 0000163B 0A0D496E7420333368-   dbgmsg2:    db 0Ah,0Dh,”Int 33h devices: ”,0
2532 00001644 20646576696365733A-
2532 0000164D 2000
2533 0000164F 0A0D434F4D20706F72-   dbgmsg3:    db 0Ah,0Dh,”COM ports: ”,0
2533 00001658 74733A2000
2534                                memprint:
2535                                ;Simple proc to print memory status
2536 0000165D 6631DB                    xor bx, bx
2537 00001660 48BD-                     mov rbp, .convmemmsg
2537 00001662 [B617000000000000]
2538 0000166A 66B80413                  mov ax, 1304h
2539 0000166E CD30                      int 30h
2540 00001670 CD32                      int 32h      ;Get conv Size
2541 00001672 25FFFF0000                and eax, 0FFFFh ;Clear upper bits
2542 00001677 E8F4000000                call .printdecimalword
2543 0000167C 48BD-                     mov rbp, .kb
2543 0000167E [2D18000000000000]
2544 00001686 66B80413                  mov ax, 1304h
2545 0000168A CD30                      int 30h
2546
2547 0000168C 66B801E8                  mov ax, 0E801h
2548 00001690 CD35                      int 35h
2549 00001692 25FFFF0000                and eax, 0FFFFh
2550 00001697 81E3FFFF0000              and ebx, 0FFFFh
2551 0000169D 81E1FFFF0000              and ecx, 0FFFFh
2552 000016A3 81E2FFFF0000              and edx, 0FFFFh
2553 000016A9 53                        push rbx
2554 000016AA 52                        push rdx
2555 000016AB 4839C8                    cmp rax, rcx
2556 000016AE 740C                      je .sense1     ;Sensible
2557 000016B0 4885C0                    test rax, rax
2558 000016B3 480F44C1                  cmovz rax, rcx
2559 000016B7 4885C0                    test rax, rax
2560 000016BA 7427                      jz .pt2
2561                                .sense1:
2562 000016BC 50                        push rax
2563 000016BD 48BD-                     mov rbp, .extmemmsg
2563 000016BF [D317000000000000]
2564 000016C7 66B80413                  mov ax, 1304h
2565 000016CB CD30                      int 30h
2566 000016CD 58                        pop rax
2567 000016CE E89D000000                call .printdecimalword
2568 000016D3 48BD-                     mov rbp, .kb
2568 000016D5 [2D18000000000000]
2569 000016DD 66B80413                  mov ax, 1304h
2570 000016E1 CD30                      int 30h
2571                                .pt2:
2572 000016E3 58                        pop rax
```

```
2573 000016E4 59                          pop rcx
2574 000016E5 4839C8                      cmp rax, rcx
2575 000016E8 740C                        je .sense2     ;Sensible
2576 000016EA 4885C0                      test rax, rax
2577 000016ED 480F44C1                    cmovz rax, rcx
2578 000016F1 4885C0                      test rax, rax
2579 000016F4 742B                        jz .pt3
2580                                  .sense2:
2581 000016F6 50                          push rax
2582 000016F7 48BD-                       mov rbp, .extmemmsg2
2582 000016F9 [F117000000000000]
2583 00001701 66B80413                    mov ax, 1304h
2584 00001705 CD30                        int 30h
2585 00001707 58                          pop rax
2586
2587 00001708 48C1E006                    shl rax, 6     ;Turn 64Kb into Kb
2588 0000170C E85F000000                  call .printdecimalword
2589 00001711 48BD-                       mov rbp, .kb
2589 00001713 [2D18000000000000]
2590 0000171B 66B80413                    mov ax, 1304h
2591 0000171F CD30                        int 30h
2592                                  .pt3:    ;Read total free size from big map
2593 00001721 50                          push rax
2594 00001722 48BD-                       mov rbp, .totalmem
2594 00001724 [1018000000000000]
2595 0000172C B804130000                  mov eax, 1304h
2596 00001731 CD30                        int 30h
2597 00001733 58                          pop rax
2598 00001734 488B0425[E0010000]          mov rax, qword [sysMem]
2599 0000173C 31DB                        xor ebx, ebx
2600 0000173E 8B1C25[E8010000]            mov ebx, dword [scpSize]
2601 00001745 4829D8                      sub rax, rbx
2602 00001748 48C1E80A                    shr rax, 0Ah                    ;Get number of Kb's free
2603 0000174C E81F000000                  call .printdecimalword
2604 00001751 48BD-                       mov rbp, .kb
2604 00001753 [2D18000000000000]
2605 0000175B 66B80413                    mov ax, 1304h
2606 0000175F CD30                        int 30h
2607
2608 00001761 B80A0E0000                  mov eax, 0E0Ah
2609 00001766 CD30                        int 30h
2610 00001768 B80D0E0000                  mov eax, 0E0Dh    ;CR/LF
2611 0000176D CD30                        int 30h
2612
2613 0000176F C3                          ret
2614
2615                                  .printdecimalword:
2616                                  ;Takes the qword in rax and prints its decimal representation
2617 00001770 52                          push rdx
2618 00001771 51                          push rcx
2619 00001772 53                          push rbx
2620 00001773 50                          push rax
2621 00001774 55                          push rbp
2622 00001775 4831C9                      xor rcx, rcx
2623 00001778 6631ED                      xor bp, bp     ;Use bp as #of digits counter
2624 0000177B 48BB0A000000000000-         mov rbx, 0Ah  ;Divide by 10
2624 00001784 00
2625
2626                                  .pdw0:
2627 00001785 FFC5                        inc ebp
2627 00001787 48C1E108                    shl rcx, 8     ;Space for next nybble
2628 0000178B 31D2                        xor edx, edx
2629 0000178D 48F7F3                      div rbx
2630 00001790 80C230                      add dl, '0'
2631 00001793 80FA39                      cmp dl, '9'
2632 00001796 7603                        jbe .pdw1
2633 00001798 80C207                      add dl, 'A'-'0'-10
2634                                  .pdw1:
2635 0000179B 88D1                        mov cl, dl     ;Save remainder byte
2636 0000179D 4885C0                      test rax, rax
2637 000017A0 75E3                        jnz .pdw0
2638                                  .pdw2:
2639 000017A2 88C8                        mov al, cl     ;Get most sig digit into al
2640 000017A4 48C1E908                    shr rcx, 8     ;Get next digit down
2641 000017A8 B40E                        mov ah, 0Eh
2642 000017AA CD30                        int 30h
2643 000017AC FFCD                        dec ebp
2644 000017AE 75F2                        jnz .pdw2
2645
2646 000017B0 5D                          pop rbp
2647 000017B1 58                          pop rax
2648 000017B2 5B                          pop rbx
2649 000017B3 59                          pop rcx
2650 000017B4 5A                          pop rdx
```

```
000017B5 C3                           ret
000017B6 0A0D4672656520436F-     .convmemmsg:          db 0Ah,0Dh,"Free Conventional Memory: ",0
000017BF 6E76656E74696F6E61-
000017C8 6C204D656D6F72793A-
000017D1 2000
000017D3 0A0D546F74616C204C-     .extmemmsg:           db 0Ah,0Dh,"Total Low Extended Memory: ",0
000017DC 6F7720457874656E64-
000017E5 6564204D656D6F7279-
000017EE 3A2000
000017F1 0A0D546F74616C2048-     .extmemmsg2:          db 0Ah,0Dh,"Total High Extended Memory: ",0
000017FA 69676820457874656E-
00001803 646564204D656D6F72-
0000180C 793A2000
00001810 0A0D546F74616C2046-     .totalmem:            db 0Ah,0Dh,"Total Free System Memory: ",0
00001819 726565205379737465-
00001822 6D204D656D6F72793A-
0000182B 2000
0000182D 4B00                    .kb:                  db "K",0
                                 ;————————————————————————Interrupt Tables——————————————————————————
                                IDT_TABLE:
                                CPU_IDT:
0000182F [124B000000000000]           dq i0
00001837 [1A4B000000000000]           dq i1
0000183F [294B000000000000]           dq i2
00001847 [384B000000000000]           dq i3
0000184F [474B000000000000]           dq i4
00001857 [564B000000000000]           dq i5
0000185F [654B000000000000]           dq i6
00001867 [744B000000000000]           dq i7
0000186F [834B000000000000]           dq i8
00001877 [924B000000000000]           dq i9
0000187F [A14B000000000000]           dq i10
00001887 [B04B000000000000]           dq i11
0000188F [BF4B000000000000]           dq i12
00001897 [CE4B000000000000]           dq i13
0000189F [DA4B000000000000]           dq i14
000018A7 [E64B000000000000]           dq i15
000018AF [F24B000000000000]           dq i16
000018B7 [FE4B000000000000]           dq i17
000018BF [0A4C000000000000]           dq i18
000018C7 [164C000000000000]           dq i19
000018CF [224C000000000000]           dq i20
000018D7 [2E4C000000000000]           dq i21
000018DF [314F000000000000]-          times 0Ah dq dummy_return_64      ;just return, reserved
                                                                          interrupts!
000018DF <rep Ah>
                                HW_IDT:
                                 ;—————————PIC1——————————:      ;Int 20h—27h
0000192F [0F01000000000000]           dq timer_IRQ0
00001937 [4F01000000000000]           dq kb_IRQ1
0000193F [2B4F000000000000]           dq dummy__interrupt.pic1
00001947 [CF09000000000000]           dq ser__IRQ3
0000194F [E109000000000000]           dq ser__IRQ4
00001957 [2B4F000000000000]           dq dummy__interrupt.pic1
0000195F [AA0A000000000000]           dq fdd__IRQ6
00001967 [B20A000000000000]           dq default__IRQ7
                                 ;—————————PIC2——————————:      ;Int 28h—2Fh
0000196F [D20A000000000000]           dq rtc__IRQ8
00001977 [244F000000000000]           dq dummy__interrupt.pic2
0000197F [244F000000000000]           dq dummy__interrupt.pic2
00001987 [244F000000000000]           dq dummy__interrupt.pic2
0000198F [244F000000000000]           dq dummy__interrupt.pic2
00001997 [244F000000000000]           dq dummy__interrupt.pic2
0000199F [050B000000000000]           dq hdd_IRQ14
000019A7 [150B000000000000]           dq default__IRQ15
                                SW_IDT:      ;Int 30h onwards!
000019AF [6B0C000000000000]           dq scr__io              ;Int 30h, VGA Screen drawing/TTY functions
000019B7 [F512000000000000]           dq machineWord__io      ;Int 31h, Give the BIOS hardware bitfield
000019BF [2C13000000000000]           dq convRAM__io          ;Int 32h, Give conv memory available
000019C7 [4F13000000000000]           dq disk__io             ;Int 33h, Storage device Functions
000019CF [C819000000000000]           dq serial__io           ;Int 34h, Serial Port Functions
000019D7 [361B000000000000]           dq misc__io             ;Int 35h, Misc functions
000019DF [D81E000000000000]           dq kb__io               ;Int 36h, Keyboard functions
000019E7 [5A1F000000000000]           dq printer__io          ;Int 37h, Reserved [Who uses parallel
                                                                          anymore?]
000019EF [FB1F000000000000]           dq MCP_int              ;Int 38h, launch MCP, and install its
                                                                          "API" handle
000019F7 [F32D000000000000]           dq bootstrapInt         ;Int 39h, restart the PC using an interrupt
000019FF [702E000000000000]           dq timerInt             ;Int 3Ah, Time of day
00001A07 [DB30000000000000]           dq ctrlbreak__io        ;Int 3Bh, user Break
00001A0F [314F000000000000]           dq dummy_return_64      ;Int 3Ch, user IRQ0 hook
00001A17 [DD30000000000000]           dq scr__params__io      ;Int 3Dh, Screen Mode parameters return
                                                                          function
```

```
00001A1F [E930000000000000]        dq disk_params_io     ;Int 3Eh, disk parameters return function
00001A27 [FB30000000000000]        dq cga_ret_io         ;Int 3Fh, video extention return function
                                 IDT_TABLE_Length equ $ - IDT_TABLE
                                 seg0len equ ($ - $$)


                                 ;------------------------------------------------------------------
                                 ;                  BIOS RESIDENT CODE AREA STARTS HERE           :
                                 ;------------------------------------------------------------------
                                 Segment codeResident follows=codeInit vfollows=data align=1 valign=1
                                 ;------------------------------Procs-------------------------------
                                 e820print:
00000000 56                            push rsi
00000001 52                            push rdx
00000002 51                            push rcx
00000003 53                            push rbx
00000004 50                            push rax
00000005 48BE-                         mov rsi, bigmapptr
00000007 [F005000000000000]
0000000F 480FB61425-                   movzx rdx, byte [bigmapSize]     ;Get the number of 24 byte
                                                                          entries
00000014 [D5010000]

                                 .e0:
00000018 48AD                          lodsq
0000001A E82D000000                    call .printqword
0000001F E845000000                    call .printpipe
00000024 48AD                          lodsq
00000026 E821000000                    call .printqword
0000002B E839000000                    call .printpipe
00000030 48AD                          lodsq
00000032 E815000000                    call .printqword
00000037 E844000000                    call .printcrlf
0000003C 6631C0                        xor ax, ax
0000003F CD36                          int 36h
00000041 48FFCA                        dec rdx
00000044 75D2                          jnz .e0
00000046 58                            pop rax
00000047 5B                            pop rbx
00000048 59                            pop rcx
00000049 5A                            pop rdx
0000004A 5E                            pop rsi
0000004B C3                            ret

                                 .printqword:
0000004C 4889C3                        mov rbx, rax
0000004F 480FCB                        bswap rbx
00000052 48B908000000000000-           mov rcx, 8
0000005B 00

                                 .pq1:
0000005C 88D8                          mov al, bl
0000005E B404                          mov ah, 04h
00000060 CD30                          int 30h
00000062 48C1EB08                      shr rbx, 8
00000066 E2F4                          loop .pq1
00000068 C3                            ret
                                 .printpipe:
00000069 55                            push rbp
0000006A 48BD-                         mov rbp, .pipestr
0000006C [7C00000000000000]
00000074 66B80413                      mov ax, 1304h
00000078 CD30                          int 30h
0000007A 5D                            pop rbp
0000007B C3                            ret
0000007C 207C2000                .pipestr:    db " | ",0
                                 .printcrlf:
00000080 55                            push rbp
00000081 48BD-                         mov rbp, .crlfstr
00000083 [9300000000000000]
0000008B 66B80413                      mov ax, 1304h
0000008F CD30                          int 30h
00000091 5D                            pop rbp
00000092 C3                            ret
00000093 0A0D00                  .crlfstr: db 0Ah,0Dh, 0
                                 beep:
                                 ;Destroys old PIT2 divisor.
                                 ;Input:
                                 ;     bx = Frequency divisor to use for tone
                                 ;     rcx = # of ms to beep for
                                 ;All registers preserved
00000096 50                            push rax
00000097 B0B6                          mov al, 0B6h ;Get PIT command bitfield, PIT2, lo/hi, Mode 3,
                                                                          Binary
00000099 E643                          out PITcommand, al

0000009B 6689D8                        mov ax, bx          ;Move frequency divisor into ax
```

```
0000009E E642                        out PIT2, al       ;Output lo byte of divisor
000000A0 88E0                        mov al, ah
000000A2 E642                        out PIT2, al       ;Output hi byte of divisor

000000A4 E461                        in al, port61h   ;Save original state of port 61h in ah
000000A6 0C03                        or al, 3         ;Set bits 0 and 1 to turn on the speaker
000000A8 E661                        out port61h, al

000000AA B486                        mov ah, 86h      ;Wait for beep to complete
000000AC CD35                        int 35h

000000AE E461                        in al, port61h     ;Read state of port 61h afresh
000000B0 24FC                        and al, ~3         ;Clear bits 0 and 1 to turn off the speaker
000000B2 E661                        out port61h, al

000000B4 58                          pop rax
000000B5 C3                          ret

                                 ps2wait:
000000B6 50                          push rax
                                 .wnok:
000000B7 EB00                        jmp short $ + 2
000000B9 E464                        in al, ps2status
000000BB A801                        test al, 1      ;Can something be read from KB?
000000BD 7406                        jz .wok         ;Zero = no, so loop back. Not zero = proceed to
                                                                   check if
                                                     ; something can be written
000000BF EB00                        jmp short $ + 2
000000C1 E460                        in al, ps2data      ;Read it in
000000C3 EBF2                        jmp short .wnok
                                 .wok:
000000C5 A802                        test al, 2    ;Can something be written to KB?
000000C7 75EE                        jnz .wnok     ;Zero if yes and proceed.
000000C9 58                          pop rax
000000CA C3                          ret

                                 idtWriteEntry:
                                 ;------------------------------------------------------------------
                                 ;This proc writes an interrupt handler to a particular IDT entry.
                                 ; rax = Interrupt handler ptr      (qword)
                                 ; rsi = Interrupt Number           (qword)
                                 ; dx = Attributes word             (word)
                                 ; bx = Segment selector            (word)
                                 ;On return:
                                 ; rsi incremented by 1
                                 ; Entry written
                                 ;------------------------------------------------------------------
000000CB 56                          push rsi
000000CC 48C1E604                    shl rsi, 4h      ;Multiply IDT entry number by 16
000000D0 48033425[04000000]          add rsi, qword [IDTpointer.Base]      ;rsx points to IDT entry
000000D8 668906                      mov word [rsi], ax   ;Get low word into offset 15...0
000000DB 66895E02                    mov word [rsi + 2], bx   ;Move segment selector into place
000000DF 66895604                    mov word [rsi + 4], dx   ;Move attribute word into place
000000E3 48C1E810                    shr rax, 10h      ;Bring next word low
000000E7 66894606                    mov word [rsi + 6], ax   ;Get low word into offset 31...16
000000EB 48C1E810                    shr rax, 10h      ;Bring last dword low
000000EF 894608                      mov dword [rsi + 8], eax
000000F2 5E                          pop rsi
000000F3 48FFC6                      inc rsi           ;rsi contains number of next interrupt handler
000000F6 C3                          ret

                                 cls:     ;Clear the screen, bl attrib, always clear active scr
000000F7 50                          push rax
000000F8 52                          push rdx
000000F9 B40F                        mov ah, 0Fh
000000FB CD30                        int 30h ;Get current active page

000000FD B402                        mov ah, 02h     ;Set cursor pos
000000FF 6631D2                      xor dx, dx
00000102 CD30                        int 30h
00000104 88DF                        mov bh, bl
                                 ;No need for coordinates since al=00 means reset fullscreen
00000106 66B80006                    mov ax, 0600h
0000010A CD30                        int 30h      ;scroll page with grey on black
0000010C 5A                          pop rdx
0000010D 58                          pop rax
0000010E C3                          ret

                                 ;---------------------Interrupt Service routines--------------------

                                 ;=======================HARDWARE INTERRUPTS=========================
                                 ;-----------------Timer Interrupt IRQ 0/Int 20h--------------------
                                 ;This interrupt simply increments an internal timer and
```

```
                                        ; calls a software interrupt (5Ch) which can be used by user
                                        ; applications.
                                        ;————————————————————————————————————————————
                                        timer_IRQ0:
0000010F FB                                 sti
00000110 50                                 push rax
00000111 FF0425[37010000]                   inc dword [pit_ticks]
00000118 8B0425[37010000]                   mov eax, dword [pit_ticks]
0000011F 25FFFF1F00                         and eax, 1FFFFFh     ;Clear OF bit [mask on bits 20:0]
00000124 3DB0001800                         cmp eax, 1800B0h     ;Ticks in one full day
00000129 7519                               jnz .tret            ;Not quite there
0000012B 66C70425[37010000]-                mov word [pit_ticks], 0      ;Zero lo count
00000133 0000
00000135 C60425[39010000]00                 mov byte [pit_ticks + 2], 0      ;Zero hi count
0000013D FE0425[3A010000]                   inc byte [pit_ticks + 3]      ;Increment day OF counter
                                        .tret:
00000144 CD3C                               int 3Ch          ;Call user handler

00000146 B020                               mov al, EOI
00000148 E620                               out pic1command, al
0000014A E680                               out waitp, al      ;allow one io cycle to run

0000014C 58                                 pop rax
0000014D 48CF                               iretq
                                        ;————————————————End of Interrupt——————————————
                                        ;————————————Keyboard Interrupt IRQ 1/Int 21h————————
                                        ;This interrupt takes scancodes from the PC keyboard, translates
                                        ; them into scancode/ASCII char pair and stores the pair into
                                        ; the buffer for the software keyboard interrupt to use.
                                        ;————————————————————————————————————————————
                                        kb_IRQ1:
0000014F FB                                 sti          ;Reenable interrupts
00000150 50                                 push rax
00000151 53                                 push rbx
00000152 51                                 push rcx
00000153 57                                 push rdi
00000154 4831C0                             xor rax, rax

                                        .k0:
00000157 E460                               in al, ps2data      ;Get the scancode (Set 1)
00000159 4885C0                             test rax, rax      ;Check to see if we got an error code from the
                                                                 keyboard.
0000015C 0F84A0020000                       jz .kb_error
00000162 483D80000000                       cmp rax, 80h
00000168 0F8ED8000000                       jle .basickey      ;A normal keypress, nothing too magical.
0000016E 483DE0000000                       cmp rax, 0E0h      ;Compare against special keys
00000174 7472                               je .special_keys
00000176 483DE1000000                       cmp rax, 0E1h      ;Pause
0000017C 747D                               je .pause
0000017E 483DAA000000                       cmp rax, 0AAh      ;LShift released
00000184 0F8490000000                       je .lshift_released
0000018A 483DB6000000                       cmp rax, 0B6h      ;RShift released
00000190 0F8488000000                       je .rshift_released
00000196 483DB8000000                       cmp rax, 0B8h      ;Alt Shift released
0000019C 7474                               je .alt_shift_released
0000019E 483D9D000000                       cmp rax, 9Dh      ;Ctrl Shift released
000001A4 7470                               je .ctrl_shift_released
000001A6 483D2B0D0000                       cmp rax, 0D2Bh      ;Toggle Insert
000001AC 7460                               je .insert_released
000001AE EB25                               jmp short .kb1_exit      ;Just exit if something weird gets sent

                                        .kb_store_in_buffer:
000001B0 488B1C25[4A000000]                 mov rbx, qword [kb_buf_tail]      ;point rbx to tail
000001B8 4889DF                             mov rdi, rbx ;Save bx in di for storing the data in AX after bx
                                                                 gets inc
000001BB E8811D0000                         call kb_io.kb_ptr_adv              ;safely advance the pointer
000001C0 483B1C25[42000000]                 cmp rbx, qword [kb_buf_head]      ;Have we wrapped around?
000001C8 745F                               je .kb_buf_full_beep              ;discard and beep
000001CA 668907                             mov word [rdi], ax                 ;mov scancode/ascii pair into
                                                                 buffer
000001CD 48891C25[4A000000]                 mov qword [kb_buf_tail], rbx      ;store new pointer back into
                                                                 tail


                                        .kb1_exit:
000001D5 B0FC                               mov al, ~(kb_flag2_e0 | kb_flag2_e1)          ;move the notted
                                                                 version into al
000001D7 200425[64000000]                   and byte [kb_flags_2], al          ;Nullify the e0 and e1 flag
                                        .kb1_exit_e0:
000001DE B020                               mov al, EOI
000001E0 E620                               out pic1command, al      ;End of interrupt to pic1 command port

000001E2 5F                                 pop rdi
000001E3 59                                 pop rcx
```

```
000001E4 5B                         pop rbx
000001E5 58                         pop rax
000001E6 48CF                       iretq

                                .special_keys:     ;An E0 process
000001E8 B002                       mov al, kb_flag2_e0          ;Set the bit for the flag
000001EA 080425[64000000]           or byte [kb_flags_2], al     ;Set the flag
000001F1 802425[64000000]FE         and byte [kb_flags_2], ~kb_flag2_e1     ;clear the E1 bit
000001F9 EBE3                       jmp short .kb1_exit_e0       ;Exit from IRQ without resetting
                                                                 flags
                                .pause:     ;An E1 process
000001FB B001                       mov al, kb_flag2_e1          ;Set the bit for the flag
000001FD 080425[64000000]           or byte [kb_flags_2], al     ;Toggle the flag, since 9D and C5
                                                                 will be
                                                                 ; ignored by the Int handler
00000204 802425[64000000]FD         and byte [kb_flags_2], ~kb_flag2_e0     ;clear the E0 bit
0000020C EBD0                       jmp short .kb1_exit_e0


                                .insert_released:
0000020E B07F                       mov al, ~kb_flag_insset      ;Flag negation
00000210 EB0E                       jmp short .shift_release_common
                                .alt_shift_released:
00000212 B0F7                       mov al, ~kb_flag_alt         ;Flag negation
00000214 EB0A                       jmp short .shift_release_common
                                .ctrl_shift_released:
00000216 B0FB                       mov al, ~kb_flag_ctrl        ;Flag negation
00000218 EB06                       jmp short .shift_release_common
                                .lshift_released:
0000021A B0FD                       mov al, ~kb_flag_lshift      ;Flag negation
0000021C EB02                       jmp short .shift_release_common
                                .rshift_released:
0000021E B0FE                       mov al, ~kb_flag_rshift      ;Flag negation
                                .shift_release_common:
00000220 200425[62000000]           and byte [kb_flags], al      ;Clear the relevant bit
00000227 EBAC                       jmp short .kb1_exit



                                .kb_buf_full_beep:
00000229 53                         push rbx
0000022A 51                         push rcx
0000022B BBA9040000                 mov ebx, 04A9h ;Frequency divisor for 1000Hz tone
00000230 48B9F4010000000000-        mov rcx, 500   ;Beep for a 1/2 second
00000239 00
0000023A E857FEFFFF                 call beep
0000023F 59                         pop rcx
00000240 5B                         pop rbx
00000241 E98FFFFFFF                 jmp .kb1_exit

                                .basickey:             ;al contains the scancode
00000246 483D46000000               cmp rax, 46h
0000024C 0F8421010000               je .e0special    ;ctrl+break checker (E0 46h is make for break
                                                                 haha)
                                .kbbk1:
00000252 483D2A000000               cmp rax, 2Ah     ;Left Shift scancode
00000258 0F84E4000000               je .lshift_pressed
0000025E 483D36000000               cmp rax, 36h     ;Right Shift scancode
00000264 0F84DC000000               je .rshift_pressed
0000026A 483D38000000               cmp rax, 38h     ;Alt Shift key scancode
00000270 0F84C4000000               je .alt_shift_pressed
00000276 483D1D000000               cmp rax, 1Dh     ;Ctrl Shift key scancode
0000027C 0F84BC000000               je .ctrl_shift_pressed

00000282 483D3A000000               cmp rax, 3Ah     ;Caps lock key
00000288 0F84CA000000               je .caps_lock
0000028E 483D45000000               cmp rax, 45h     ;Num lock key
00000294 0F84C2000000               je .num_lock
                                ;     cmp rax, 46h     ;Scroll lock key
                                ;     je .scroll_lock
0000029A 483D52000000               cmp rax, 52h     ;Insert key pressed
000002A0 0F84AE000000               je .ins_toggle
000002A6 483D53000000               cmp rax, 53h     ;Delete key, for CTRL+ALT+DEL
000002AC 0F8408010000               je .ctrl_alt_del
                                .keylookup:
000002B2 48BB-                      mov rbx, .kb_sc_ascii_lookup
000002B4 [3F04000000000000]
                                                 ; upper 7 bytes of rax are completely clear
000002BC 66C1E004                   shl ax, 4        ;multiply ax, the scancode, by 16, to offset to
                                                                 correct row
000002C0 4801C3                     add rbx, rax     ;offset rbx to the correct row
                                ;Now check shift states, to align with column. rax is free again
000002C3 8A0425[62000000]           mov al, byte [kb_flags]

000002CA A802                       test al, kb_flag_lshift
```

```
000002CC 7525                         jnz .addshiftvalue           ;If that bit is set, jump!
000002CE A801                         test al, kb_flag_rshift
000002D0 7521                         jnz .addshiftvalue
000002D2 A804                         test al, kb_flag_ctrl
000002D4 752E                         jnz .addctrlvalue
000002D6 A808                         test al, kb_flag_alt
000002D8 7533                         jnz .addaltvalue
000002DA A820                         test al, kb_flag_numset
000002DC 7538                         jnz .addnumvalue
000002DE A840                         test al, kb_flag_capsset
000002E0 753D                         jnz .addcapsvalue

                                  .keyget:
000002E2 668B03                       mov ax, word [rbx]  ;Get correct word into ax!
000002E5 6685C0                       test ax, ax          ;check if the value is zero, if so, dont
                                                                store in buffer
000002E8 0F84E7FEFFFF                 jz .kb1_exit
000002EE E9BDFEFFFF                   jmp .kb_store_in_buffer

                                  .addshiftvalue:     ;first check if we shift with caps or num
000002F3 A820                         test al, kb_flag_numset
000002F5 753A                         jnz .addshiftnum
000002F7 A840                         test al, kb_flag_capsset
000002F9 752D                         jnz .addshiftcaps
                                      ;Collapse through, it is just shift, add 2 to rbx
000002FB 4881C302000000               add rbx, 1h*2h
00000302 EBDE                         jmp short .keyget
                                  .addctrlvalue:
00000304 4881C304000000               add rbx, 2h*2h
0000030B EBD5                         jmp short .keyget
                                  .addaltvalue:
0000030D 4881C306000000               add rbx, 3h*2h
00000314 EBCC                         jmp short .keyget
                                  .addnumvalue:
00000316 4881C308000000               add rbx, 4h*2h
0000031D EBC3                         jmp short .keyget
                                  .addcapsvalue:
0000031F 4881C30A000000               add rbx, 5h*2h
00000326 EBBA                         jmp short .keyget
                                  .addshiftcaps:
00000328 4881C30C000000               add rbx, 6h*2h
0000032F EBB1                         jmp short .keyget
                                  .addshiftnum:
00000331 4881C30E000000               add rbx, 7h*2h
00000338 EBA8                         jmp short .keyget

                                  .alt_shift_pressed:
0000033A B008                         mov al, kb_flag_alt
0000033C EB0A                         jmp short .shift_pressed_common
                                  .ctrl_shift_pressed:
0000033E B004                         mov al, kb_flag_ctrl
00000340 EB06                         jmp short .shift_pressed_common
                                  .lshift_pressed:
00000342 B002                         mov al, kb_flag_lshift
00000344 EB02                         jmp short .shift_pressed_common
                                  .rshift_pressed:
00000346 B001                         mov al, kb_flag_rshift
                                  .shift_pressed_common:
00000348 080425[62000000]             or byte [kb_flags], al      ;toggle flag bits
0000034F E981FEFFFF                   jmp .kb1_exit               ;Exit

                                  .ins_toggle:
00000354 B080                         mov al, kb_flag_insset
00000356 EB0A                         jmp short .lock_common
                                  .caps_lock:
00000358 B040                         mov al, kb_flag_capsset
0000035A EB06                         jmp short .lock_common
                                  .num_lock:
0000035C B020                         mov al, kb_flag_numset
0000035E EB02                         jmp short .lock_common
                                  .scroll_lock:
00000360 B010                         mov al, kb_flag_scrlset
                                  .lock_common:
00000362 300425[62000000]             xor byte [kb_flags], al      ;toggle bit
00000369 E875000000                   call .set_kb_lights
0000036E E962FEFFFF                   jmp .kb1_exit

                                  .e0special:
00000373 F60425[64000000]02           test byte [kb_flags_2], 00000010b      ;Check for E0 set
0000037B 7505                         jnz .ctrl_break
0000037D E9DEFFFFFF                   jmp .scroll_lock      ;Assume scroll lock set
                                  .ctrl_break:
00000382 800C25[65000000]01           or byte [break_flag], 1          ;set break_flag
```

```
0000038A 6631C0                       xor ax, ax
0000038D 53                           push rbx
0000038E 48BB-                        mov rbx, kb_buffer              ;mov the buffer addr to rbx
00000390 [2200000000000000]
00000398 48891C25[42000000]           mov qword [kb_buf_head], rbx
000003A0 48891C25[4A000000]           mov qword [kb_buf_tail], rbx
000003A8 668903                       mov word [rbx], ax     ;Store zero as the first two bytes of the
000003AB 5B                           pop rbx
000003AC CD3B                         int 3Bh                         ;Call the CTRL+Break handler
000003AE 200425[65000000]             and byte [break_flag], al       ;clear break_flag
000003B5 E91BFEFFFF                   jmp .kb1_exit           ;return clearing E0

                                  .ctrl_alt_del:
000003BA 50                           push rax     ;save scancode
000003BB 8A0425[64000000]             mov al, byte [kb_flags_2]
000003C2 A802                         test al, kb_flag2_e0     ;Delete scancode is E0, 53, check if we
                                                               first had E0
000003C4 7417                         jz .ctrl_alt_del_no_reset

000003C6 8A0425[62000000]             mov al, byte [kb_flags]
000003CD 240C                         and al,  kb_flag_ctrl | kb_flag_alt
000003CF 3C0C                         cmp al, kb_flag_ctrl | kb_flag_alt     ;Test if Ctrl + Alt is
                                                               being pressed
000003D1 750A                         jne .ctrl_alt_del_no_reset
                                  .ctrl_alt_del_killPC:
000003D3 E464                         in al, 64h     ;Check if the input buffer is empty
000003D5 A802                         test al, 2     ;Check if clear
000003D7 75FA                         jne .ctrl_alt_del_killPC     ;keep waiting
000003D9 B0FE                         mov al, 0FEh     ;Pulse kill lines
000003DB E664                         out 64h, al
                                      ;PC dead, time to reboot!
                                  .ctrl_alt_del_no_reset:
000003DD 58                           pop rax          ;return the OG scancode and proceed as normal
000003DE E9CFFEFFFF                   jmp .keylookup



                                  .set_kb_lights:
000003E3 50                           push rax

000003E4 E8CDFCFFFF                   call ps2wait

000003E9 B0ED                         mov al, 0EDh
000003EB E660                         out ps2data, al

000003ED E8C4FCFFFF                   call ps2wait

000003F2 8A0425[62000000]             mov al, byte [kb_flags]     ;get flag into al
000003F9 C0E804                       shr al, 4
000003FC 2407                         and al, 111b     ;mask Insert bit off to isolate the
                                                               NUM,CAPS,SCRL status
                                                 ; bits <=> LED status.
000003FE E660                         out ps2data, al     ;send the led status away

00000400 58                           pop rax
00000401 C3                           ret

                                  .kb_error:       ;If error recieved from Keyboard, hang the system,
                                                               cold reboot
                                                   ; needed.
00000402 FA                           cli          ;Disable interrupts/Further keystrokes
00000403 66BB0700                     mov bx, 0007h     ;cls attribs
00000407 E8EBFCFFFF                   call cls     ;clear the screen
0000040C 66B80413                     mov ax, 1304h
00000410 30FF                         xor bh, bh
00000412 48BD-                        mov rbp, .kb_error_msg
00000414 [2204000000000000]
0000041C CD30                         int 30h
                                  .kber1:
0000041E F390                         pause
00000420 EBFC                         jmp short .kber1
00000422 4B6579626F61726420-      .kb_error_msg:     db     "Keyboard Error. Halting...", 0Ah, 0Dh, 0
0000042B 4572726F722E204861-
00000434 6C74696E672E2E2E0A-
0000043D 0D00


                                  .kb_sc_ascii_lookup:     ;Scancodes 00h-58h
                                  ; Scancodes 00h-0Fh
                                  ;    base    shift    ctrl    alt    num    caps    shcap   shnum
0000043F 000000000000000000-       dw 0000h, 0000h, 0000h, 0000h, 0000h, 0000h, 0000h, 0000h ;NUL
00000448 00000000000000
0000044F 1B011B011B011B011B-       dw 011Bh, 011Bh, 011Bh, 011Bh, 011Bh, 011Bh, 011Bh, 011Bh ;Esc
00000458 011B011B011B01
0000045F 310221020000007831-       dw 0231h, 0221h, 0000h, 7800h, 0231h, 0231h, 0221h, 0221h ;1 !
```

```
00000468 02310221022102
0000046F 320322030003007932-     dw 0332h, 0322h, 0300h, 7900h, 0332h, 0332h, 0322h, 0322h ;2 ”
00000478 03320322032203
0000047F 33049C040000007A33-     dw 0433h, 049Ch, 0000h, 7A00h, 0433h, 0433h, 049Ch, 049Ch ;3 £
00000488 0433049C049C04
0000048F 340524050000007B34-     dw 0534h, 0524h, 0000h, 7B00h, 0534h, 0534h, 0524h, 0524h ;4 $
00000498 05340524052405
0000049F 350625060000007C35-     dw 0635h, 0625h, 0000h, 7C00h, 0635h, 0635h, 0625h, 0625h ;5 %
000004A8 06350625062506
000004AF 36075E071E07007D36-     dw 0736h, 075Eh, 071Eh, 7D00h, 0736h, 0736h, 075Eh, 075Eh ;6 ^
000004B8 0736075E075E07
000004BF 370826080000007E37-     dw 0837h, 0826h, 0000h, 7E00h, 0837h, 0837h, 0826h, 0826h ;7 &
000004C8 08370826082608
000004CF 38092A090000007F38-     dw 0938h, 092Ah, 0000h, 7F00h, 0938h, 0938h, 092Ah, 092Ah ;8 *
000004D8 0938092A092A09
000004DF 390A280A0000008039-     dw 0A39h, 0A28h, 0000h, 8000h, 0A39h, 0A39h, 0A28h, 0A28h ;9 (
000004E8 0A390A280A280A
000004EF 300B290B0000008130-     dw 0B30h, 0B29h, 0000h, 8100h, 0B30h, 0B30h, 0B29h, 0B29h ;0 )
000004F8 0B300B290B290B
000004FF 2D0C5F0C000000822D-     dw 0C2Dh, 0C5Fh, 0000h, 8200h, 0C2Dh, 0C2Dh, 0C5Fh, 0C5Fh ;- _
00000508 0C2D0C5F0C5F0C
0000050F 3D0D2B0D000000833D-     dw 0D3Dh, 0D2Bh, 0000h, 8300h, 0D3Dh, 0D3Dh, 0D2Bh, 0D2Bh ;= +
00000518 0D3D0D2B0D2B0D
0000051F 080E080E7F0E000008-     dw 0E08h, 0E08h, 0E7Fh, 0000h, 0E08h, 0E08h, 0E08h, 0E08h ;bksp
                                                                    (ctrl -> del)
00000528 0E080E080E080E
0000052F 090F000F0000000009-     dw 0F09h, 0F00h, 0000h, 0000h, 0F09h, 0F09h, 0F00h, 0F00h ;L2R
                                                                    Horizontal Tab
00000538 0F090F000F000F

                                 ; Scancodes 10h-1Fh
                                 ;   base   shift   ctrl   alt    num    caps   shcap  shnum
0000053F 711051101110001071-     dw 1071h, 1051h, 1011h, 1000h, 1071h, 1051h, 1071h, 1051h ;q Q
00000548 10511071105110
0000054F 771157111711001177-     dw 1177h, 1157h, 1117h, 1100h, 1177h, 1157h, 1177h, 1157h ;w W
00000558 11571177115711
0000055F 651245120512001265-     dw 1265h, 1245h, 1205h, 1200h, 1265h, 1245h, 1265h, 1245h ;e E
00000568 12451265124512
0000056F 721352131213001372-     dw 1372h, 1352h, 1312h, 1300h, 1372h, 1352h, 1372h, 1352h ;r R
00000578 13521372135213
0000057F 741454141414001474-     dw 1474h, 1454h, 1414h, 1400h, 1474h, 1454h, 1474h, 1454h ;t T
00000588 14541474145414
0000058F 791559151915001579-     dw 1579h, 1559h, 1519h, 1500h, 1579h, 1559h, 1579h, 1559h ;y Y
00000598 15591579155915
0000059F 751655161516001675-     dw 1675h, 1655h, 1615h, 1600h, 1675h, 1655h, 1675h, 1655h ;u U
000005A8 16551675165516
000005AF 691749170917001769-     dw 1769h, 1749h, 1709h, 1700h, 1769h, 1749h, 1769h, 1749h ;i I
000005B8 17491769174917
000005BF 6F184F180F1800186F-     dw 186Fh, 184Fh, 180Fh, 1800h, 186Fh, 184Fh, 186Fh, 184Fh ;o O
000005C8 184F186F184F18
000005CF 701950191019001970-     dw 1970h, 1950h, 1910h, 1900h, 1970h, 1950h, 1970h, 1950h ;p P
000005D8 19501970195019
000005DF 5B1A7B1A1B1A00005B-     dw 1A5Bh, 1A7Bh, 1A1Bh, 0000h, 1A5Bh, 1A5Bh, 1A7Bh, 1A7Bh ;[ {
000005E8 1A5B1A7B1A7B1A
000005EF 5D1B7D1B1D1B00005D-     dw 1B5Dh, 1B7Dh, 1B1Dh, 0000h, 1B5Dh, 1B5Dh, 1B7Dh, 1B7Dh ;] }
000005F8 1B5D1B7D1B7D1B
000005FF 0D1C0D1C0A1C00000D-     dw 1C0Dh, 1C0Dh, 1C0Ah, 0000h, 1C0Dh, 1C0Dh, 1C0Ah, 1C0Ah ;Enter
                                                                    (CR/LF)
00000608 1C0D1C0A1C0A1C
0000060F 001D001D001D001D00-     dw 1D00h, 1D00h, 1D00h, 1D00h, 1D00h, 1D00h, 1D00h, 1D00h ;CTRL
                                                                    (left)
00000618 1D001D001D001D
0000061F 611E411E011E001E61-     dw 1E61h, 1E41h, 1E01h, 1E00h, 1E61h, 1E41h, 1E61h, 1E41h ;a A
00000628 1E411E611E411E
0000062F 731F531F131F001F73-     dw 1F73h, 1F53h, 1F13h, 1F00h, 1F73h, 1F53h, 1F73h, 1F53h ;s S
00000638 1F531F731F531F

                                 ; Scancodes 20h-2Fh
                                 ;   base   shift   ctrl   alt    num    caps   shcap  shnum
0000063F 642044200420002064-     dw 2064h, 2044h, 2004h, 2000h, 2064h, 2044h, 2064h, 2044h ;d D
00000648 20442064204420
0000064F 662146210621002166-     dw 2166h, 2146h, 2106h, 2100h, 2166h, 2146h, 2166h, 2146h ;f F
00000658 21462166214621
0000065F 672247220722002267-     dw 2267h, 2247h, 2207h, 2200h, 2267h, 2247h, 2267h, 2247h ;g G
00000668 22472267224722
0000066F 682348230823002368-     dw 2368h, 2348h, 2308h, 2300h, 2368h, 2348h, 2368h, 2348h ;h H
00000678 23482368234823
0000067F 6A244A240A2400246A-     dw 246Ah, 244Ah, 240Ah, 2400h, 246Ah, 244Ah, 246Ah, 244Ah ;j J
00000688 244A246A244A24
0000068F 6B254B250B2500256B-     dw 256Bh, 254Bh, 250Bh, 2500h, 256Bh, 254Bh, 256Bh, 254Bh ;k K
00000698 254B256B254B25
0000069F 6C264C260C2600266C-     dw 266Ch, 264Ch, 260Ch, 2600h, 266Ch, 264Ch, 266Ch, 264Ch ;l L
000006A8 264C266C264C26
```

```
3225 000006AF 3B273A27000000003B-      dw 273Bh, 273Ah, 0000h, 0000h, 273Bh, 273Bh, 273Ah, 273Ah ;; :
3225 000006B8 273B273A273A27
3226 000006BF 2728402800000000027-     dw 2827h, 2840h, 0000h, 0000h, 2827h, 2827h, 2840h, 2840h ;' @
3226 000006C8 28272840284028
3227 000006CF 5C297C29000000005C-      dw 295Ch, 297Ch, 0000h, 0000h, 295Ch, 295Ch, 297Ch, 297Ch ;\ |
3227 000006D8 295C297C297C29
3228 000006DF 002A002A002A002A00-      dw 2A00h, 2A00h, 2A00h, 2A00h, 2A00h, 2A00h, 2A00h, 2A00h ;LShift
                                                                    (2Ah)
3228 000006E8 2A002A002A002A
3229 000006EF 232B7E2B1C2B000023-      dw 2B23h, 2B7Eh, 2B1Ch, 0000h, 2B23h, 2B23h, 2B7Eh, 2B7Eh ;# ~
3229 000006F8 2B232B7E2B7E2B
3230 000006FF 7A2C5A2C1A2C002C7A-      dw 2C7Ah, 2C5Ah, 2C1Ah, 2C00h, 2C7Ah, 2C5Ah, 2C7Ah, 2C5Ah ;z Z
3230 00000708 2C5A2C7A2C5A2C
3231 0000070F 782D582D182D002D78-      dw 2D78h, 2D58h, 2D18h, 2D00h, 2D78h, 2D58h, 2D78h, 2D58h ;x X
3231 00000718 2D582D782D582D
3232 0000071F 632E432E032E002E63-      dw 2E63h, 2E43h, 2E03h, 2E00h, 2E63h, 2E43h, 2E63h, 2E43h ;c C
3232 00000728 2E432E632E432E
3233 0000072F 762F562F162F002F76-      dw 2F76h, 2F56h, 2F16h, 2F00h, 2F76h, 2F56h, 2F76h, 2F56h ;v V
3233 00000738 2F562F762F562F
3234
3235                                   ; Scancodes 30h–3Fh
3236                                   ;    base   shift   ctrl   alt    num    caps   shcap  shnum
3237 0000073F 6230423002300030 62-     dw 3062h, 3042h, 3002h, 3000h, 3062h, 3042h, 3062h, 3042h ;b B
3237 00000748 30423062304230
3238 0000074F 6E314E310E3100316E-      dw 316Eh, 314Eh, 310Eh, 3100h, 316Eh, 314Eh, 316Eh, 314Eh ;n N
3238 00000758 314E316E314E31
3239 0000075F 6D324D320D3200326D-      dw 326Dh, 324Dh, 320Dh, 3200h, 326Dh, 324Dh, 326Dh, 324Dh ;m M
3239 00000768 324D326D324D32
3240 0000076F 2C333C33000000002C-      dw 332Ch, 333Ch, 0000h, 0000h, 332Ch, 332Ch, 333Ch, 333Ch ;, <
3240 00000778 332C333C333C33
3241 0000077F 2E343E34000000002E-      dw 342Eh, 343Eh, 0000h, 0000h, 342Eh, 342Eh, 343Eh, 343Eh ;. >
3241 00000788 342E343E343E34
3242 0000078F 2F353F35000000002F-      dw 352Fh, 353Fh, 0000h, 0000h, 352Fh, 352Fh, 353Fh, 353Fh ;/ ?
3242 00000798 352F353F353F35
3243 0000079F 003600360036003600-      dw 3600h, 3600h, 3600h, 3600h, 3600h, 3600h, 3600h, 3600h ;RShift
3243 000007A8 36003600360036
3244 000007AF 2A370000103700002A-      dw 372Ah, 0000h, 3710h, 0000h, 372Ah, 372Ah, 0000h, 0000h ;KP *
3244 000007B8 372A3700000000
3245 000007BF 003800380038003800-      dw 3800h, 3800h, 3800h, 3800h, 3800h, 3800h, 3800h, 3800h ;Alt
3245 000007C8 38003800380038
3246 000007CF 203920390039000020-      dw 3920h, 3920h, 3900h, 0000h, 3920h, 3920h, 3920h, 3920h ;Space
3246 000007D8 39203920392039
3247 000007DF 003A003A003A003A00-      dw 3A00h, 3A00h, 3A00h, 3A00h, 3A00h, 3A00h, 3A00h, 3A00h ;Caps
                                                                    Lock
3247 000007E8 3A003A003A003A
3248 000007EF 003B0054005E006800-      dw 3B00h, 5400h, 5E00h, 6800h, 3B00h, 3B00h, 5400h, 5400h ;F1
3248 000007F8 3B003B00540054
3249 000007FF 003C0055005F006900-      dw 3C00h, 5500h, 5F00h, 6900h, 3C00h, 3C00h, 5500h, 5500h ;F2
3249 00000808 3C003C00550055
3250 0000080F 003D00560060006A00-      dw 3D00h, 5600h, 6000h, 6A00h, 3D00h, 3D00h, 5600h, 5600h ;F3
3250 00000818 3D003D00560056
3251 0000081F 003E00570061006B00-      dw 3E00h, 5700h, 6100h, 6B00h, 3E00h, 3E00h, 5700h, 5700h ;F4
3251 00000828 3E003E00570057
3252 0000082F 003F00580062006C00-      dw 3F00h, 5800h, 6200h, 6C00h, 3F00h, 3F00h, 5800h, 5800h ;F5
3252 00000838 3F003F00580058
3253
3254                                   ; Scancodes 40h–4Fh
3255                                   ;    base   shift   ctrl   alt    num    caps   shcap  shnum
3256 0000083F 004000590063006D00-      dw 4000h, 5900h, 6300h, 6D00h, 4000h, 4000h, 5900h, 5900h ;F6
3256 00000848 40004000590059
3257 0000084F 0041005A0064006E00-      dw 4100h, 5A00h, 6400h, 6E00h, 4100h, 4100h, 5A00h, 5A00h ;F7
3257 00000858 410041005A005A
3258 0000085F 0042005B0065006F00-      dw 4200h, 5B00h, 6500h, 6F00h, 4200h, 4200h, 5B00h, 5B00h ;F8
3258 00000868 420042005B005B
3259 0000086F 0043005C0066007000-      dw 4300h, 5C00h, 6600h, 7000h, 4300h, 4300h, 5C00h, 5C00h ;F9
3259 00000878 430043005C005C
3260 0000087F 0044005D0067007100-      dw 4400h, 5D00h, 6700h, 7100h, 4400h, 4400h, 5D00h, 5D00h ;F10
3260 00000888 440044005D005D
3261 0000088F 004500450045004500-      dw 4500h, 4500h, 4500h, 4500h, 4500h, 4500h, 4500h, 4500h ;Num Lock
3261 00000898 45004500450045
3262 0000089F 004600460046004600-      dw 4600h, 4600h, 4600h, 4600h, 4600h, 4600h, 4600h, 4600h ;Scroll
                                                                    Lock
3262 000008A8 46004600460046
3263 000008AF 004737470077000037-      dw 4700h, 4737h, 7700h, 0000h, 4737h, 4700h, 4737h, 4700h ;(KP)Home
3263 000008B8 47004737470047
3264 000008BF 004838480000000038-      dw 4800h, 4838h, 0000h, 0000h, 4838h, 4800h, 4838h, 4800h ;(KP)Up
                                                                    arrow
3264 000008C8 48004838480048
3265 000008CF 004939490084000039-      dw 4900h, 4939h, 8400h, 0000h, 4939h, 4900h, 4939h, 4900h ;(KP)PgUp
3265 000008D8 49004939490049
3266 000008DF 2D4A2D4A000000002D-      dw 4A2Dh, 4A2Dh, 0000h, 0000h, 4A2Dh, 4A2Dh, 4A2Dh, 4A2Dh ;(KP)–
3266 000008E8 4A2D4A2D4A2D4A
3267 000008EF 004B344B0073000034-      dw 4B00h, 4B34h, 7300h, 0000h, 4B34h, 4B00h, 4B34h, 4B00h
```

```
                                                                      ;(KP)Left arrow
000008F8 4B004B344B004B
000008FF 004C354C0000000035-     dw 4C00h, 4C35h, 0000h, 0000h, 4C35h, 4C00h, 4C35h, 4C00h
                                                                      ;(KP)Center
00000908 4C004C354C004C
0000090F 004D364D0074000036-     dw 4D00h, 4D36h, 7400h, 0000h, 4D36h, 4D00h, 4D36h, 4D00h
                                                                      ;(KP)Right arrow
00000918 4D004D364D004D
0000091F 2B4E2B4E000000002B-     dw 4E2Bh, 4E2Bh, 0000h, 0000h, 4E2Bh, 4E2Bh, 4E2Bh, 4E2Bh ;(KP)+
00000928 4E2B4E2B4E2B4E
0000092F 004F314F0075000031-     dw 4F00h, 4F31h, 7500h, 0000h, 4F31h, 4F00h, 4F31h, 4F00h ;(KP)End
00000938 4F004F314F004F

                                 ; Scancodes 50h-58h
                                 ;   base   shift   ctrl   alt   num   caps   shcap   shnum
0000093F 005032500000000032-     dw 5000h, 5032h, 0000h, 0000h, 5032h, 5000h, 5032h, 5000h
                                                                      ;(KB)Down arrow
00000948 50005032500050
0000094F 005133510076000033-     dw 5100h, 5133h, 7600h, 0000h, 5133h, 5100h, 5133h, 5100h ;(KB)PgDn
00000958 51005133510051
0000095F 005230520000000030-     dw 5200h, 5230h, 0000h, 0000h, 5230h, 5200h, 5230h, 5200h ;(KB)Ins
00000968 52005230520052
0000096F 00532E53000000002E-     dw 5300h, 532Eh, 0000h, 0000h, 532Eh, 5300h, 532Eh, 5300h ;(KB)Del
00000978 5300532E530053
0000097F 005400540054005400-     dw 5400h, 5400h, 5400h, 5400h, 5400h, 5400h, 5400h, 5400h
                                                                      ;ALT+PRTSC -> Sysreq
00000988 54005400540054
0000098F 000000000000000000-     dw 0000h, 0000h, 0000h, 0000h, 0000h, 0000h, 0000h, 0000h
                                                                      ;xxxxxNOTUSEDxxxxx
00000998 00000000000000
0000099F 5C567C56000000005C-     dw 565Ch, 567Ch, 0000h, 0000h, 565Ch, 565Ch, 567Ch, 567Ch ;\ |
000009A8 565C567C567C56
000009AF 005700000000000000-     dw 5700h, 0000h, 0000h, 0000h, 5700h, 5700h, 0000h, 0000h ;F11
000009B8 57005700000000
000009BF 005800000000000000-     dw 5800h, 0000h, 0000h, 0000h, 5800h, 5800h, 0000h, 0000h ;F12
000009C8 58005800000000
                                 ;--------------------------End of Interrupt--------------------------
                                 ;--------------------Serial Interrupt IRQ 3/Int 23h------------------
                                 ;Serves serial ports 1 and 3 should they exist. Only considers
                                 ; data recieving. Disregards all sending data interrupts.
                                 ;Puts recieved data into respective buffer and clears RTS
                                 ; (base+5) if buffer full.
                                 ;--------------------------------------------------------------------
                                 ser_IRQ3:
000009CF FA                          cli
000009D0 50                          push rax
000009D1 52                          push rdx
000009D2 55                          push rbp
000009D3 51                          push rcx
000009D4 57                          push rdi
000009D5 53                          push rbx

000009D6 BB08000000                  mov ebx, 8
000009DB 66BAFA02                    mov dx, com2_base + 2  ;Interrupt ID register
000009DF EB10                        jmp short ser_common
                                 ;--------------------------End of Interrupt--------------------------
                                 ;-----------------------Serial Interrupt IRQ 3/Int 23h---------------
                                 ;Serves serial ports 1 and 3 should they exist. Only considers
                                 ; data recieving. Disregards all sending data interrupts.
                                 ;Puts recieved data into respective buffer and clears RTS
                                 ; (base+5) if buffer full.
                                 ;--------------------------------------------------------------------
                                 ser_IRQ4:
000009E1 FA                          cli
000009E2 50                          push rax
000009E3 52                          push rdx
000009E4 55                          push rbp
000009E5 51                          push rcx
000009E6 57                          push rdi
000009E7 53                          push rbx

000009E8 BB06000000                  mov ebx, 6
000009ED 66BAFA03                    mov dx, com1_base + 2  ;Interrupt ID register
                                 ser_common:
000009F1 EC                          in al, dx
000009F2 A801                        test al, 1       ;Check if bit zero is clear ie interrupt pending
000009F4 741F                        jz .si1          ;Clear, interrupt pending on COM 1 port
                                 .si0:
000009F6 668B93[67000000]            mov dx, word [com_addresses + rbx]  ;now point to HI COM
                                                                      Interrupt ID registr
000009FD 6685D2                      test dx, dx
00000A00 0F8497000000                jz .siexit              ;Nothing here, exit
00000A06 66FFC2                      inc dx
```

```
00000A09 66FFC2                    inc dx              ;dx = base + 2
00000A0C EC                        in al, dx
00000A0D A801                      test al, 1       ;Check if bit zero is clear
00000A0F 0F8588000000              jnz .siexit      ;Bad behavior, or no Int on com3 after com1
                                                          processed, exit
                               .si1:
                               ;Confirm Data available Interrupt (ie bits 1,2,3 are 010b)
00000A15 A804                      test al, 00000100b
00000A17 0F8480000000              jz .siexit    ;bad behavior, exit
00000A1D 6681C20300                add dx, 3     ;dx = base + 5
                               .si41:
00000A22 EC                        in al, dx
00000A23 2401                      and al, 1
00000A25 74FB                      jz .si41

00000A27 6681EA0500                sub dx, 5
00000A2C EC                        in al, dx     ;get char into al
00000A2D 88C4                      mov ah, al    ;save al in ah temporarily
00000A2F 4831C9                    xor rcx, rcx
                               .si2:     ;Get offset into table structures into cx
00000A32 663B9409[67000000]        cmp dx, word [com_addresses + rcx*2]     ;table of addresses, dx
                                                          is at base
00000A3A 740C                      je .si3
00000A3C 66FFC1                    inc cx
00000A3F 6681F90400                cmp cx, 4     ;rcx should be {0,3}
00000A44 7CEC                      jl .si2
00000A46 EB55                      jmp short .siexit     ;bad value, exit
                               .si3:     ;Store in buffer algorithm
00000A48 488B1CCD[CF000000]        mov rbx, qword [comX_buf_tail + rcx*8]
00000A50 4889DF                    mov rdi, rbx
00000A53 48FFC3                    inc rbx          ;increment by one char
00000A56 483B1CCD[0F010000]        cmp rbx, qword [comX_buf_end + rcx*8]
00000A5E 7508                      jne .si4
00000A60 488B1CCD[EF000000]        mov rbx, qword [comX_buf_start + rcx*8]     ;Wrap around buffer
                               .si4:
00000A68 483B1CCD[AF000000]        cmp rbx, qword [comX_buf_head + rcx*8]     ;Check if buffer full
00000A70 740F                      je .si5     ;Buffer full, indicate wait to data source

00000A72 8827                      mov byte [rdi], ah     ;store char into buffer
00000A74 48891CCD[CF000000]        mov qword [comX_buf_tail + rcx*8], rbx     ;store new tail into
                                                          variable

00000A7C E975FFFFFF                jmp .si0     ;If com1/2, now check that com 3/4 didnt fire
                                                          interrupt.

                               .si5:     ;Buffer full, Deassert DTR bit
                               ;dx points at the base register
00000A81 6681C20400                add dx, 4     ;Point at Modem Control Register
00000A86 EC                        in al, dx
00000A87 24FE                      and al, 11111110b     ;Clear the bottom bit
00000A89 EE                        out dx, al     ;Set the DTR bit down (not ready to recieve data)
00000A8A 6681C20300                add dx, 3     ;Point to scratch register
00000A8F 88E0                      mov al, ah     ;return ah into al
00000A91 EE                        out dx, al     ;put the overrun char into scratch register
00000A92 6681F90200                cmp cx, 2     ;If this was com1/2, now check for com 3/4.
00000A97 0F8559FFFFFF              jne .si0
                               ;exit since we dont want to take whats in the UART buffer just yet.
                               .siexit:
00000A9D B020                      mov al, EOI
00000A9F E620                      out pic1command, al

00000AA1 5B                        pop rbx
00000AA2 5F                        pop rdi
00000AA3 59                        pop rcx
00000AA4 5D                        pop rbp
00000AA5 5A                        pop rdx
00000AA6 58                        pop rax
00000AA7 FB                        sti
00000AA8 48CF                      iretq
                               ;------------------------End of Interrupt------------------------
                               ;-----------------FDD Interrupt IRQ 6/Int 26h---------------------
                               fdd_IRQ6:
00000AAA 50                        push rax
00000AAB B020                      mov al, EOI
00000AAD E620                      out pic1command, al
00000AAF 58                        pop rax
00000AB0 48CF                      iretq
                               ;------------------------End of Interrupt------------------------
                               ;-----------------Spurious Int Handler/Int 27h-------------------
                               ; Catches and handles spurious interrupts on the first pic.
                               ;-----------------------------------------------------------------
                               default_IRQ7:
00000AB2 50                        push rax
```

```
00000AB3 B00B                          mov al, 0Bh      ;Read ISR
00000AB5 E620                          out pic1command, al
00000AB7 E680                          out waitp, al      ;Latch wait
00000AB9 EB00                          jmp short $+2
00000ABB E420                          in al, pic1command      ;Get the ISR
00000ABD A880                          test al, 80h
00000ABF 750A                          jne .exit
00000AC1 66FF0425[20000000]            inc word [spurint1]
00000AC9 EB04                          jmp short .e2      ;Avoid sending EOI
                                   .exit:
00000ACB B020                          mov al, EOI
00000ACD E620                          out pic1command, al
                                   .e2:
00000ACF 58                            pop rax
00000AD0 48CF                          iretq
                                   ;-----------------RTC Interrupt IRQ 8/Int 28h------------------------
                                   ; This IRQ should only trigger for the periodic and alarm
                                   ; interrupts. If a programmer wishes to use the time update
                                   ; complete interrupt feature, they should hook their own
                                   ; interrupt handler.
                                   ;---------------------------------------------------------------------
                                   rtc_IRQ8:
00000AD2 50                            push rax
00000AD3 FA                            cli                 ;Disable interrupts
00000AD4 B08C                          mov al, 8Ch       ;Register C with NMI disabled
00000AD6 E670                          out cmos_base, al
00000AD8 E680                          out waitp, al      ;allow one io cycle to run
00000ADA EB00                          jmp short $+2
00000ADC E471                          in al, cmos_data      ;Get the data byte to confirm IRQ recieved
00000ADE 2460                          and al, 060h           ;Isolate Alarm and Periodic bits only
00000AE0 A840                          test al, 40h           ;Periodic?
00000AE2 7408                          jz .noPeriodic         ;No, skip the periodic
                                   .periodic:
00000AE4 48FF0C25[3B010000]            dec qword [rtc_ticks]
                                   .noPeriodic:
00000AEC A820                          test al, 20h           ;Alarm?
00000AEE 7402                          jz .exit
                                   .alarm:
00000AF0 CD6A                          int 6Ah      ;User Alarm handler, behaves like Int 4Ah on 16-bit BIOS
                                   .exit:
00000AF2 B00D                          mov al, 0Dh       ;Read Register D and reenable NMI
00000AF4 E670                          out cmos_base, al
00000AF6 E680                          out waitp, al      ;allow one io cycle to run
00000AF8 EB00                          jmp short $+2
00000AFA E471                          in al, cmos_data
00000AFC B020                          mov al, EOI
00000AFE E6A0                          out pic2command, al
00000B00 E620                          out pic1command, al
00000B02 58                            pop rax
00000B03 48CF                          iretq
                                   ;---------------------------End of Interrupt----------------------------
                                   ;-----------------HDD Interrupt IRQ 14/Int 2Eh------------------------
                                   hdd_IRQ14:
00000B05 50                            push rax
00000B06 C60425[AB010000]00            mov byte [ir14_mutex], 0
00000B0E B020                          mov al, EOI
00000B10 E620                          out pic1command, al
00000B12 58                            pop rax
00000B13 48CF                          iretq
                                   ;---------------------------End of Interrupt----------------------------
                                   ;--------------------Spurious Int Handler/Int 2Fh----------------------
                                   ; Catches and handles spurious interrupts on the second pic.
                                   ;---------------------------------------------------------------------
                                   default_IRQ15:
00000B15 50                            push rax
00000B16 803C25[AD010000]01            cmp byte [ir15_mutex], 1      ;Check if mutex set
00000B1E 7508                          jne .spurcheck                ;If not set, then just check spur
00000B20 C60425[AD010000]00            mov byte [ir15_mutex], 0      ;Exit and check spur
                                   .spurcheck:
00000B28 B00B                          mov al, 0Bh      ;Read ISR
00000B2A E6A0                          out pic2command, al
00000B2C E680                          out waitp, al      ;Latch wait
00000B2E EB00                          jmp short $+2
00000B30 E4A0                          in al, pic2command      ;Get the ISR
00000B32 A880                          test al, 80h
00000B34 B020                          mov al, EOI      ;Still need to send EOI to pic1
00000B36 750A                          jne .exit
00000B38 66FF0425[21000000]            inc word [spurint2]
00000B40 EB02                          jmp short .e2      ;Avoid sending EOI
                                   .exit:
00000B42 E6A0                          out pic2command, al
                                   .e2:
```

```
00000B44 E620                        out pic1command, al
00000B46 58                          pop rax
00000B47 48CF                        iretq
                                 ;——————————————————End of Interrupt——————————————————
                                 ;———————————————EHCI Int Handler/Int 2Xh———————————————
                                 ;This is installed by the PCI proc at runtime, onto the
                                 ; appropriate IRQ.
                                 ;
                                 ;If USB Host controller is doing transaction, this HC is
                                 ; nominally turned off. Bits [7:2] in the eAsyncMutex identify
                                 ; how many interrupts to ignore, before switching off the
                                 ; Schedule. This value is nominally zero.
                                 ;——————————————————————————————————————————————————————
                                 ehci_IRQ:
00000B49 68[244F0000]                push qword dummy_interrupt.pic2
00000B4E EB05                        jmp short .intr
                                 .pic1:
00000B50 68[2B4F0000]                push qword dummy_interrupt.pic1
                                 .intr:
                                 ;EHCI Interrupt Handler
00000B55 53                          push rbx
00000B56 50                          push rax

00000B57 8A0425[47020000]            mov al, byte [eActiveCtrlr]
00000B5E 3CFF                        cmp al, -1     ;Spurious case, replace with manual poll then discard proc
00000B60 743F                        je .spur

00000B62 E8903F0000                  call USB.ehciGetOpBase     ;returns opreg base in rax
                                 .nonIRQmain:
00000B67 678B5804                    mov ebx, dword [eax + ehcists]   ;save USBSTS and clear usb interrupt
00000B6B 67095804                    or dword [eax + ehcists], ebx    ;WC all interrupt status
00000B6F 881C25[48020000]            mov byte [eActiveInt], bl      ;save interrupt status

                                 ;Test based on which bits are set. Higher bits have higher priority
                                     ;test bl, 10h            ;Check if host error bit set
                                     ;test bl, 8              ;Frame List rollover
                                     ;test bl, 4              ;Port status change detected
00000B76 F6C302                      test bl, 2              ;Check if transation error bit is set
00000B79 7542                        jnz .transactionError
00000B7B F6C301                      test bl, 1              ;Check if short packet/interrupt bit set
00000B7E 741E                        jz .exit                ;If none of the bits were set, continue IRQ chain
                                 ;IoC and Short Packet section
00000B80 8A0425[49020000]            mov al, byte [eAsyncMutex]      ;check if we should ignore interrupt
00000B87 24FC                        and al, 11111100b     ;clear out bottom two bits (dont care)
00000B89 84C0                        test al, al             ;Set zero flag if al is zero
00000B8B 7509                        jnz .usbignoreirq     ;If not zero, ignore irq (and dec counter!)

00000B8D 880425[49020000]            mov byte [eAsyncMutex], al ;Wait no longer!! Data available

00000B94 EB08                        jmp short .exit      ;Ignore the "ignore usb" section
                                 .usbignoreirq:
00000B96 802C25[49020000]04          sub byte [eAsyncMutex], 4      ;sub the semaphore
                                 .exit:
00000B9E 58                          pop rax
00000B9F 5B                          pop rbx
00000BA0 C3                          ret

                                 .spur:
00000BA1 30C0                        xor al, al
                                 .s1:
00000BA3 E84F3F0000                  call USB.ehciGetOpBase
00000BA8 678B5804                    mov ebx, dword [eax + ehcists]  ;save USBSTS and clear usb interrupt
00000BAC 67095804                    or dword [eax + ehcists], ebx      ;WC all interrupt status
00000BB0 FEC0                        inc al     ;Clear all interrupts on all controllers
00000BB2 3A0425[14020000]            cmp al, byte [eControllers]
00000BB9 72E8                        jb .s1
00000BBB EBE1                        jmp short .exit
                                 .transactionError:
00000BBD C60425[49020000]00          mov byte [eAsyncMutex], 0     ;Unblock wait
00000BC5 EBD7                        jmp short .exit
                                 .nonIRQep:
00000BC7 53                          push rbx
00000BC8 50                          push rax
00000BC9 EB9C                        jmp short .nonIRQmain
                                 ;——————————————————End of Interrupt——————————————————
                                 ;══════════════════SOFTWARE INTERRUPTS══════════════════
                                 ;————————————————Video Interrupt Int 30h————————————————
                                 scr_io_table:
00000BCB [980C000000000000]          dq      scr_io.change_mode       ;AH = 0 -> Change Screen Mode
```

```
                                                                  (Currently no
                                                                   ; options)
00000BD3 [A70C000000000000]          dq     scr_io.set_curs_shape   ;AH = 1 -> Set Cursor Shape
00000BDB [BA0C000000000000]          dq     scr_io.set_curs_pos     ;AH = 2 -> Set Cursor Position
00000BE3 [DE0C000000000000]          dq     scr_io.get_curs_pos     ;AH = 3 -> Get Cursor Position
00000BEB [FE0C000000000000]          dq     scr_io.write_register   ;AH = 4 -> Reserved, Undoc, Write
                                                                  al in ASCII
                                                                   ; at cursor
00000BF3 [470D000000000000]          dq     scr_io.select_page      ;AH = 5 -> Select Active Page
00000BFB [ED0D000000000000]          dq     scr_io.scroll_up        ;AH = 6 -> Scroll Active Page up
00000C03 [AB0E000000000000]          dq     scr_io.scroll_down      ;AH = 7 -> Scroll Active Page down
00000C0B [3D0F000000000000]          dq     scr_io.read_att_char    ;AH = 8 -> Read Attribute and Char
                                                                  at curs pos
00000C13 [6F0F000000000000]          dq     scr_io.write_att_char   ;AH = 9 -> Write Attribute and
                                                                  Char at curs pos
00000C1B [AF0F000000000000]          dq     scr_io.write_char       ;AH = 0Ah -> Write Char at curs
                                                                  position
                                                                   ; (using default attribute)
00000C23 [F70F000000000000]          dq     scr_io.gset_col_palette ;AH = 0Bh -> Graphics, Set Colour
                                                                  Palette
00000C2B [0610000000000000]          dq     scr_io.gwritedot        ;AH = 0Ch -> Graphics, Write a Dot
                                                                  to screen
00000C33 [1510000000000000]          dq     scr_io.greaddot         ;AH = 0Dh -> Graphics, Read a Dot
                                                                  from screen
00000C3B [2410000000000000]          dq     scr_io.write_tty        ;AH = 0Eh -> Write Teletype
00000C43 [F210000000000000]          dq     scr_io.get_mode         ;AH = 0Fh -> Get Screen Mode
                                                                  (currently, no
                                                                   ; options)
00000C4B [8D0C000000000000]          dq     scr_io.exitf            ;AH = 10h -> Reserved
00000C53 [8D0C000000000000]          dq   scr_io.exitf              ;AH = 11h -> Reserved
00000C5B [8D0C000000000000]          dq     scr_io.exitf            ;AH = 12h -> Reserved
00000C63 [1B11000000000000]          dq   scr_io.write_string       ;AH = 13h -> Write string
                                 scr_io_table_length     equ     $ - scr_io_table
                                 scr_io:
00000C6B FC                          cld          ;set direction to read the right way
00000C6C 56                          push rsi
00000C6D 50                          push rax
00000C6E C0E403                      shl ah, 3  ;Use ah as offset into table
00000C71 80FC98                      cmp ah, (scr_io_table_length - 8)     ;Ensure function number is
                                                                  within table
00000C74 7717                        ja .exitf
00000C76 88E0                        mov al, ah
00000C78 480FB6C0                    movzx rax, al                  ;Zero extend ax into rax
00000C7C 4889C6                      mov rsi, rax                   ;Note rsi is not being saved here!
00000C7F 58                          pop rax                        ;recover back into ax
00000C80 8A2425[58010000]            mov ah, byte [scr_mode]        ;Get the current mode into ah
00000C87 FFA6[CB0B0000]              jmp [scr_io_table + rsi]       ;Jump to correct function
                                 .exitf:
00000C8D 58                          pop rax
00000C8E B480                        mov ah, 80h ;Function not supported
00000C90 804C241801                  or byte [rsp + 3*8h], 1 ;Set Carry flag, invalid function, skip
                                                                  rsi on stack
                                 .exit:
00000C95 5E                          pop rsi
00000C96 48CF                        iretq


                                 .change_mode:
00000C98 48B8FFFF0000000000-         mov rax, 0FFFFh
00000CA1 00
00000CA2 E9EEFFFFFF                  jmp .exit     ;Currently unsupported function
                                 .set_curs_shape:
                                 ;Input: CH = Scan Row Start, CL = Scan Row End
00000CA7 52                          push rdx
00000CA8 66890C25[55010000]          mov word [scr_curs_shape], cx

00000CB0 B00A                        mov al, 0Ah
00000CB2 E86B050000                  call .write_crtc_word

00000CB7 5A                          pop rdx
00000CB8 EBDB                        jmp short .exit
                                 .set_curs_pos:
                                 ;Input: DH = Row, DL = Column, BH = active page
00000CBA 51                          push rcx
00000CBB 52                          push rdx

00000CBC 53                          push rbx
00000CBD 88FB                        mov bl, bh
00000CBF 480FB6DB                    movzx rbx, bl
00000CC3 6689941B[43010000]          mov word [scr_curs_pos + 2*rbx], dx
00000CCB 5B                          pop rbx
00000CCC 3A3C25[59010000]            cmp bh, byte [scr_active_page]
00000CD3 7505                        jne .scpexit     ;if the page is not the active page
00000CD5 E8E3050000                  call .cursor_proc
```

```
3640                                    .scpexit:
3641 00000CDA 5A                            pop rdx
3642 00000CDB 59                            pop rcx
3643 00000CDC EBB7                          jmp short .exit
3644
3645
3646                                    .get_curs_pos:
3647                                    ;Return: AX = 0, CH = Start scan line, CL = End scan line, DH =
                                                                   Row, DL = Column
3648 00000CDE 53                            push rbx
3649
3650 00000CDF 88FB                          mov bl, bh
3651 00000CE1 480FB6DB                      movzx rbx, bl
3652 00000CE5 668B941B[43010000]            mov dx, word [scr_curs_pos + 2*rbx]
3653 00000CED 668B0C25[55010000]            mov cx, word [scr_curs_shape]     ;Get cursor shape
3654
3655 00000CF5 5B                            pop rbx
3656 00000CF6 6631C0                        xor ax, ax
3657 00000CF9 E997FFFFFF                    jmp .exit
3658
3659                                    .write_register:     ;al contains the byte to convert
3660 00000CFE 52                            push rdx
3661 00000CFF 53                            push rbx
3662 00000D00 50                            push rax
3663
3664 00000D01 88C2                          mov dl, al              ;save byte in dl
3665 00000D03 6625F000                      and ax, 00F0h           ;Hi nybble
3666 00000D07 6681E20F00                    and dx, 000Fh           ;Lo nybble
3667 00000D0C 66C1E804                      shr ax, 4               ;shift one hex place value pos right
3668 00000D10 E810000000                    call .wrchar
3669 00000D15 6689D0                        mov ax, dx              ;mov lo nybble, to print
3670 00000D18 E808000000                    call .wrchar
3671
3672 00000D1D 58                            pop rax
3673 00000D1E 5B                            pop rbx
3674 00000D1F 5A                            pop rdx
3675 00000D20 E970FFFFFF                    jmp .exit
3676                                    .wrchar:
3677 00000D25 48BB-                         mov rbx, .wrascii
3677 00000D27 [370D000000000000]
3678 00000D2F D7                            xlatb    ;point al to entry in ascii table, using al as offset
                                                                    into table
3679 00000D30 B40E                          mov ah, 0Eh
3680 00000D32 B307                          mov bl, 07h
3681 00000D34 CD30                          int 30h  ;print char
3682 00000D36 C3                            ret
3683 00000D37 303132333435363738-       .wrascii:    db     '0123456789ABCDEF'
3683 00000D40 39414243444546
3684                                    .select_page:
3685                                    ;ah contains the current screen mode
3686                                    ;al contains new screen page
3687                                    ;vga just returns as invalid FOR NOW
3688                                    ;Handled differently between vga and classic modes
3689 00000D47 80FC04                        cmp ah, 04
3690 00000D4A 761D                          jbe .sp1
3691 00000D4C 80FC07                        cmp ah, 07
3692 00000D4F 7418                          je .sp1
3693 00000D51 80FC0D                        cmp ah, 0Dh
3694 00000D54 0F838E000000                  jae .sp_vga
3695                                    .spbad:
3696 00000D5A 48B8FFFF0000000000-           mov rax, 0FFFFh
3696 00000D63 00
3697 00000D64 E92CFFFFFF                    jmp .exit     ;Bad argument
3698                                    .sp1:
3699 00000D69 3C08                          cmp al, 8
3700 00000D6B 73ED                          jae .spbad     ;page should be 0-7
3701                                    .spmain:
3702 00000D6D 50                            push rax
3703 00000D6E 53                            push rbx
3704 00000D6F 51                            push rcx
3705 00000D70 52                            push rdx
3706 00000D71 880425[59010000]              mov byte [scr_active_page], al     ;change active page
3707                                    ;------Modify this proc with data tables when finalised!!----
3708 00000D78 48BE0008000000000000-         mov rsi, 800h      ;mode 0,1 page size
3708 00000D81 00
3709 00000D82 48BB0010000000000000-         mov rbx, 1000h      ;mode 2,3,7 page size
3709 00000D8B 00
3710 00000D8C 480FB6C8                      movzx rcx, al     ;Get count into rcx
3711 00000D90 80FC02                        cmp ah, 2
3712 00000D93 480F42DE                      cmovb rbx, rsi
3713 00000D97 48BA00800B00000000-           mov rdx, vga_bpage2
3713 00000DA0 00
3714 00000DA1 48BE00000B00000000-           mov rsi, vga_bpage1     ;Base addr for mode 7
```

```
00000DAA 00
                                  ;———Modify this proc with data tables when finalised!!————
00000DAB 80FC07                       cmp ah, 7
00000DAE 480F44D6                     cmove rdx, rsi
00000DB2 52                           push rdx      ;Push the saved page 0 address
00000DB3 E307                         jrcxz .spm2      ;If 0th page, dont add
                                  .spm1:
00000DB5 4801DA                       add rdx, rbx
00000DB8 FEC9                         dec cl
00000DBA 75F9                         jnz .spm1
                                  .spm2:
00000DBC 5E                           pop rsi       ;Get saved base into rsi
00000DBD 891425[5C010000]             mov dword [scr_page_addr], edx      ;Get new base addr
00000DC4 4829F2                       sub rdx, rsi      ;rsi has conditionally b8000 or b0000
00000DC7 50                           push rax
00000DC8 66D1EA                       shr dx, 1      ;Divide dx by 2 to get # of PELs
00000DCB 6689D1                       mov cx, dx      ;Get offset from crtc base addr
00000DCE 66B80C00                     mov ax, 0Ch      ;6845 Start Addr register
00000DD2 E84B040000                   call .write_crtc_word      ;Change ”crtc view window”

00000DD7 58                           pop rax          ;Get original ax back for page number
00000DD8 88C7                         mov bh, al
00000DDA E8DE040000                   call .cursor_proc      ;Move cursor on page

00000DDF 5A                           pop rdx
00000DE0 59                           pop rcx
00000DE1 5B                           pop rbx
00000DE2 58                           pop rax
00000DE3 E9ADFEFFFF                   jmp .exit      ;Bad argument
                                  .sp_vga:
00000DE8 E96DFFFFFF                   jmp .spbad

                                  .scroll_up:
                                  ;Scrolls ACTIVE SCREEN only
                                  ;Called with AL=number of lines to scroll, BH=Attribute for new area
                                  ;     CH=ycor of top of scroll, CL=xcor of top of scroll
                                  ;     DH=ycor of bottom of scroll, DL=xcor of bottom of scroll
                                  ;If AL=0 then entire window is blanked, BH is used for blank attrib
                                  ;ah contains the current screen mode
00000DED 80FC04                       cmp ah, 04      ;Test for Alpha mode
00000DF0 7209                         jb .su0
00000DF2 80FC07                       cmp ah, 07      ;Test for MDA Alpha mode
00000DF5 0F8509040000                 jne .gscrollup      ;We in graphics mode, go to correct proc
                                  .su0:
00000DFB 55                           push rbp
00000DFC 57                           push rdi
00000DFD 50                           push rax      ;Treat AX more or less as clobbered

00000DFE 84C0                         test al, al     ;Check if zero
00000E00 747F                         je .sblank      ;recall ah=06 then reset cursor and exit
00000E02 88C3                         mov bl, al      ;Save number of lines to scroll in bl
                                  .su1:
00000E04 8B3425[5C010000]             mov esi, dword [scr_page_addr]      ;zeros upper dword
00000E0B 4889F7                       mov rdi, rsi    ;Point both pointers at base of active page
00000E0E 6689C8                       mov ax, cx      ;Bottom top corner into ax
00000E11 E870040000                   call .offset_from_ax      ;Get the page offset of dx
00000E16 480FB7C0                     movzx rax, ax
00000E1A 48D1E0                       shl rax, 1      ;Multiply by two for words
00000E1D 4801C7                       add rdi, rax    ;point to the top left of window
00000E20 4801C6                       add rsi, rax
00000E23 480FB60425-                  movzx rax, byte [scr_cols]
00000E28 [53010000]
00000E2C 48D1E0                       shl rax, 1        ;number of columns * 2 for words!
00000E2F 4801C6                       add rsi, rax      ;Point rsi one row down
00000E32 51                           push rcx
00000E33 52                           push rdx

00000E34 28EE                         sub dh, ch      ;work out number of rows to copy
                                  .su2:
00000E36 56                           push rsi
00000E37 57                           push rdi
00000E38 E865040000                   call .text_scroll_c1      ;Scroll the selected row
00000E3D 5F                           pop rdi
00000E3E 5E                           pop rsi
00000E3F 4801C7                       add rdi, rax      ;goto next row
00000E42 4801C6                       add rsi, rax
00000E45 FECE                         dec dh
00000E47 75ED                         jnz .su2

00000E49 5A                           pop rdx
00000E4A 59                           pop rcx
                                  ;Draw blank line
00000E4B 50                           push rax
```

```
00000E4C 51                         push rcx
00000E4D 57                         push rdi

00000E4E 6689C8                     mov ax, cx
00000E51 88F4                       mov ah, dh      ;Starting column from cx, starting row from dx
00000E53 E82E040000                 call .offset_from_ax
00000E58 8B3C25[5C010000]           mov edi, dword [scr_page_addr]
00000E5F 480FB7C0                   movzx rax, ax
00000E63 48D1E0                     shl rax, 1
00000E66 01C7                       add edi, eax    ;point to new line
00000E68 88FC                       mov ah, bh
00000E6A B020                       mov al, 20h     ;Blank char
00000E6C 4889E9                     mov rcx, rbp    ;move word count into cx
00000E6F F366AB                     rep stosw       ;write the word bp number of times
00000E72 5F                         pop rdi
00000E73 59                         pop rcx
00000E74 58                         pop rax
00000E75 FECB                       dec bl
00000E77 758B                       jnz .su1     ;Once we have done bl rows, exit


                                .suexit:
00000E79 58                         pop rax
00000E7A 5F                         pop rdi
00000E7B 5D                         pop rbp
00000E7C E914FEFFFF                 jmp .exit
                                .sblank:
                                ;Fast clear function
00000E81 51                         push rcx
00000E82 52                         push rdx

00000E83 88FC                       mov ah, bh      ;mov attrib into ah
00000E85 B020                       mov al, 20h     ;Space char
00000E87 8B3C25[5C010000]           mov edi, dword [scr_page_addr]
00000E8E 480FB61425-                movzx rdx, byte [scr_rows]
00000E93 [54010000]

                                .sbl0:
00000E97 480FB60C25-                movzx rcx, byte [scr_cols]
00000E9C [53010000]
00000EA0 F366AB                     rep stosw
00000EA3 FECA                       dec dl
00000EA5 75F0                       jnz .sbl0

00000EA7 5A                         pop rdx
00000EA8 59                         pop rcx
00000EA9 EBCE                       jmp short .suexit

                                .scroll_down:
                                ;Scrolls ACTIVE SCREEN only
                                ;Called with AL=number of lines to scroll, BH=Attribute for new area
                                ;     CH=ycor of top of scroll, CL=xcor of top of scroll
                                ;     DH=ycor of bottom of scroll, DL=xcor of bottom of scroll
                                ;If AL=0 then entire window is blanked, BH is used for blank attrib
                                ;ah contains the current screen mode
00000EAB 80FC04                     cmp ah, 04      ;Test for Alpha mode
00000EAE 7209                       jb .sd0
00000EB0 80FC07                     cmp ah, 07      ;Test for MDA Alpha mode
00000EB3 0F855A030000               jne .gscrolldown      ;We in graphics mode, go to correct proc

                                .sd0:
00000EB9 55                         push rbp
00000EBA 57                         push rdi
00000EBB 50                         push rax     ;Treat AX more or less as clobbered

00000EBC 84C0                       test al, al     ;Check if zero
00000EBE 74C1                       je .sblank      ;recall ah=06 then reset cursor and exit
00000EC0 88C3                       mov bl, al      ;Save number of lines to scroll in bl
00000EC2 FD                         std      ;change the direction of string operations

                                .sd1:
00000EC3 8B3425[5C010000]           mov esi, dword [scr_page_addr]      ;point esi to bottom
00000ECA 6689D0                     mov ax, dx      ;point to bottom right
00000ECD E8B4030000                 call .offset_from_ax
00000ED2 480FB7C0                   movzx rax, ax
00000ED6 48D1E0                     shl rax, 1
00000ED9 4801C6                     add rsi, rax
00000EDC 4889F7                     mov rdi, rsi
00000EDF 480FB60425-                movzx rax, byte [scr_cols]
00000EE4 [53010000]
00000EE8 48D1E0                     shl rax, 1
00000EEB 4829C6                     sub rsi, rax     ;Point rsi one row above rdi

00000EEE 51                         push rcx
00000EEF 52                         push rdx
00000EF0 28EE                       sub dh, ch      ;Number of rows to copy
                                .sd2:
```

```
00000EF2 56                             push rsi
00000EF3 57                             push rdi
00000EF4 E8A9030000                     call .text_scroll_c1
00000EF9 5F                             pop rdi
00000EFA 5E                             pop rsi
00000EFB 4829C7                         sub rdi, rax
00000EFE 4829C6                         sub rsi, rax
00000F01 FECE                           dec dh
00000F03 75ED                           jnz .sd2

00000F05 5A                             pop rdx
00000F06 59                             pop rcx
                                    ;Draw blank line
00000F07 50                             push rax
00000F08 51                             push rcx
00000F09 57                             push rdi

00000F0A 6689D0                         mov ax, dx
00000F0D 88EC                           mov ah, ch     ;Starting column from dx, starting row from cx
00000F0F E872030000                     call .offset_from_ax
00000F14 8B3C25[5C010000]               mov edi, dword [scr_page_addr]
00000F1B 480FB7C0                       movzx rax, ax
00000F1F 48D1E0                         shl rax, 1
00000F22 01C7                           add edi, eax     ;Point to appropriate line and col
00000F24 88FC                           mov ah, bh
00000F26 B020                           mov al, 20h
00000F28 4889E9                         mov rcx, rbp
00000F2B F366AB                         rep stosw     ;Store backwards
00000F2E 5F                             pop rdi
00000F2F 59                             pop rcx
00000F30 58                             pop rax
00000F31 FECB                           dec bl
00000F33 758E                           jnz .sd1


                                    .sdexit:
00000F35 58                             pop rax
00000F36 5F                             pop rdi
00000F37 5D                             pop rbp
00000F38 E958FDFFFF                     jmp .exit
                                    .read_att_char:
                                    ;Get ASCII char and attr at current cursor position on chosen page
                                    ;Called with AH=08h, BH=Page number (if supported),
                                    ;Returns, AH=Attrib, AL=Char


                                    ;On entry, ah contains current screen mode
00000F3D 80FC04                         cmp ah, 04      ;Test for Alpha mode
00000F40 7209                           jb .rac1
00000F42 80FC07                         cmp ah, 07      ;Test for MDA Alpha mode
00000F45 0F859B020000                   jne .gread      ;We in graphics mode, go to correct proc
                                    .rac1:
00000F4B 80FF07                         cmp bh, 7
00000F4E 0F8739FDFFFF                   ja .exitf      ;All A/N modes can have 8 pages, any more, fail

00000F54 88E3                           mov bl, ah      ;Move screen mode into bl for function call
00000F56 8B3425[5C010000]               mov esi, dword [scr_page_addr]
00000F5D E814030000                     call .page_cursor_offset      ;bx preserved
00000F62 48D1E0                         shl rax, 1
00000F65 4801C6                         add rsi, rax      ;rsi should point to attrib/char
00000F68 66AD                           lodsw             ;Load ah with attrib/char
00000F6A E926FDFFFF                     jmp .exit      ;Restoring rsi

                                    .write_att_char:
                                    ;Puts ASCII char and attribute/colour at cursor
                                    ;Called with AH=09h, AL=Char, BH=Page,
                                    ;     BL=Attrib/Color, CX=number of repeats
                                    ;Returns nothing (just prints in page)


                                    ;When called, ah contains current screen mode
00000F6F 80FC04                         cmp ah, 04      ;Test for Alpha mode
00000F72 7209                           jb .wac1
00000F74 80FC07                         cmp ah, 07      ;Test for MDA Alpha mode
00000F77 0F8578020000                   jne .gwrite      ;We in graphics mode, go to correct proc
                                    .wac1:
00000F7D 80FF07                         cmp bh, 7
00000F80 0F8707FDFFFF                   ja .exitf      ;All A/N modes can have 8 pages, any more, fail

00000F86 86DC                           xchg bl, ah ;swap attrib and scr mode bytes
00000F88 57                             push rdi
00000F89 50                             push rax      ;Save the char/attrib word
00000F8A 8B3425[5C010000]               mov esi, dword [scr_page_addr]
00000F91 E8E0020000                     call .page_cursor_offset      ;bx preserved
00000F96 4889F7                         mov rdi, rsi      ;Change register for string ops
00000F99 48D1E0                         shl rax, 1
```

```
00000F9C 4801C7                       add rdi, rax     ;rsi now points to right place on right page
00000F9F 58                           pop rax

00000FA0 51                           push rcx
00000FA1 480FB7C9                     movzx rcx, cx    ;zero upper bytes
00000FA5 F366AB                       rep stosw        ;Store packed ah/al cx times
00000FA8 59                           pop rcx
00000FA9 5F                           pop rdi
00000FAA E9E6FCFFFF                   jmp .exit     ;Restoring rsi

                                  .write_char:
                                  ;Puts ASCII char and attribute/colour at cursor
                                  ;Called with AH=0Ah, AL=Char, BH=Page,
                                  ;     BL=Color (G modes ONLY), CX=number of repeats
                                  ;Returns nothing (just prints in page)
00000FAF 80FC04                       cmp ah, 04     ;Test for Alpha mode
00000FB2 7209                         jb .wc1
00000FB4 80FC07                       cmp ah, 07     ;Test for MDA Alpha mode
00000FB7 0F8538020000                 jne .gwrite     ;We in graphics mode, go to correct proc
                                  .wc1:
00000FBD 80FF07                       cmp bh, 7
00000FC0 0F87C7FCFFFF                 ja .exitf     ;All A/N modes can have 8 pages, any more, fail

00000FC6 88E3                         mov bl, ah ;mov scr mode byte into bl
00000FC8 57                           push rdi
00000FC9 50                           push rax     ;Save the char word
00000FCA 8B3425[5C010000]             mov esi, dword [scr_page_addr]
00000FD1 E8A0020000                   call .page_cursor_offset     ;bx preserved
00000FD6 4889F7                       mov rdi, rsi     ;Change register for string ops
00000FD9 48D1E0                       shl rax, 1
00000FDC 4801C7                       add rdi, rax     ;rdi now points to right place on right page
00000FDF 58                           pop rax

00000FE0 51                           push rcx
00000FE1 480FB7C9                     movzx rcx, cx     ;zero upper bytes
00000FE5 E309                         jrcxz .wc3     ;If cx is zero, dont print anything, exit
                                  .wc2:
00000FE7 AA                           stosb
00000FE8 48FFC7                       inc rdi
00000FEB 48FFC9                       dec rcx
00000FEE 75F7                         jnz .wc2
                                  .wc3:
00000FF0 59                           pop rcx
00000FF1 5F                           pop rdi
00000FF2 E99EFCFFFF                   jmp .exit     ;Exit restoring rsi


                                  .gset_col_palette:
00000FF7 48B8FFFF0000000000-          mov rax, 0FFFFh
00001000 00
00001001 E98FFCFFFF                   jmp .exit     ;Currently unsupported function

                                  .gwritedot:
00001006 48B8FFFF0000000000-          mov rax, 0FFFFh
0000100F 00
00001010 E980FCFFFF                   jmp .exit     ;Currently unsupported function

                                  .greaddot:
00001015 48B8FFFF0000000000-          mov rax, 0FFFFh
0000101E 00
0000101F E971FCFFFF                   jmp .exit     ;Currently unsupported function

                                  .write_tty:
                                  ;Called with al=char, bl=foreground color (graphics)
                                  ;When called, ah contains current screen mode
00001024 51                           push rcx
00001025 52                           push rdx
00001026 53                           push rbx
00001027 50                           push rax

00001028 8A3C25[59010000]             mov bh, byte [scr_active_page]     ;Get active page
0000102F 50                           push rax
00001030 B403                         mov ah, 3     ;Get cursor into dx
00001032 CD30                         int 30h
00001034 58                           pop rax

00001035 3C08                         cmp al, 08h     ;Check for backspace
00001037 746B                         je .wttybspace
00001039 3C0A                         cmp al, 0Ah     ;Check for line feed
0000103B 747F                         je .wttylf
0000103D 3C0D                         cmp al, 0Dh     ;Check for carriage return
0000103F 0F848D000000                 je .wttycr
00001045 3C07                         cmp al, 07h     ;ASCII bell
00001047 0F848C000000                 je .wttybell

                                  .wttywrite:
```

```
4039 0000104D 48B901000000000000-          mov rcx, 1
4039 00001056 00
4040 00001057 B40A                         mov ah, 0Ah     ;Write 1 char w/o attrib byte
4041 00001059 CD30                         int 30h     ;bh contains page to write for
4042
4043                                   .wttycursorupdate:
4044 0000105B FEC2                         inc dl
4045 0000105D 3A1425[53010000]             cmp dl, byte [scr_cols]
4046 00001064 730D                         jae .wttycu0     ;go down by a line, and start of the line
4047                                   .wttycursorupdatego:
4048 00001066 B402                         mov ah, 2
4049 00001068 CD30                         int 30h      ;set cursor
4050                                   .wttyexit:
4051 0000106A 58                           pop rax
4052 0000106B 5B                           pop rbx
4053 0000106C 5A                           pop rdx
4054 0000106D 59                           pop rcx
4055 0000106E E922FCFFFF                   jmp .exit
4056
4057                                   .wttycu0:
4058 00001073 30D2                         xor dl, dl     ;Return to start of line
4059 00001075 FEC6                         inc dh
4060 00001077 3A3425[54010000]             cmp dh, byte [scr_rows]     ;are past the bottom of the screen?
4061 0000107E 72E6                         jb .wttycursorupdatego     ;we are not past the bottom of the
                                                                        screen
4062                                   .wttyscrollupone:
4063 00001080 53                           push rbx
4064 00001081 B408                         mov ah, 08h     ;Read char/attrib at cursor
4065 00001083 CD30                         int 30h
4066 00001085 88E7                         mov bh, ah     ;Move attrib byte into bh
4067 00001087 4831C9                       xor rcx, rcx
4068 0000108A 668B1425[53010000]           mov dx, word [scr_cols]     ;word access all ok
4069 00001092 FECE                         dec dh
4070 00001094 FECA                         dec dl
4071 00001096 66B80106                     mov ax, 0601h     ;scroll up one line
4072 0000109A CD30                         int 30h
4073
4074 0000109C 30D2                         xor dl, dl
4075 0000109E 5B                           pop rbx
4076 0000109F E9C2FFFFFF                   jmp .wttycursorupdatego
4077                                   .wttybspace:
4078 000010A4 84D2                         test dl, dl     ;compare if the column is zero
4079 000010A6 750D                         jnz .wttybs1    ;if not just decrement row pos
4080 000010A8 84F6                         test dh, dh     ;compare if zero row, if so do nothing
4081 000010AA 74BE                         jz .wttyexit    ;at top left, just exit
4082 000010AC FECE                         dec dh
4083 000010AE 8A1425[53010000]             mov dl, byte [scr_cols]     ;move to end of prev row + 1
4084                                   .wttybs1:
4085 000010B5 FECA                         dec dl
4086 000010B7 E9AAFFFFFF                   jmp .wttycursorupdatego
4087
4088                                   .wttylf:
4089 000010BC 52                           push rdx
4090 000010BD 8A1425[54010000]             mov dl, byte [scr_rows]
4091 000010C4 FECA                         dec dl
4092 000010C6 38D6                         cmp dh, dl
4093 000010C8 5A                           pop rdx
4094 000010C9 74B5                         je .wttyscrollupone     ;if we need to scroll, scroll
4095 000010CB FEC6                         inc dh     ;otherwise just send cursor down by one
4096 000010CD E994FFFFFF                   jmp     .wttycursorupdatego
4097                                   .wttycr:
4098 000010D2 B200                         mov dl, 0     ;Set to 0 on row
4099 000010D4 E98DFFFFFF                   jmp .wttycursorupdatego
4100                                   .wttybell:
4101 000010D9 48B9E803000000000000-        mov rcx, 1000    ;Beep for a second
4101 000010E2 00
4102 000010E3 BBA9040000                   mov ebx, 04A9h  ;Frequency divisor for 1000Hz tone
4103 000010E8 E8A9EFFFFF                   call beep
4104 000010ED E978FFFFFF                   jmp .wttyexit
4105
4106                                   .get_mode:
4107                                   ;Takes no arguments
4108                                   ;Returns ah=Number of Columns, al=Current Screen mode, bh=active
                                                                        page
4109 000010F2 8A2425[53010000]             mov ah, byte [scr_cols]
4110 000010F9 8A0425[58010000]             mov al, byte [scr_mode]
4111 00001100 8A3C25[59010000]             mov bh, byte [scr_active_page]
4112 00001107 E989FBFFFF                   jmp .exit
4113
4114
4115                                   ;Bad string argument for below function
4116                                   .wsbad:
4117 0000110C 48B8FFFF000000000000-        mov rax, 0FFFFh
```

```
4117 00001115 00
4118 00001116 E97AFBFFFF                jmp .exit
4119                                 .write_string:
4120                                 ;bh=page to print on, bl=attribute, cx=number of chars to print
4121                                 ;dh=y coord to print at, dl=x coord to print at, rbp=string
4122                                 ;al contains subfunction
4123                                 ;al=0 attrib in bl, cursor NOT updated
4124                                 ;al=1 attrib in bl, cursor updated
4125                                 ;al=2 string alt attrib/char, cursor NOT updated
4126                                 ;al=3 string alt attrib/char, cursor updated
4127                                 ;al=4 print 0 terminated string
4128 0000111B 3C04                      cmp al, 4h
4129 0000111D 0F84AA000000              je .wszero     ;If its a zero terminated string, go down
4130 00001123 E3E7                      jrcxz .wsbad
4131 00001125 3C04                      cmp al, 4h     ;Bad argument
4132 00001127 77E3                      ja .wsbad
4133                                 .ws:
4134 00001129 56                        push rsi
4135 0000112A 51                        push rcx
4136 0000112B 52                        push rdx
4137 0000112C 53                        push rbx
4138 0000112D 50                        push rax
4139
4140 0000112E 53                        push rbx
4141 0000112F 88FB                      mov bl, bh
4142 00001131 0FB6DB                    movzx ebx, bl
4143 00001134 66678BB41B-               mov si, word [scr_curs_pos + 2*ebx]     ;Fast get cursor position
4143 00001139 [43010000]
4144 0000113D 5B                        pop rbx
4145 0000113E 56                        push rsi     ;Save the current cursor position
4146
4147 0000113F 50                        push rax
4148 00001140 B402                      mov ah, 02h     ;Set cursor at dx
4149 00001142 CD30                      int 30h
4150 00001144 58                        pop rax
4151
4152                                 .ws0:
4153 00001145 51                        push rcx
4154 00001146 53                        push rbx
4155 00001147 50                        push rax
4156 00001148 88C4                      mov ah, al
4157 0000114A 8A4500                    mov al, byte [rbp]  ;Get char
4158 0000114D 48FFC5                    inc rbp
4159 00001150 3C07                      cmp al, 07h
4160 00001152 7462                      je .wsctrlchar
4161 00001154 3C08                      cmp al, 08h
4162 00001156 745E                      je .wsctrlchar
4163 00001158 3C0A                      cmp al, 0Ah
4164 0000115A 745A                      je .wsctrlchar
4165 0000115C 3C0D                      cmp al, 0Dh
4166 0000115E 7456                      je .wsctrlchar
4167
4168 00001160 80FC02                    cmp ah, 2     ;Check if we need to get the char attrib too
4169 00001163 7206                      jb .ws1
4170 00001165 8A5D00                    mov bl, byte [rbp]     ;Get char attrib
4171 00001168 48FFC5                    inc rbp
4172                                 .ws1:
4173 0000116B 66B90100                  mov cx, 1
4174 0000116F B409                      mov ah, 09h     ;Print char and attrib (either given or taken)
4175 00001171 CD30                      int 30h
4176
4177 00001173 FEC2                      inc dl
4178 00001175 3A1425[53010000]          cmp dl, byte [scr_cols]     ;Check if we passed the end of the
                                                                    row
4179 0000117C 7515                      jne .ws2     ;We havent, skip the reset
4180 0000117E 30D2                      xor dl, dl     ;Reset horizontal pos
4181 00001180 FEC6                      inc dh          ;Goto next row
4182 00001182 3A3425[53010000]          cmp dh, byte [scr_cols]     ;Have we passed the last row?
4183 00001189 7508                      jne .ws2     ;No, put cursor
4184 0000118B 66B80A0E                  mov ax, 0E0Ah     ;Yes, do TTY Line feed
4185 0000118F CD30                      int 30h
4186 00001191 FECE                      dec dh          ;Mov cursor to start of last row on page
4187                                 .ws2:
4188 00001193 B402                      mov ah, 02
4189 00001195 CD30                      int 30h     ;Put cursor at new location
4190                                 .ws3:
4191 00001197 58                        pop rax
4192 00001198 5B                        pop rbx
4193 00001199 59                        pop rcx
4194
4195 0000119A 66FFC9                    dec cx
4196 0000119D 75A6                      jnz .ws0
4197
```

```
                                 .wsexitupdate:      ;Exit returning char to original position
0000119F 5A                          pop rdx
000011A0 3C01                        cmp al, 01h
000011A2 7408                        je .wsexit
000011A4 3C03                        cmp al, 03h
000011A6 7404                        je .wsexit
                                 ;Exit returning char to original position
000011A8 B402                        mov ah, 02h
000011AA CD30                        int 30h
                                 .wsexit:
000011AC 58                          pop rax
000011AD 5B                          pop rbx
000011AE 5A                          pop rdx
000011AF 59                          pop rcx
000011B0 5E                          pop rsi
000011B1 E9DFFAFFFF                  jmp .exit
                                 .wsctrlchar:
                                 ;Handles Control Characters: ASCII Bell, Bspace, LF and CR
000011B6 B40E                        mov ah, 0Eh
000011B8 CD30                        int 30h     ;Print control char as TTY
000011BA 88FB                        mov bl, bh
000011BC 0FB6DB                      movzx ebx, bl
000011BF 66678B941B-                 mov dx, word [scr_curs_pos + 2*ebx]     ;Fast get cursor position
000011C4 [43010000]
000011C8 E9CAFFFFFF                  jmp .ws3
                                 .wszero:
                                 ;Print zero terminated string at cursor on current active page
                                 ;Called with ax=1304, rbp=pointer to string
000011CD 55                          push rbp
000011CE 50                          push rax
                                 .wsz1:
000011CF 8A4500                      mov al, byte [rbp]
000011D2 84C0                        test al, al     ;Check al got a zero char
000011D4 7409                        jz .wsz2
000011D6 48FFC5                      inc rbp
000011D9 B40E                        mov ah, 0Eh
000011DB CD30                        int 30h
000011DD EBF0                        jmp short .wsz1
                                 .wsz2:
000011DF 58                          pop rax
000011E0 5D                          pop rbp
000011E1 E9AFFAFFFF                  jmp .exit

                                 ;Graphics mode specific versions!
                                 .gread:
000011E6 48B8FFFF0000000000-         mov rax, 0FFFFh
000011EF 00
000011F0 E9A0FAFFFF                  jmp .exit      ;Currently unsupported function
                                 .gwrite:
000011F5 48B8FFFF0000000000-         mov rax, 0FFFFh
000011FE 00
000011FF E991FAFFFF                  jmp .exit      ;Currently unsupported function
                                 .gscrollup:
00001204 48B8FFFF0000000000-         mov rax, 0FFFFh
0000120D 00
0000120E E982FAFFFF                  jmp .exit      ;Currently unsupported function
                                 .gscrolldown:
00001213 48B8FFFF0000000000-         mov rax, 0FFFFh
0000121C 00
0000121D E973FAFFFF                  jmp .exit      ;Currently unsupported function

                                 .write_crtc_word: ;Writes cx to the CRTC register in al and al+1
00001222 52                          push rdx

00001223 668B1425[5A010000]          mov dx, word [scr_crtc_base]
0000122B EE                          out dx, al
0000122C FEC2                        inc dl
0000122E 88C4                        mov ah, al     ;Temp save al
00001230 88E8                        mov al, ch     ;Set high bits first
00001232 EE                          out dx, al

00001233 FECA                        dec dl
00001235 88E0                        mov al, ah     ;Bring back al into al
00001237 FEC0                        inc al ;GOTO next CTRC address

00001239 EE                          out dx, al
0000123A FEC2                        inc dl
0000123C 88C8                        mov al, cl
0000123E EE                          out dx, al

0000123F 5A                          pop rdx
00001240 C3                          ret

```

```
                                    .get_page_base:
                                    ;Returns in rsi, the base address of the selected page
                                    ;Called with BH = page number, BL=screen mode
                                    ;return RSI=Base of selected page, since rsi is already clobbered
00001241 51                             push rcx
00001242 53                             push rbx

00001243 88F9                           mov cl, bh     ;mov into cl, free bx
00001245 480FB6C9                       movzx rcx, cl
                                    ;------Modify this proc with data tables when finalised!!----
00001249 80FB02                         cmp bl, 2
0000124C 66BB0010                       mov bx, 1000h     ;Doesnt affect flags
00001250 48BE00080000000000-            mov rsi, 800h     ;si is a free register
00001259 00
0000125A 660F42DE                       cmovb bx, si     ;if below, replace with 800h
0000125E 480FB7DB                       movzx rbx, bx         ;zero extend
00001262 8B3425[5C010000]               mov esi, dword [scr_page_addr]
00001269 E308                           jrcxz .gpb1          ;Dont enter the loop if cx is zero
                                    .gpb0:
0000126B 4801DE                         add rsi, rbx     ;add pagesize cx times
0000126E 48FFC9                         dec rcx
00001271 75F8                           jnz .gpb0          ;go around


                                    .gpb1:
00001273 5B                             pop rbx
00001274 59                             pop rcx
00001275 C3                             ret

                                    .page_cursor_offset:
                                    ;Returns in rax the offset into the RAM page of the cursor
                                    ;Works for A/N modes and graphic, though must be shl by 1 for A/N modes
                                    ;bh contains page to work out address
00001276 53                             push rbx
00001277 88FB                           mov bl, bh     ;bring the page number from bh into bl
00001279 480FB6DB                       movzx rbx, bl
0000127D 668B841B[43010000]             mov ax, word [scr_curs_pos + 2*rbx]      ;move cursor position into ax
00001285 5B                             pop rbx
                                    .offset_from_ax:
                                    ;Same as above but now ax needs to be packed as in the cursor
00001286 52                             push rdx
00001287 53                             push rbx
00001288 4831DB                         xor rbx, rbx
0000128B 00C3                           add bl, al     ;move columns into bl
0000128D 66C1E808                       shr ax, 8     ;mov rows from ah to al to use 8 bit mul

00001291 F62425[53010000]               mul byte [scr_cols]     ;multiply the row we are on by columns, store in ax
00001298 6601D8                         add ax, bx          ;add number of columns to this mix!
0000129B 480FB7C0                       movzx rax, ax

0000129F 5B                             pop rbx
000012A0 5A                             pop rdx
000012A1 C3                             ret
                                    .text_scroll_c1:
                                    ;Common function
                                    ;Scrolls a single pair of lines from column given in cl to dl
                                    ;rsi/rdi assumed to be pointing at the right place
                                    ;Direction to be set by calling function
                                    ;All registers EXCEPT pointers preserved, rbp returns # of words
000012A2 51                             push rcx
000012A3 52                             push rdx
000012A4 4831ED                         xor rbp, rbp
000012A7 88CE                           mov dh, cl     ;Save upper left corner in dh, freeing cx
000012A9 88D1                           mov cl, dl
000012AB 28F1                           sub cl, dh     ;Get correct number of words to copy into cl
000012AD 480FB6C9                       movzx rcx, cl
000012B1 48FFC1                         inc rcx     ;absolute value, not offset
000012B4 4889CD                         mov rbp, rcx     ;Save number of words in rbp
000012B7 F366A5                         rep movsw     ;Move char/attrib for one row
000012BA 5A                             pop rdx
000012BB 59                             pop rcx
000012BC C3                             ret
                                    .cursor_proc:
                                    ;Called with bh containing page number
                                    ;Sets cursor on page in bh
                                    ;Returns nothing
000012BD E8B4FFFFFF                     call .page_cursor_offset     ;rax rets offset, no shift needed

000012C2 88F9                           mov cl, bh
000012C4 480FB6C9                       movzx rcx, cl
                                    ;------Modify this proc with data tables when finalised!!----
```

```
000012C8 6631F6                     xor si, si
000012CB 66BA0008                   mov dx, 800h ;Most legacy Pages are sized 800h PELs, VGA greater
000012CF 803C25[58010000]02         cmp byte [scr_mode], 2
000012D7 7303                       jae .cp1
000012D9 66D1EA                     shr dx, 1     ;If in modes 0,1, 400h PELs per page

                                .cp1:
000012DC 84C9                       test cl, cl
000012DE 7407                       jz .cpwrite
000012E0 6601D6                     add si, dx
000012E3 FEC9                       dec cl
000012E5 75F5                       jnz .cp1


                                .cpwrite:
000012E7 6689C1                     mov cx, ax     ;move ax into cx
000012EA 6601F1                     add cx, si
000012ED B00E                       mov al, 0Eh    ;Cursor row
000012EF E82EFFFFFF                 call .write_crtc_word    ;cx has data to output, al is crtc reg

000012F4 C3                         ret
                                ;------------------------End of Interrupt------------------------
                                ;------------------------Basic Config Int 31h------------------------
                                ;This interrupt returns in ax the Hardware Bitfield from the
                                ; data area and the mass storage device details.
                                ;----------------------------------------------------------------------
                                machineWord_io:
000012F5 668B0425[C9010000]         mov ax, word [MachineWord]     ;Return the legacy bitfield

000012FD 4C0FB60425-                movzx r8, byte [i33Devices] ;Get Number of i33h devices
00001302 [A8010000]
00001306 49C1E008                   shl r8, 8    ;Shift up by a byte
0000130A 448A0425[4B020000]         mov r8b, byte [numMSD]  ;Get the number of Mass Storage Devices
                                                              (on EHCI)
00001312 49C1E008                   shl r8, 8    ;Shift up by a byte again
00001316 448A0425[AA010000]         mov r8b, byte [fdiskNum]     ;Get the number of fixed disks
0000131E 49C1E008                   shl r8, 8  ;Shift up by a byte again
00001322 448A0425[66000000]         mov r8b, byte [numCOM]       ;Get the number of COM ports

0000132A 48CF                       iretq
                                ;------------------------End of Interrupt------------------------
                                ;------------------------Basic RAM Int 32h------------------------
                                ;This interrupt returns in ax amount of conventional memory in ax
                                ;----------------------------------------------------------------------
                                convRAM_io:
0000132C 668B0425[CB010000]         mov ax, word [convRAM]     ;Return the amount of conventional RAM
00001334 4C8B0425[CD010000]         mov r8, qword [userBase]     ;Return the userbase to a caller
0000133C 4C8B0C25[F0050000]         mov r9, qword [bigmapptr]    ;Return the big Map pointer
00001344 4C0FB61425-                movzx r10, byte [bigmapSize]     ;Return the number of 24 byte
                                                              entries
00001349 [D5010000]
0000134D 48CF                       iretq
                                ;------------------------End of Interrupt------------------------
                                ;----------------------Storage Interrupt Int 33h----------------------
                                ;Input : dl = Drive number, rbx = Address of buffer,
                                ;        al = number of sectors, ch = Track number,
                                ;        cl = Sector number, dh = Head number
                                ;Input LBA: dl = Drive Number, rbx = Address of Buffer,
                                ;           al = number of sectors, rcx = LBA number
                                ;
                                ;All registers not mentioned above, preserved
                                ;----------------------------------------------------------------------
                                disk_io:
0000134F F6C280                     test dl, 80h
00001352 0F858A000000               jnz .baddev    ;If bit 7 set, exit (temp for v0.9)
00001358 52                         push rdx
00001359 FEC2                       inc dl          ;Inc device number count to absolute value
0000135B 3A1425[A8010000]           cmp dl, byte [i33Devices]
00001362 5A                         pop rdx
00001363 777D                       ja .baddev

00001365 E8D3030000                 call .busScan    ;Bus scan only in valid cases
0000136A 803C25[A9010000]40         cmp byte [msdStatus], 40h    ;Media seek failed
00001372 747E                       je .noDevInDrive

00001374 84E4                       test ah, ah
00001376 0F8484000000               jz .reset              ;ah = 00h Reset Device
0000137C FECC                       dec ah
0000137E 0F84AF000000               jz .statusreport      ;ah = 01h Get status of last op and req.
                                                              sense if ok

00001384 C60425[A9010000]00         mov byte [msdStatus], 00     ;Reset status byte for following
                                                              operations

0000138C FECC                       dec ah
```

```
0000138E 0F841E010000            jz .readsectors      ;ah = 02h CHS Read Sectors
00001394 FECC                    dec ah
00001396 0F843E010000            jz .writesectors     ;ah = 03h CHS Write Sectors
0000139C FECC                    dec ah
0000139E 0F8457010000            jz .verify           ;ah = 04h CHS Verify Sectors
000013A4 FECC                    dec ah
000013A6 0F8470010000            jz .format           ;ah = 05h CHS Format Track (Select Head and
                                                          Cylinder)

000013AC 80FC02                  cmp ah, 02h
000013AF 0F84A3020000            je .formatLowLevel   ;ah = 07h (SCSI) Low Level Format Device

000013B5 80FC7D                  cmp ah, 7Dh          ;ah = 82h LBA Read Sectors
000013B8 0F84E6010000            je .lbaread
000013BE 80FC7E                  cmp ah, 7Eh          ;ah = 83h LBA Write Sectors
000013C1 0F8402020000            je .lbawrite
000013C7 80FC7F                  cmp ah, 7Fh          ;ah = 84h LBA Verify Sectors
000013CA 0F841E020000            je .lbaverify
000013D0 80FC80                  cmp ah, 80h          ;ah = 85h LBA Format Sectors
000013D3 0F843A020000            je .lbaformat
000013D9 80FC83                  cmp ah, 83h          ;ah = 88h LBA Read Drive Parameters
000013DC 0F8498020000            je .lbareadparams
                             .baddev:
000013E2 B401                    mov ah, 01h
000013E4 882425[A9010000]        mov byte [msdStatus], ah    ;Invalid function requested signature
                             .bad:
000013EB 804C241001              or byte [rsp + 2*8h], 1     ;Set Carry flag on for invalid
                                                              function
000013F0 48CF                    iretq
                             .noDevInDrive:
000013F2 8A2425[A9010000]        mov ah, byte [msdStatus]
000013F9 804C241001              or byte [rsp + 2*8h], 1     ;Set Carry flag on for invalid
                                                              function
000013FE 48CF                    iretq
                             .reset: ;Device Reset
00001400 56                      push rsi
00001401 52                      push rdx
00001402 E8E7020000              call .i33ehciGetDevicePtr
00001407 E8A91F0000              call USB.ehciAdjustAsyncSchedCtrlr
0000140C E87D2E0000              call USB.ehciMsdBOTResetRecovery
                             .rrexit:
00001411 5A                      pop rdx
00001412 5E                      pop rsi
00001413 720E                    jc .rrbad
00001415 8A2425[A9010000]        mov ah, byte [msdStatus]
0000141C 80642410FE              and byte [rsp + 2*8h], 0FEh ;Clear CF
00001421 48CF                    iretq
                             .rrbad:
00001423 B405                    mov ah, 5    ;Reset failed
00001425 882425[A9010000]        mov byte [msdStatus], ah
0000142C 804C241001              or byte [rsp + 2*8h], 1     ;Set Carry flag on for invalid
                                                              function
00001431 48CF                    iretq
                             .statusreport:
                             ;If NOT a host/bus/ctrlr type error, request sense and ret code
00001433 8A2425[A9010000]        mov ah, byte [msdStatus]     ;Get last status into ah
0000143A 84E4                    test ah, ah ;If status is zero, exit
0000143C 7507                    jnz .srmain
0000143E 80642410FE              and byte [rsp + 2*8h], 0FEh      ;Clear CF
00001443 48CF                    iretq
                             .srmain:
00001445 C60425[A9010000]00      mov byte [msdStatus], 00     ;Reset status byte
0000144D 80FC20                  cmp ah, 20h       ;General Controller failure?
00001450 7449                    je .srexit
00001452 80FC80                  cmp ah, 80h       ;Timeout?
00001455 7444                    je .srexit
                             ;Issue a Request sense command
00001457 56                      push rsi
00001458 50                      push rax     ;Save original error code in ah on stack
00001459 E890020000              call .i33ehciGetDevicePtr
0000145E E8521F0000              call USB.ehciAdjustAsyncSchedCtrlr
00001463 7241                    jc .srexitbad1
00001465 E8F0330000              call USB.ehciMsdBOTRequestSense
0000146A E8FF2E0000              call USB.ehciMsdBOTCheckTransaction
0000146F 6685C0                  test ax, ax
00001472 58                      pop rax           ;Get back original error code
00001473 752D                    jnz .srexitbad2
00001475 4C0FB60425-             movzx r8, byte [ehciDataIn + 13]   ;Get ASCQ into r8
0000147A [CD030000]
0000147E 49C1E008                shl r8, 8                           ;Make space in lower byte of
                                                              r8 for ASC key
00001482 448A0425[CC030000]      mov r8b, byte [ehciDataIn + 12]    ;Get ASC into r8
0000148A 49C1E008                shl r8, 8                        ;Make space in lower byte of r8
```

```
                                                                   for sense key
0000148E 448A0425[C2030000]              mov r8b, byte [ehciDataIn + 2]  ;Get sense key into al
00001496 4180C8F0                        or r8b, 0F0h                    ;Set sense signature (set upper
                                                                   nybble F)
0000149A 5E                              pop rsi
                                     .srexit:
0000149B 804C241001                      or byte [rsp + 2*8h], 1 ;Non-zero error, requires CF=CY
000014A0 48CF                            iretq
                                     .srexitbad2:
000014A2 B4FF                            mov ah, -1   ;Sense operation failed
000014A4 EB02                            jmp short .srexitbad
                                     .srexitbad1:
000014A6 B420                            mov ah, 20h ;General Controller Failure
                                     .srexitbad:
000014A8 5E                              pop rsi
000014A9 882425[A9010000]                mov byte [msdStatus], ah
000014B0 EB21                            jmp short .rsbad


                                     .readsectors:
000014B2 57                              push rdi
000014B3 48BF-                           mov rdi, USB.ehciMsdBOTInSector512
000014B5 [7E49000000000000]
000014BD E8E0010000                      call .sectorsEHCI
000014C2 5F                              pop rdi
000014C3 8A2425[A9010000]                mov ah, byte [msdStatus]      ;Return Error code in ah
000014CA 7207                            jc .rsbad
000014CC 80642410FE                      and byte [rsp + 2*8h], 0FEh ;Clear CF
000014D1 48CF                            iretq
                                     .rsbad:
000014D3 804C241001                      or byte [rsp + 2*8h], 1      ;Set Carry flag on for invalid
                                                                   function
000014D8 48CF                            iretq


                                     .writesectors:
000014DA 57                              push rdi
000014DB 48BF-                           mov rdi, USB.ehciMsdBOTOutSector512
000014DD [3349000000000000]
000014E5 E8B8010000                      call .sectorsEHCI
000014EA 5F                              pop rdi
000014EB 8A2425[A9010000]                mov ah, byte [msdStatus]
000014F2 72DF                            jc .rsbad
000014F4 80642410FE                      and byte [rsp + 2*8h], 0FEh ;Clear CF
000014F9 48CF                            iretq


                                     .verify:
000014FB 57                              push rdi
000014FC 48BF-                           mov rdi, USB.ehciMsdBOTVerify
000014FE [0B48000000000000]
00001506 E897010000                      call .sectorsEHCI    ;Verify sector by sector
0000150B 5F                              pop rdi
0000150C 8A2425[A9010000]                mov ah, byte [msdStatus]
00001513 72BE                            jc .rsbad
00001515 80642410FE                      and byte [rsp + 2*8h], 0FEh ;Clear CF
0000151A 48CF                            iretq
                                     .format:
                                     ;Cleans sectors on chosen track. DOES NOT Low Level Format.
                                     ;Fills sectors with fill byte from table
0000151C 50                              push rax
0000151D 53                              push rbx
0000151E 51                              push rcx
0000151F 56                              push rsi
00001520 57                              push rdi
00001521 55                              push rbp
00001522 FC                              cld

00001523 51                              push rcx                      ;Save ch = Cylinder number
00001524 488B3425[AF010000]              mov rsi, qword [diskDptPtr]
0000152C B880000000                      mov eax, 80h                    ;128 bytes
00001531 8A4E03                          mov cl, byte [rsi + 3]   ;Bytes per track
00001534 D3E0                            shl eax, cl                     ;Multiply 128 bytes per sector by
                                                                   multiplier
00001536 89C1                            mov ecx, eax
00001538 8A4608                          mov al, byte [rsi + 8]   ;Fill byte for format
0000153B 48BF-                           mov rdi, sectorbuffer         ;Large enough buffer
0000153D [C003000000000000]
00001545 F3AA                            rep stosb                     ;Create mock sector

00001547 8A4E04                          mov cl, byte [rsi + 4]   ;Get sectors per track
0000154A 0FB6E9                          movzx ebp, cl                 ;Put number of sectors in Cylinder
                                                                   in ebp

0000154D 59                              pop rcx                       ;Get back Cylinder number in ch
0000154E B101                            mov cl, 1                     ;Ensure start at sector 1 of
```

```
                                                              Cylinder

00001550 E8C7010000                    call .convertCHSLBA ;Converts to valid 32 bit LBA in ecx for
                                                              geometry type
                                       ;ecx now has LBA
                                  .formatcommon:
00001555 E894010000                    call .i33ehciGetDevicePtr
0000155A 7245                          jc .fbad
0000155C 89CA                          mov edx, ecx      ;Load edx for function call
                                  ;Replace this section with a single USB function
0000155E E8521E0000                    call USB.ehciAdjustAsyncSchedCtrlr
00001563 48BB-                         mov rbx, sectorbuffer
00001565 [C003000000000000]

                                  .f0:
0000156D E8C1330000                    call USB.ehciMsdBOTOutSector512
00001572 0F826B010000                  jc .sebadBB
00001578 FFC2                          inc edx ;Inc LBA
0000157A FFCD                          dec ebp ;Dec number of sectors to act on
0000157C 75EF                          jnz .f0
0000157E F8                            clc
                                  .formatexit:
0000157F 5D                            pop rbp
00001580 5F                            pop rdi
00001581 5E                            pop rsi
00001582 59                            pop rcx
00001583 5B                            pop rbx
00001584 58                            pop rax
00001585 8A2425[A9010000]              mov ah, byte [msdStatus]
0000158C 0F8241FFFFFF                  jc .rsbad
00001592 80642410FE                    and byte [rsp + 2*8h], 0FEh ;Clear CF
00001597 48CF                          iretq
                                  .fbadBB:
00001599 C60425[A9010000]BB            mov byte [msdStatus], 0BBh  ;Unknown Error, request sense
                                  .fbad:
000015A1 F9                            stc
000015A2 EBDB                          jmp short .formatexit
                                  .lbaread:
000015A4 57                            push rdi
000015A5 48BF-                         mov rdi, USB.ehciMsdBOTInSector512
000015A7 [7E49000000000000]

000015AF E88E000000                    call .lbaCommon
000015B4 5F                            pop rdi
000015B5 8A2425[A9010000]              mov ah, byte [msdStatus]      ;Return Error code in ah
000015BC 0F8211FFFFFF                  jc .rsbad
000015C2 80642410FE                    and byte [rsp + 2*8h], 0FEh ;Clear CF
000015C7 48CF                          iretq
                                  .lbawrite:
000015C9 57                            push rdi
000015CA 48BF-                         mov rdi, USB.ehciMsdBOTOutSector512
000015CC [3349000000000000]

000015D4 E869000000                    call .lbaCommon
000015D9 5F                            pop rdi
000015DA 8A2425[A9010000]              mov ah, byte [msdStatus]      ;Return Error code in ah
000015E1 0F82ECFEFFFF                  jc .rsbad
000015E7 80642410FE                    and byte [rsp + 2*8h], 0FEh ;Clear CF
000015EC 48CF                          iretq
                                  .lbaverify:
000015EE 57                            push rdi
000015EF 48BF-                         mov rdi, USB.ehciMsdBOTVerify
000015F1 [0B48000000000000]

000015F9 E844000000                    call .lbaCommon
000015FE 5F                            pop rdi
000015FF 8A2425[A9010000]              mov ah, byte [msdStatus]      ;Return Error code in ah
00001606 0F82C7FEFFFF                  jc .rsbad
0000160C 80642410FE                    and byte [rsp + 2*8h], 0FEh ;Clear CF
00001611 48CF                          iretq
                                  .lbaformat:
00001613 50                            push rax
00001614 53                            push rbx
00001615 51                            push rcx
00001616 56                            push rsi
00001617 57                            push rdi
00001618 55                            push rbp
00001619 FC                            cld
0000161A 0FB6E8                        movzx ebp, al ;Save the number of sectors to format in ebp
0000161D 51                            push rcx
0000161E 52                            push rdx
0000161F B900020000                    mov ecx, 200h
00001624 48BF-                         mov rdi, sectorbuffer
00001626 [C003000000000000]

0000162E 488B1425[AF010000]            mov rdx, qword [diskDptPtr]
00001636 8A4208                        mov al, byte [rdx + 8]   ;Fill byte for format
00001639 F3AA                          rep stosb
```

```
0000163B 5A                         pop rdx
0000163C 59                         pop rcx
0000163D E913FFFFFF                 jmp .formatcommon

                                .lbaCommon:
00001642 50                         push rax
00001643 56                         push rsi
00001644 53                         push rbx
00001645 51                         push rcx
00001646 52                         push rdx
00001647 55                         push rbp
00001648 84C0                       test al, al
0000164A 0F848C000000               jz .se2 ;If al=0, skip copying sectors, clears CF
00001650 0FB6E8                     movzx ebp, al
00001653 E95C000000                 jmp .seCommon

                                ;Low level format, ah=07h
                                .formatLowLevel:
00001658 56                         push rsi
00001659 50                         push rax
0000165A E88F000000                 call .i33ehciGetDevicePtr    ;al = bus num, rsi = ehci device structure ptr
0000165F E83D310000                 call USB.ehciMsdBOTFormatUnit
00001664 58                         pop rax
00001665 5E                         pop rsi
00001666 8A2425[A9010000]           mov ah, byte [msdStatus]
0000166D 0F8260FEFFFF               jc .rsbad
00001673 80642410FE                 and byte [rsp + 2*8h], 0FEh ;Clear CF
00001678 48CF                       iretq
                                .lbareadparams:
                                ;Reads drive parameters (for drive dl which is always valid at this point)
                                ;Output: rax = dBlockSize (Dword for LBA block size)
                                ;        rcx = qLastLBANum (Qword address of last LBA)
0000167A 52                         push rdx
0000167B 480FB6C2                   movzx rax, dl    ;Move drive number offset into rax
0000167F 48BA10000000000000-        mov rdx, int33TblEntrySize
00001688 00

00001689 48F7E2                     mul rdx
0000168C 488D90[BB030000]           lea rdx, qword [diskDevices + rax]   ;Move address into rdx
00001693 8B4203                     mov eax, dword [rdx + 3]    ;Get dBlockSize for device
00001696 488B4A07                   mov rcx, qword [rdx + 7]    ;Get qLastLBANum for device
0000169A 5A                         pop rdx
0000169B 80642410FE                 and byte [rsp + 2*8h], 0FEh ;Clear CF
000016A0 48CF                       iretq
                                .sectorsEHCI:
                                ;Input: rdi = Address of USB EHCI MSD BBB function
                                ;Output: CF = CY: Error, exit
                                ;        CF = NC: No Error
000016A2 50                         push rax
000016A3 56                         push rsi
000016A4 53                         push rbx
000016A5 51                         push rcx
000016A6 52                         push rdx
000016A7 55                         push rbp
000016A8 84C0                       test al, al
000016AA 7430                       jz .se2 ;If al=0, skip copying sectors, clears CF
000016AC 0FB6E8                     movzx ebp, al    ;Move the number of sectors into ebp
000016AF E868000000                 call .convertCHSLBA ;Converts to valid 32 bit LBA in ecx for geometry type
                                    ;ecx now has LBA
                                .seCommon:   ;Entered with ebp = Number of Sectors and ecx = Start LBA
000016B4 E835000000                 call .i33ehciGetDevicePtr
000016B9 7230                       jc .sebad
000016BB 4889CA                     mov rdx, rcx     ;Load edx for function call
                                ;Replace this section with a single USB function
000016BE E8F21C0000                 call USB.ehciAdjustAsyncSchedCtrlr
000016C3 30C0                       xor al, al       ;Sector counter

                                .se1:
000016C5 FEC0                       inc al   ;Inc Sector counter
000016C7 50                         push rax
000016C8 FFD7                       call rdi
000016CA 58                         pop rax
000016CB 7216                       jc .sebadBB
000016CD 4881C300020000             add rbx, 200h    ;Goto next sector
000016D4 48FFC2                     inc rdx ;Inc LBA
000016D7 FFCD                       dec ebp ;Dec number of sectors to act on
000016D9 75EA                       jnz .se1
000016DB F8                         clc

                                .se2:
000016DC 5D                         pop rbp
000016DD 5A                         pop rdx
```

```
4737 000016DE 59                        pop rcx
4738 000016DF 5B                        pop rbx
4739 000016E0 5E                        pop rsi
4740 000016E1 58                        pop rax
4741 000016E2 C3                        ret
4742                                .sebadBB:
4743 000016E3 C60425[A9010000]BB        mov byte [msdStatus], 0BBh  ;Unknown Error, request sense
4744                                .sebad:
4745 000016EB F9                        stc
4746 000016EC EBEE                      jmp short .se2
4747
4748                                .i33ehciGetDevicePtr:
4749                                ;Input: dl = Int 33h number whose
4750                                ;Output: rsi = Pointer to ehci msd device parameter block
4751                                ;         al = EHCI bus the device is on
4752 000016EE 53                        push rbx      ;Need to temporarily preserve rbx
4753 000016EF 480FB6C2                  movzx rax, dl    ;Move drive number offset into rax
4754 000016F3 48BA10000000000000-       mov rdx, int33TblEntrySize
4754 000016FC 00
4755 000016FD 48F7E2                    mul rdx
4756 00001700 488D90[BB030000]          lea rdx, qword [diskDevices + rax]   ;Move address into rdx
4757 00001707 803A00                    cmp byte [rdx], 0    ;Check to see if the device type is 0 (ie
                                                          doesnt exist)
4758 0000170A 740E                      jz .i33egdpbad ;If not, exit
4759 0000170C 668B4201                  mov ax, word [rdx + 1]   ;Get address/Bus pair into ax
4760 00001710 E871250000                call USB.ehciGetDevicePtr    ;Get device pointer into rsi
4761 00001715 88E0                      mov al, ah            ;Get the bus into al
4762 00001717 5B                        pop rbx
4763 00001718 F8                        clc
4764 00001719 C3                        ret
4765                                .i33egdpbad:
4766 0000171A F9                        stc
4767 0000171B C3                        ret
4768
4769                                .convertCHSLBA:
4770                                ;Converts a CHS address to LBA
4771                                ;Input: dl = Drive number, if dl < 80h, use diskdpt. If dl > 80h,
                                                          use hdiskdpt
4772                                ;        ch = Track number, cl = Sector number, dh = Head number
4773                                ;Output: ecx = LBA address
4774                                ;------------Reference Equations------------
4775                                ;C = LBA / (HPC x SPT)
4776                                ;H = (LBA / SPT) mod HPC
4777                                ;S = (LBA mod SPT) + 1
4778                                ;+++++++++++++++++++++++++++++++++++++++++++
4779                                ;LBA = (( C x HPC ) + H ) x SPT + S - 1
4780                                ;-------------------------------------------
4781                                ;Use diskdpt.spt for sectors per track value!
4782                                ;1.44Mb geometry => H=2, C=80, S=18
4783 0000171C 50                        push rax
4784 0000171D 56                        push rsi
4785 0000171E 488B3425[AF010000]        mov rsi, qword [diskDptPtr]
4786 00001726 D0E5                      shl ch, 1     ;Multiply by HPC=2
4787 00001728 00F5                      add ch, dh    ;Add head number
4788 0000172A 88E8                      mov al, ch    ;al = ch = (( C x HPC ) + H )
4789 0000172C F66604                    mul byte [rsi + 4]   ;Sectors per track
4790 0000172F 30ED                      xor ch, ch
4791 00001731 6601C8                    add ax, cx   ;Add sector number to ax
4792 00001734 66FFC8                    dec ax
4793 00001737 0FB7C8                    movzx ecx, ax
4794 0000173A 5E                        pop rsi
4795 0000173B 58                        pop rax
4796 0000173C C3                        ret
4797                                .busScan:
4798                                ;Will request the hub bitfield from the RMH the device is plugged
                                                          in to.
4799                                ;Preserves ALL registers.
4800                                ;dl = Device number
4801
4802                                ;If status changed bit set, call appropriate enumeration function.
4803                                ;If enumeration returns empty device, keep current device data
                                                          blocks in memory,
4804                                ; but return Int 33h error 40h = Seek operation Failed.
4805 0000173D 50                        push rax
4806 0000173E 53                        push rbx
4807 0000173F 51                        push rcx
4808 00001740 52                        push rdx
4809 00001741 56                        push rsi
4810 00001742 57                        push rdi
4811 00001743 55                        push rbp
4812 00001744 4150                      push r8
4813 00001746 4151                      push r9
4814 00001748 4152                      push r10
```

```
4815 0000174A 4153                      push r11
4816
4817 0000174C 4C0FB61C25-               movzx r11, byte [msdStatus] ;Preserve the original status
4817 00001751 [A9010000]
4818
4819 00001755 0FB6EA                    movzx ebp, dl              ;Save the device number in ebp
4820 00001758 E891FFFFFF                call .i33ehciGetDevicePtr  ;Get MSD dev data block ptr in rsi
                                                                    and bus in al
4821                                ;Check port on device for status change.
4822 0000175D 807E0200                  cmp byte [rsi + 2], 0   ;Check if root hub
4823 00001761 0F84CC000000              jz .bsRoot
4824                                ;External Hub procedure
4825 00001767 668B4601                  mov ax, word [rsi + 1]  ;Get bus and host hub address
4826 0000176B 86C4                      xchg al, ah             ;Swap endianness
4827 0000176D 4989F1                    mov r9, rsi
4828 00001770 E811250000                call USB.ehciGetDevicePtr   ;Get the hub address in rsi
4829 00001775 88E0                      mov al, ah
4830 00001777 E8391C0000                call USB.ehciAdjustAsyncSchedCtrlr
4831 0000177C C70425[C0030000]00-       mov dword [ehciDataIn], 0
4831 00001784 000000
4832 00001787 48BAA3000000000004-       mov rdx, 00040000000000A3h ;Get Port status
4832 00001790 00
4833 00001791 410FB65903                movzx ebx, byte [r9 + 3]    ;Get the port number from device
                                                                    parameter block
4834 00001796 48C1E320                  shl rbx, 4*8     ;Shift port number to right position
4835 0000179A 4809D3                    or rbx, rdx
4836 0000179D 0FB64E04                  movzx ecx, byte [rsi + 4]  ;bMaxPacketSize0
4837 000017A1 8A06                      mov al, byte [rsi]       ;Get upstream hub address
4838 000017A3 E8031E0000                call USB.ehciGetRequest
4839 000017A8 722C                      jc .bsErrorExit
4840
4841 000017AA 49B8-                     mov r8, USB.ehciEnumerateHubPort    ;Store address for if bit
                                                                    is set
4841 000017AC [113F000000000000]
4842 000017B4 8B1425[C0030000]          mov edx, dword [ehciDataIn]
4843 000017BB 81E201000100              and edx, 10001h
4844 000017C1 F7C200000100              test edx, 10000h
4845 000017C7 752A                      jnz .bsClearPortChangeStatus    ;If top bit set, clear port
                                                                    change bit
4846                                .bsret:
4847 000017C9 F6C201                    test dl, 1h
4848 000017CC 7418                      jz .bsrExit06h  ;Bottom bit not set, exit media changed Error
                                                                    (edx = 00000h)
4849                                .bsexit:    ;The fall through is (edx = 00001h), no change to dev
                                                                    in port
4850 000017CE 44881C25[A9010000]        mov byte [msdStatus], r11b  ;Get back the original status byte
4851                                .bsErrorExit:
4852 000017D6 415B                      pop r11
4853 000017D8 415A                      pop r10
4854 000017DA 4159                      pop r9
4855 000017DC 4158                      pop r8
4856 000017DE 5D                        pop rbp
4857 000017DF 5F                        pop rdi
4858 000017E0 5E                        pop rsi
4859 000017E1 5A                        pop rdx
4860 000017E2 59                        pop rcx
4861 000017E3 5B                        pop rbx
4862 000017E4 58                        pop rax
4863 000017E5 C3                        ret
4864                                .bsrExit06h:    ;If its clear, nothing in port, return media
                                                                    changed error
4865 000017E6 49BB06000000000000-       mov r11, 06h ;Change the msdStatus byte, media changed or
                                                                    removed
4865 000017EF 00
4866 000017F0 F9                        stc
4867 000017F1 EBDB                      jmp short .bsexit
4868                                .bsClearPortChangeStatus:
4869 000017F3 52                        push rdx
4870 000017F4 C70425[C0030000]00-       mov dword [ehciDataIn], 0
4870 000017FC 000000
4871 000017FF 48BA23011000000000-       mov rdx, 0000000000100123h  ;Set Port status
4871 00001808 00
4872 00001809 410FB65903                movzx ebx, byte [r9 + 3]    ;Get the port number from device
                                                                    parameter block
4873 0000180E 48C1E320                  shl rbx, 4*8     ;Shift port number to right position
4874 00001812 4809D3                    or rbx, rdx
4875 00001815 0FB64E04                  movzx ecx, byte [rsi + 4]  ;bMaxPacketSize0
4876 00001819 8A06                      mov al, byte [rsi]       ;Get device address
4877 0000181B E8EC1C0000                call USB.ehciSetNoData
4878 00001820 5A                        pop rdx
4879 00001821 72B3                      jc .bsErrorExit  ;If error exit by destroying the old msdStatus
4880
4881 00001823 F6C201                    test dl, 1h
```

```
00001826 74BE                     jz .bsrExit06h  ;Bottom bit not set, exit media changed error
                                                  (edx = 10000h)
00001828 EB4C                     jmp short .bsCommonEP   ;Else new device in port needs enum
                                                  (edx = 10001h)
                               .bsRtNoDev:
0000182A 67814C984402000000       or dword [eax + 4*ebx + ehciportsc], 2   ;Clear the bit
                               .bsRoot:
                               ;Root hub procedure.
00001833 E87D1B0000               call USB.ehciAdjustAsyncSchedCtrlr  ;Reset the bus if needed
00001838 E8BA320000               call USB.ehciGetOpBase        ;Get opbase into rax
0000183D 0FB65E03                 movzx ebx, byte [rsi + 3]     ;Get MSD port number into dl
00001841 FFCB                     dec ebx                       ;Reduce by one
00001843 678B549844               mov edx, dword [eax + 4*ebx + ehciportsc]   ;Get port status
                                                  into eax
00001848 80E203                   and dl, 3h        ;Only save bottom two bits
0000184B 84D2                     test dl, dl       ;No device in port  (dl=00b)
0000184D 7497                     jz .bsrExit06h    ;Exit media changed error
0000184F FECA                     dec dl            ;Device in port      (dl=01b)
00001851 0F8477FFFFFF             jz .bsexit        ;Exit, no status change
00001857 FECA                     dec dl            ;New device, Device removed from port    (dl=10b)
00001859 74CF                     jz .bsRtNoDev     ;Clear state change bit and exit Seek error
                               ;Fallthrough case, New device, Device inserted in port  (dl=11b)
0000185B 67814C984402000000       or dword [eax + 4*ebx + ehciportsc], 2   ;Clear the state change
                                                  bit
00001864 49B8-                    mov r8,  USB.ehciEnumerateRootPort    ;The enumeration function
                                                  to call
00001866 [CE37000000000000]
0000186E 4989F1                   mov r9, rsi          ;Store the device pointer in r9
00001871 BE00000000               mov esi, 0           ;Store 0 for root hub parameter block
                               .bsCommonEP:
                               ;Invalidate USB MSD and Int 33h table entries for device
                               ;r9 has device pointer block and rsi has host hub pointer (if on
                                                  RMH)
00001876 66418B19                 mov bx, word [r9]              ;bl = Address, bh = Bus
0000187A 88FE                     mov dh, bh                     ;dh = Bus
0000187C 418A5103                 mov dl, byte [r9 + 3]          ;dl = Device Port
00001880 4D0FB65102               movzx r10, byte [r9 + 2]       ;r10b = Host hub address (0 = Root
                                                  hub)
00001885 6689D8                   mov ax, bx                     ;ax needs a copy for
                                                  RemoveDevFromTables
00001888 E884220000               call USB.ehciRemoveDevFromTables     ;Removes device from USB
                                                  tables
0000188D 87EA                     xchg ebp, edx                       ;device number -><- bus/dev
                                                  pair
0000188F E8E5000000               call .i33removeFromTable            ;Removes device from Int
                                                  33h table
00001894 87EA                     xchg ebp, edx                       ;bus/dev pair -><- device
                                                  number
                               ;Devices enumerated, time to reenumerate!
00001896 B903000000               mov ecx, 3
0000189B 85F6                     test esi, esi     ;Is device on root hub?
0000189D 7502                     jnz .bsr0
0000189F FECA                     dec dl   ;Recall that device port must be device port - 1 for
                                                  Root hub enum
                               .bsr0:
000018A1 41FFD0                   call r8
000018A4 7410                     jz .bsr1
000018A6 803C25[A9010000]20       cmp byte [msdStatus], 20h    ;General Controller Failure?
000018AE 7439                     je .bsrFail
000018B0 FFC9                     dec ecx
000018B2 75ED                     jnz .bsr0
000018B4 EB33                     jmp short .bsrFail
                               .bsr1:
000018B6 4C87CE                   xchg r9, rsi      ;MSD parameter blk -><- Hub parameter blk (or 0
                                                  if root)
000018B9 E8EE280000               call USB.ehciMsdInitialise
000018BE 84C0                     test al, al
000018C0 7527                     jnz .bsrFail      ;Exit if the device failed to initialise
                               ;Multiply dl by int33TblEntrySize to get the address to write
                                                  Int33h table
000018C2 89EA                     mov edx, ebp      ;Move the device number into edx (dl)
000018C4 B810000000               mov eax, int33TblEntrySize  ;Zeros the upper bytes
000018C9 F6E2                     mul dl  ;Multiply dl by al. ax has offset into diskDevices table
000018CB 4805[BB030000]           add rax, diskDevices
000018D1 4889C7                   mov rdi, rax      ;Put the offset into the table into rdi
000018D4 E820000000               call .deviceInit
000018D9 84C0                     test al, al
000018DB 0F84EDFEFFFF             jz .bsexit   ;Successful, exit!
000018E1 3C03                     cmp al, 3
000018E3 0F84E5FEFFFF             je .bsexit   ;Invalid device type, but ignore for now
                               .bsrFail:
000018E9 49BB20000000000000-      mov r11, 20h ;Change the msdStatus byte to Gen. Ctrlr Failure
000018F2 00
```

```
000018F3 F9                          stc
000018F4 E9D5FEFFFF                  jmp .bsexit
                                 .deviceInit:
                                 ;Further initialises an MSD device for use with the int33h interface.
                                 ;Adds device data to the allocated int33h data table.
                                 ;Input: rdi = device diskDevice ptr (given by device number*int33TblEntrySize)
                                 ;       rsi = device MSDDevTbl entry (USB address into getDevPtr)
                                 ;Output: al = 0 : Device added successfully
                                 ;        al = 1 : Bus error
                                 ;        al = 2 : Read Capacities/Reset recovary failed after 10 attempts
                                 ;        al = 3 : Invalid device type (Endpoint size too small, temporary)
                                 ;   rax destroyed
                                 ;IF DEVICE HAS MAX ENDPOINT SIZE 64, DO NOT WRITE IT TO INT 33H TABLES
000018F9 51                          push rcx
000018FA B003                        mov al, 3    ;Invalid EP size error code
000018FC 66817E090002                cmp word [rsi + 9], 200h   ;Check IN max EP packet size
00001902 7573                        jne .deviceInitExit
00001904 66817E0C0002                cmp word [rsi + 12], 200h ;Check OUT max EP packet size
0000190A 756B                        jne .deviceInitExit

0000190C 8A4601                      mov al, byte [rsi + 1]   ;Get bus number
0000190F E8A11A0000                  call USB.ehciAdjustAsyncSchedCtrlr
00001914 B001                        mov al, 1        ;Bus error exit
00001916 725F                        jc .deviceInitExit
00001918 B90A000000                  mov ecx, 10
                                 .deviceInitReadCaps:
0000191D E82E2E0000                  call USB.ehciMsdBOTReadCapacity10    ;Preserve al error code
00001922 803C25[A9010000]20          cmp byte [msdStatus], 20h    ;General Controller Failure
0000192A 744B                        je .deviceInitExit
0000192C E83D2A0000                  call USB.ehciMsdBOTCheckTransaction
00001931 6685C0                      test ax, ax      ;Clears CF
00001934 7418                        jz .deviceInitWriteTableEntry    ;Success, write table entry
00001936 E853290000                  call USB.ehciMsdBOTResetRecovery     ;Just force a device reset
0000193B 803C25[A9010000]20          cmp byte [msdStatus], 20h    ;General Controller Failure
00001943 7432                        je .deviceInitExit
00001945 FFC9                        dec ecx
00001947 75D4                        jnz .deviceInitReadCaps
00001949 B002                        mov al, 2    ;Non bus error exit
0000194B F9                          stc ;Set carry, device failed to initialise properly
0000194C EB29                        jmp short .deviceInitExit
                                 .deviceInitWriteTableEntry:
0000194E C60701                      mov byte [rdi], 1    ;MSD USB device signature

00001951 668B06                      mov ax, word [rsi]   ;Get address and bus into ax
00001954 66894701                    mov word [rdi + 1], ax  ;Store in Int 33h table

00001958 8B0425[C4030000]            mov eax, dword [ehciDataIn + 4] ;Get LBA block size
0000195F 0FC8                        bswap eax
00001961 894703                      mov dword [rdi + 3], eax

00001964 8B0425[C0030000]            mov eax, dword [ehciDataIn] ;Get zx qword LastLBA
0000196B 0FC8                        bswap eax
0000196D 48894707                    mov qword [rdi + 7], rax

00001971 C6470F02                    mov byte [rdi + 15], 2  ;Temporary, only accept devices with 200h EP sizes
00001975 30C0                        xor al, al
                                 .deviceInitExit:
00001977 59                          pop rcx
00001978 C3                          ret
                                 .i33removeFromTable:
                                 ;Uses Int 33h device number to invalidate the device table entry
                                 ;Input: dl = Device number
                                 ;Output: Nothing, device entry invalidated
00001979 50                          push rax
0000197A 52                          push rdx
0000197B B010                        mov al, int33TblEntrySize
0000197D F6E2                        mul dl  ;Multiply tbl entry size by device number, offset in ax
0000197F 480FB7C0                    movzx rax, ax
00001983 C680[BB030000]00            mov byte [diskDevices + rax], 0 ;Invalidate entry
0000198A 5A                          pop rdx
0000198B 58                          pop rax
0000198C C3                          ret

                                 diskdpt:    ;Imaginary floppy disk parameter table with disk geometry.
                                 ;For more information on layout, see Page 3−26 of IBM BIOS ref
                                 ;Assume 2 head geometry due to emulating a floppy drive
```

```
0000198D 00                        .fsb:    db 0     ;First specify byte
0000198E 00                        .ssb:    db 0     ;Second specify byte
0000198F 00                        .tto:    db 0     ;Number of timer ticks to wait before turning off
                                                               drive motors
00001990 02                        .bps:    db 2     ;Number of bytes per sector in multiples of 128
                                                               bytes, editable.
                                                     ; 0 = 128 bytes, 1 = 256 bytes, 2 = 512 bytes etc
                                                     ;Left shift 128 by bps to get the real bytes per
                                                               sector
00001991 09                        .spt:    db 9     ;Sectors per track
00001992 00                        .gpl:    db 0     ;Gap length
00001993 00                        .dtl:    db 0     ;Data length
00001994 00                        .glf:    db 0     ;Gap length for format
00001995 FF                        .fbf:    db 0FFh  ;Fill byte for format
00001996 00                        .hst:    db 0     ;Head settle time in ms
00001997 01                        .mst:    db 1     ;Motor startup time in multiples of 1/8 of a second.

                                   fdiskdpt: ;Fixed drive table, only cyl, nhd and spt are valid.
                                   ;            This schema gives roughly 8.42Gb of storage.
                                   ;            All fields with 0 in the comments are reserved post XT
                                                               class BIOS.
00001998 0004                      .cyl:    dw  1024     ;1024 cylinders
0000199A FF                        .nhd:    db  255      ;255 heads
0000199B 0000                      .rwc:    dw  0        ;Reduced write current cylinder, 0
0000199D FFFF                      .wpc:    dw  -1       ;Write precompensation number (-1=none)
0000199F 00                        .ecc:    db  0        ;Max ECC burst length, 0
000019A0 08                        .ctl:    db  08h      ;Control byte (more than 8 heads)
000019A1 00                        .sto:    db  0        ;Standard timeout, 0
000019A2 00                        .fto:    db  0        ;Formatting timeout, 0
000019A3 00                        .tcd:    db  0        ;Timeout for checking drive, 0
000019A4 FF03                      .clz:    dw  1023     ;Cylinder for landing zone
000019A6 3F                        .spt:    db  63       ;Sectors per track
000019A7 00                        .res:    db  0        ;Reserved byte
                                   ;------------------------End of Interrupt------------------------
                                   ;------------------Serial IO Interrupts Int 34h------------------
                                   serial_baud_table:    ;DLAB devisor values
000019A8 1704                          dw    0417h    ;110 baud,     00
000019AA 0003                          dw    0300h    ;150 baud,     01
000019AC 8001                          dw    0180h    ;300 baud,     02
000019AE C000                          dw    00C0h    ;600 baud,     03
000019B0 6000                          dw    0060h    ;1200 baud,    04
000019B2 3000                          dw    0030h    ;2400 baud,    05
000019B4 1800                          dw    0018h    ;4800 baud,    06
000019B6 0C00                          dw    000Ch    ;9600 baud,    07
000019B8 0600                          dw    0006h    ;19200 baud,   08
000019BA 0300                          dw    0003h    ;38400 baud,   09
000019BC 0200                          dw    0002h    ;57600 baud,   0A
000019BE 0100                          dw    0001h    ;115200 baud,  0B
                                   serial_abt: ;serial port address base table. List of supported
                                                               addresses!
000019C0 F803                          dw com1_base
000019C2 F802                          dw com2_base
000019C4 E803                          dw com3_base
000019C6 E802                          dw com4_base
                                   serial_io:
000019C8 52                            push rdx          ;Save upper 7 bytes
000019C9 6681FA0400                    cmp dx, 4          ;Check to see if the selected com port is
                                                               within range
000019CE 7D5A                          jge .sbadexit1     ;Bad dx value
000019D0 480FB7D2                      movzx rdx, dx      ;zero the upper 6 bytes of rdx
000019D4 668B9412[67000000]            mov dx, word [com_addresses + rdx*2]     ;get serial port base
                                                               addr into dx
000019DC 6685D2                        test dx, dx          ;is the address zero?
000019DF 744D                          jz .sbadexit2      ;com port doesnt exist
000019E1 50                            push rax          ;Saves upper 6 bytes
000019E2 52                            push rdx          ;Save base for exit algorithm

000019E3 84E4                          test ah, ah
000019E5 7451                          jz .userinit
000019E7 FECC                          dec ah
000019E9 0F848A000000                  jz .transmit
000019EF FECC                          dec ah
000019F1 0F84B7000000                  jz .recieve
000019F7 FECC                          dec ah
000019F9 741E                          jz .sioexit      ;since this puts the status into ax
000019FB FECC                          dec ah
000019FD 0F842A010000                  jz .extinit
00001A03 FECC                          dec ah
00001A05 0F8422010000                  jz .extstatus
00001A0B FECC                          dec ah
00001A0D 0F841A010000                  jz .custombaud

                                   .badin:
```

```
00001A13 5A                          pop rdx
00001A14 58                          pop rax
00001A15 B480                        mov ah, 80h     ;Invalid Function
00001A17 EB17                        jmp short .sbadcommon

                                 .sioexit:
00001A19 5A                          pop rdx    ;Get base back, to know exact offset
00001A1A 58                          pop rax          ;Return the upper bytes of rax into rax
00001A1B 6681C20500                  add dx, 5     ;point to the line status register
00001A20 EC                          in al, dx     ;get status
00001A21 88C4                        mov ah, al     ;save line status in ah
00001A23 66FFC2                      inc dx          ;point to the modem status register
00001A26 EC                          in al, dx     ;save modem status in al
00001A27 5A                          pop rdx
00001A28 48CF                        iretq


                                 .sbadexit1:
00001A2A B0FF                        mov al, 0FFh     ;dx was too large
00001A2C EB02                        jmp short .sbadcommon
                                 .sbadexit2:
00001A2E B0FE                        mov al, 0FEh     ;COM port doesnt exist
                                 .sbadcommon:
00001A30 5A                          pop rdx          ;return original rdx value
00001A31 804C241001                  or byte [rsp + 2*8h], 1     ;Set Carry flag on for invalid function
00001A36 48CF                        iretq


                                 .userinit:
00001A38 88C4                        mov ah, al     ;save the data in ah for the baud rate
00001A3A 6681C20300                  add dx, 3     ;Point to the line control register
00001A3F 241F                        and al, 00011111b   ;Zero out the upper three bits
00001A41 0C80                        or al, 10000000b     ;Set the DLAB bit
00001A43 EE                          out dx, al

00001A44 6681EA0300                  sub dx, 3     ;return point to base
00001A49 66C1E80D                    shr ax, 0Dh  ;0Dh=move hi bits of hi word into low bits of low word
00001A4D 480FB6C0                    movzx rax, al     ;zero upper 7 bytes of rax
00001A51 3C07                        cmp al, 00000111b     ;Check if set to 9600baud (for extension)
00001A53 7414                        je .ui2
                                 .ui1:
00001A55 668B80[A8190000]            mov ax, word [serial_baud_table + rax]      ;rax is the offset into the table
00001A5C 66EF                        out dx, ax     ;dx points to base with dlab on, set divisor! (word out)
                                 ;Disable DLAB bit now
00001A5E 6681C20300                  add dx, 3
00001A63 EC                          in al, dx     ;Get the Line Control Register (preserving the written data)
00001A64 247F                        and al, 01111111b    ;Clear the DLAB bit, preserve the other bits
00001A66 EE                          out dx, al     ;Clear the bit

00001A67 EBB0                        jmp short .sioexit      ;exit!
                                 .ui2:     ;Check r8b to make sure it is 0-4 inclusive.
00001A69 4180F804                    cmp r8b, 4     ;greater than four defaults to 4
00001A6D 7F05                        jg .ui3     ;r8b is greater than four, error!
00001A6F 4400C0                      add al, r8b     ;increase the offset into the table
00001A72 EBE1                        jmp short .ui1     ;return to the get value from table
                                 .ui3:     ;If r8b greater than 4, default to 4
00001A74 41B004                      mov r8b, 4    ;Error caught, user used a value greater than 4, default to 4
00001A77 EBF0                        jmp short .ui2     ;return to checker


                                 .transmit:
00001A79 6681C20500                  add dx, 5     ;dx contains base address, point to Line status register
00001A7E 88C4                        mov ah, al    ;temp save char to send in ah
00001A80 51                          push rcx
00001A81 6631C9                      xor cx, cx
                                 .t1:
00001A84 66FFC9                      dec cx
00001A87 7410                        jz .t2          ;timeout
00001A89 EC                          in al, dx     ;get the LSR byte in
00001A8A 2420                        and al, 00100000b    ;Check the transmit holding register empty bit
00001A8C 74F6                        jz .t1     ;if this is zero, keep looping until it is 1 (aka empty)

00001A8E 59                          pop rcx
00001A8F 88E0                        mov al, ah    ;return data byte down to al
00001A91 6681EA0500                  sub dx, 5     ;reaim to the IO port
00001A96 EE                          out dx, al    ;output the data byte to the serial line!!
00001A97 EB80                        jmp short .sioexit
```

```
                                      .t2:
00001A99 59                               pop rcx
00001A9A 5A                               pop rdx       ;Get base back, to know exact offset
00001A9B 58                               pop rax       ;Return the upper bytes of rax into rax
00001A9C 6681C20500                       add dx, 5     ;point to the line status register
00001AA1 EC                               in al, dx     ;get status
00001AA2 88C4                             mov ah, al    ;save line status in ah
00001AA4 80E480                           and ah, 80h   ;Set error bit (bit 7)
00001AA7 66FFC2                           inc dx        ;point to the modem status register
00001AAA EC                               in al, dx     ;save modem status in al
00001AAB 5A                               pop rdx
00001AAC 48CF                             iretq
                                      .recieve:
                                          ;Gets byte out of appropriate buffer head and places it in al
00001AAE 5A                               pop rdx
00001AAF 58                               pop rax
00001AB0 5A                               pop rdx      ;Undoes the address entry and returns COM port
                                                                   number into dx
00001AB1 52                               push rdx     ;Save it once more
00001AB2 53                               push rbx
00001AB3 480FB7D2                         movzx rdx, dx

00001AB7 FA                               cli      ;Entering a critical area, interrupts off
00001AB8 488B1CD5[AF000000]               mov rbx, qword [comX_buf_head + rdx*8]
00001AC0 483B1CD5[CF000000]               cmp rbx, qword [comX_buf_tail + rdx*8]
00001AC8 7426                             je .r1      ;We are at the head of the buffer, signal error, no
                                                                   char to get.
00001ACA 8A03                             mov al, byte [rbx]      ;store byte into al
00001ACC 88C4                             mov ah, al ;temp save al in ah
00001ACE 48FFC3                           inc rbx      ;move buffer head
00001AD1 483B1CD5[0F010000]               cmp rbx, qword [comX_buf_end + rdx*8]      ;are we at the end of
                                                                   the buffer
00001AD9 7508                             jne .r0      ;no, save new position
00001ADB 488B1CD5[EF000000]               mov rbx, qword [comX_buf_start + rdx*8]   ;yes, wrap around
                                      .r0:
00001AE3 48891CD5[AF000000]               mov qword [comX_buf_head + rdx*8], rbx    ;save new buffer
                                                                   position
00001AEB FB                               sti
00001AEC 5B                               pop rbx
00001AED 5A                               pop rdx
00001AEE EB07                             jmp short .rexit
                                      .r1:
00001AF0 FB                               sti
00001AF1 B480                             mov ah, 80h      ;Equivalent to a timeout error.
00001AF3 5B                               pop rbx
00001AF4 5A                               pop rdx
00001AF5 48CF                             iretq

                                      .rexit:      ;Line status in ah. Char was got so ensure DTR is now
                                                                   high again!
00001AF7 668B9412[67000000]               mov dx, word [com_addresses + rdx*2]      ;Get the base address
                                                                   back into dx
00001AFF 6681C20400                       add dx, 4      ;point to the modem control register
00001B04 EC                               in al, dx
00001B05 A801                             test al, 1     ;Test DTR is clear
00001B07 740B                             jz .getscratch
                                      .gsret:
00001B09 0C01                             or al, 1      ;Set DTR bit on again
00001B0B EE                               out dx, al
00001B0C 66FFC2                           inc dx        ;point to the line status register
00001B0F EC                               in al, dx     ;get status
00001B10 86E0                             xchg ah, al ;swap them around
00001B12 48CF                             iretq
                                      .getscratch:
00001B14 0C10                             or al, 00010000b      ;Enable loopback mode with DTR on
00001B16 EE                               out dx, al
00001B17 6681C20300                       add dx, 3      ;Point to scratch register
00001B1C EC                               in al, dx      ;Get overrun char
00001B1D 6681EA0700                       sub dx, 7      ;transmit register
00001B22 EE                               out dx, al     ;send the char (no need to play with DTR, we
                                                                   sending to
                                                         ; ourselves, generating an INT)
00001B23 6681C20400                       add dx, 4      ;point back to modem control register again!
00001B28 EC                               in al, dx
00001B29 24EF                             and al, 11101111b      ;Clear loopback mode, DTR bit gets set in
                                                                   main proc
00001B2B EBDC                             jmp short .gsret


                                      .extinit:
                                      .extstatus:
                                      .custombaud:
00001B2D 5A                               pop rdx
00001B2E 58                               pop rax
```

```
00001B2F B486                      mov ah, 86h
00001B31 E9FAFEFFFF                jmp .sbadcommon
                               ;——————————————————End of Interrupt——————————————————
                               ;———————————————Misc IO Interrupts Int 35h———————————————
                               ;Misc features int that can be used for a variety of things.
                               ;This will break compatibility with BIOS, since hopefully more
                               ; advanced features will be present.
                               ;
                               ; ah = 0 − 82h System Reserved
                               ; ah = 83h −> Reserved, Event wait
                               ; ah = 86h −> Delay rcx = # of milliseconds to wait
                               ; ah = 88h −> Basic High Mem Map 1 (First 16MB only)
                               ; ah = 89h to C4h − System Reserved
                               ; ++++++++++++++++++++++++++++++++++++++++++++++++++++
                               ; ah = C5h − FFh BIOS device class dispatcher extensions
                               ; ++++++++++++++++++++++++++++++++++++++++++++++++++++
                               ; ah = C5h −> Misc sys function dispatcher       (3 funct)
                               ; ah = E8h −> Adv mem management sys dispatcher (4 funct)
                               ; ah = F0h −> Sys data table dispatcher          (15 funct)
                               ; ah = F1h −> EHCI system dispatcher             (4 funct)
                               ;————————————————————————————————————————————————————————
                               misc_io:
00001B36 80FC86                    cmp ah, 86h
00001B39 722F                      jb .badFunction
00001B3B 7436                      jz .delay
00001B3D 80FC88                    cmp ah, 88h
00001B40 0F84B0000000              jz .memory16MB

00001B46 80FCC5                    cmp ah, 0C5h    ;Miscellaneous function dispatcher
00001B49 0F84B1000000              jz .miscDispatcher
00001B4F 80FCE8                    cmp ah, 0E8h    ;Advanced memory management system dispatcher
00001B52 0F847E010000              jz .advSysMemDispatcher
00001B58 80FCF0                    cmp ah, 0F0h    ;System table dispatcher
00001B5B 0F84F3010000              jz .sysDataTableDispatcher
00001B61 80FCF1                    cmp ah, 0F1h    ;EHCI function dispatcher
00001B64 0F843E030000              jz .ehciFunctionDispatcher
                               .badFunction:
00001B6A B480                      mov ah, 80h     ;Invalid Function
                               .badout:
00001B6C 804C241001                or byte [rsp + 2*8h], 1     ;Set Carry flag on for invalid
                                                               function
00001B71 48CF                      iretq

                               .delay:
                               ;Input: rcx = milliseconds to wait (rcx < 7FFFFFFFFFFFFFFFh)
                               ;Init IRQ 8, wait for loop to end, deactivate
00001B73 FA                        cli     ;NO INTERRUPTS
00001B74 4885C9                    test rcx, rcx
00001B77 747B                      jz .return  ;Can avoid sti since we return caller flags
00001B79 50                        push rax
                               ;Ensure PIC is saved
00001B7A E421                      in al, pic1data
00001B7C 50                        push rax    ;Save unaltered pic1 value
00001B7D 24FB                      and al, 0FBh ;Ensure Cascading pic1 line unmasked
00001B7F E621                      out pic1data, al

00001B81 E4A1                      in al, pic2data
00001B83 50                        push rax    ;Save unaltered pic2 value
00001B84 24FE                      and al, 0FEh ;Ensure line 0 of pic2 unmasked
00001B86 E6A1                      out pic2data, al

00001B88 48890C25[3B010000]        mov qword [rtc_ticks], rcx
00001B90 66B88B8B                  mov ax, 8B8Bh
00001B94 E670                      out cmos_base, al    ;NMI disabled
00001B96 E680                      out waitp, al
00001B98 EB00                      jmp short $+2
00001B9A E471                      in al, cmos_data
00001B9C 247F                      and al, 7Fh     ;Clear upper bit
00001B9E 0C40                      or al, 40h     ;Set periodic interrupt bit
00001BA0 86E0                      xchg ah, al
00001BA2 E670                      out cmos_base, al
00001BA4 E680                      out waitp, al
00001BA6 EB00                      jmp short $+2
00001BA8 86C4                      xchg al, ah
00001BAA E671                      out cmos_data, al
00001BAC B00D                      mov al, 0Dh      ;Read Register D and reenable NMI
00001BAE E670                      out cmos_base, al
00001BB0 E680                      out waitp, al      ;allow one io cycle to run
00001BB2 EB00                      jmp short $+2
00001BB4 E471                      in al, cmos_data
00001BB6 FB                        sti          ;Reenable interrupts
                               .loopdelay:
00001BB7 F390                      pause ;allow an interrupt to occur
```

```
5330 00001BB9 48813C25[3B010000]-          cmp qword [rtc_ticks], 0         ;See if we at 0 yet
5330 00001BC1 00000000
5331 00001BC5 7FF0                         jg .loopdelay     ;If not, keep looping
5332                                     ;Return CMOS to default state
5333 00001BC7 FA                           cli
5334 00001BC8 66B88B8B                     mov ax, 8B8Bh    ;NMI disabled
5335 00001BCC E670                         out cmos_base, al
5336 00001BCE E680                         out waitp, al
5337 00001BD0 EB00                         jmp short $+2
5338 00001BD2 E471                         in al, cmos_data
5339 00001BD4 240F                         and al, 0Fh     ;Clear all upper 4 bits
5340 00001BD6 86E0                         xchg ah, al
5341 00001BD8 E670                         out cmos_base, al
5342 00001BDA E680                         out waitp, al
5343 00001BDC EB00                         jmp short $+2
5344 00001BDE 86E0                         xchg ah, al
5345 00001BE0 E671                         out cmos_data, al
5346 00001BE2 B00D                         mov al, 0Dh      ;Read Register D and reenable NMI
5347 00001BE4 E670                         out cmos_base, al
5348 00001BE6 E680                         out waitp, al      ;allow one io cycle to run
5349 00001BE8 EB00                         jmp short $+2
5350 00001BEA E471                         in al, cmos_data
5351
5352 00001BEC 58                           pop rax ;Return pic2 value
5353 00001BED E6A1                         out pic2data, al
5354 00001BEF 58                           pop rax      ;Return pic1 value
5355 00001BF0 E621                         out pic1data, al
5356
5357 00001BF2 58                           pop rax      ;Return rax value
5358 00001BF3 FB                           sti
5359                                     .return:
5360 00001BF4 48CF                         iretq
5361                                     .memory16MB:     ;ah=88 function
5362 00001BF6 668B0425[DE010000]           mov ax, word [srData1]
5363 00001BFE 48CF                         iretq
5364
5365                                     .miscDispatcher:
5366                                     ; ax = C500h -> Beep PC speaker
5367                                     ; ax = C501h -> Connect Debugger
5368                                     ; ax = C502h -> Disconnect Debugger
5369 00001C00 84C0                         test al, al       ;Play a tone using PC speaker
5370 00001C02 0F84C7000000                 jz .mdBeeper
5371 00001C08 3C01                         cmp al, 01h       ;Connect Debugger
5372 00001C0A 7409                         jz .mdConnectDebugger
5373 00001C0C 3C02                         cmp al, 02h       ;Disconnect Debugger
5374 00001C0E 7460                         jz .mdDisconnectDebugger
5375 00001C10 E955FFFFFF                   jmp .badFunction
5376                                     .mdConnectDebugger:
5377 00001C15 50                           push rax
5378 00001C16 53                           push rbx
5379 00001C17 52                           push rdx
5380 00001C18 56                           push rsi
5381 00001C19 BA008F0000                   mov edx, 8F00h
5382 00001C1E BB08000000                   mov ebx, codedescriptor
5383 00001C23 48B8-                        mov rax, MCP_int.singleStepsEP   ;Pointer
5383 00001C25 [7D20000000000000]
5384 00001C2D 48BE01000000000000-          mov rsi, 01 ;Interrupt number, Single Step
5384 00001C36 00
5385 00001C37 E88FE4FFFF                   call idtWriteEntry
5386 00001C3C 48B8-                        mov rax, MCP_int.debugEp   ;Pointer
5386 00001C3E [D220000000000000]
5387 00001C46 48BE03000000000000-          mov rsi, 03 ;Interrupt number, Software Breakpoint
5387 00001C4F 00
5388 00001C50 E876E4FFFF                   call idtWriteEntry
5389 00001C55 48B8-                        mov rax, MCP_int.debugEpHardware   ;Pointer
5389 00001C57 [B220000000000000]
5390 00001C5F 48BE3B000000000000-          mov rsi, 3Bh ;Interrupt number, Invoke debugger through
                                                            hardware CTRL+BREAK
5390 00001C68 00
5391 00001C69 E85DE4FFFF                   call idtWriteEntry
5392 00001C6E EB59                         jmp short .mdDebugExit
5393                                     .mdDisconnectDebugger:
5394 00001C70 50                           push rax
5395 00001C71 53                           push rbx
5396 00001C72 52                           push rdx
5397 00001C73 56                           push rsi
5398 00001C74 BA008F0000                   mov edx, 8F00h
5399 00001C79 BB08000000                   mov ebx, codedescriptor
5400 00001C7E 48B8-                        mov rax, i1   ;Pointer
5400 00001C80 [1A4B000000000000]
5401 00001C88 48BE01000000000000-          mov rsi, 01 ;Interrupt number, Single Step
5401 00001C91 00
5402 00001C92 E834E4FFFF                   call idtWriteEntry
```

```
5403 00001C97 48B8-                       mov rax, i3   ;Pointer
5403 00001C99 [384B000000000000]
5404 00001CA1 48BE03000000000000-         mov rsi, 03 ;Interrupt number, Software Breakpoint
5404 00001CAA 00
5405 00001CAB E81BE4FFFF                  call idtWriteEntry
5406 00001CB0 48B8-                       mov rax, ctrlbreak_io   ;Pointer
5406 00001CB2 [DB30000000000000]
5407 00001CBA 48BE3B000000000000-         mov rsi, 3Bh ;Interrupt number, CTRL+Break
5407 00001CC3 00
5408 00001CC4 E802E4FFFF                  call idtWriteEntry
5409                                  .mdDebugExit:
5410 00001CC9 5E                          pop rsi
5411 00001CCA 5A                          pop rdx
5412 00001CCB 5B                          pop rbx
5413 00001CCC 58                          pop rax
5414 00001CCD 48CF                        iretq
5415
5416                                  .mdBeeper:
5417                                  ;Input:
5418                                  ;    bx = Frequency divisor to use for tone
5419                                  ;    rcx = # of ms to beep for
5420                                  ; All registers including ax preserved
5421 00001CCF E8C2E3FFFF                  call beep
5422 00001CD4 48CF                        iretq
5423
5424                                  .advSysMemDispatcher:
5425                                  ; ax = E800h -> Return userBase pointer
5426                                  ; ax = E801h -> Give RAM count, minus the size of SCPBIOS, in ax,
                                                                        bx, cx, dx.
5427                                  ; ax = E802h -> Total RAM count (without SCP/BIOS)
5428                                  ; ax = E820h -> Full Memory Map, including entry for SCPBIOS
5429 00001CD6 84C0                        test al, al
5430 00001CD8 7411                        jz .retUserBase
5431 00001CDA 3C01                        cmp al, 01h
5432 00001CDC 7417                        je .memory64MB
5433 00001CDE 3C02                        cmp al, 02h
5434 00001CE0 7435                        je .memoryBIOSseg
5435 00001CE2 3C20                        cmp al, 20h
5436 00001CE4 7451                        je .fullMemoryMap
5437 00001CE6 E97FFEFFFF                  jmp .badFunction
5438
5439                                  .retUserBase:
5440 00001CEB 488B0425[CD010000]          mov rax, qword [userBase]
5441 00001CF3 48CF                        iretq
5442                                  .memory64MB:
5443 00001CF5 668B0425[D6010000]          mov ax, word [srData]
5444 00001CFD 668B1C25[D8010000]          mov bx, word [srData + 2]
5445 00001D05 668B0C25[DA010000]          mov cx, word [srData + 4]
5446 00001D0D 668B1425[DC010000]          mov dx, word [srData + 6]
5447 00001D15 48CF                        iretq
5448                                  .memoryBIOSseg:
5449                                  ;This gives information about the SCP/BIOS segment
5450 00001D17 48B800001100000000-         mov rax, BIOSStartAddr   ;Start address of BIOS
5450 00001D20 00
5451 00001D21 31DB                        xor ebx, ebx
5452 00001D23 8B1C25[E8010000]            mov ebx, dword [scpSize]      ;Total sum of segment sizes
5453 00001D2A 488B1425[E0010000]          mov rdx, qword [sysMem]       ;Get total usable memory count
5454 00001D32 4829DA                      sub rdx, rbx      ;Remove SCP/BIOS allocation from the size
5455 00001D35 48CF                        iretq
5456
5457                                  .fullMemoryMap:
5458 00001D37 488B0425[CD010000]          mov rax, qword [userBase]      ;Start space, returns userbase in
                                                                        r8
5459 00001D3F 48BE-                       mov rsi, bigmapptr
5459 00001D41 [F005000000000000]
5460 00001D49 8A0C25[D5010000]            mov cl, byte [bigmapSize]     ;Get the number of 24 byte entries
5461 00001D50 30ED                        xor ch, ch                    ;Reserve the upper byte
5462 00001D52 48CF                        iretq
5463
5464                                  .sysDataTableDispatcher:
5465                                  ; ax = F000h, Register new GDT ptr
5466                                  ; ax = F001h, Register new IDT ptr
5467                                  ; ax = F002h, Get Current GDT ptr
5468                                  ; ax = F003h, Get Current IDT ptr
5469                                  ; ax = F004h, Register New Page Tables
5470                                  ; ax = F005h, Get physical address of PTables
5471                                  ; ax = F006h, Get pointer to BIOS Data Area
5472                                  ; ax = F007h, Read IDT entry
5473                                  ; ax = F008h, Write IDT entry
5474                                  ; ax = F009h, Register new Disk Parameter Table
5475                                  ; ax = F00Ah, Get current DPT
5476                                  ; ax = F00Bh, Register new Fixed Disk Parameter Table
5477                                  ; ax = F00Ch, Get current fDPT
```

```
5478                                ; ax = F00Dh, Register new SysInit parameters
5479                                ; ax = F00Eh, Get current SysInit parameters
5480 00001D54 3C04                      cmp al, 4h
5481 00001D56 725D                      jb .sdtDT             ;al = 00 - 03, goto sdtDT
5482 00001D58 3C04                      cmp al, 4
5483 00001D5A 0F8493000000              jz .sdtRegisterPage   ;al = 04
5484 00001D60 3C05                      cmp al, 5
5485 00001D62 0F8495000000              jz .sdtGetPagePtr     ;al = 05
5486 00001D68 3C06                      cmp al, 6
5487 00001D6A 0F8497000000              jz .sdtDataptr        ;al = 06
5488 00001D70 3C07                      cmp al, 7
5489 00001D72 0F849B000000              jz .sdtReadIDTEntry   ;al = 07
5490 00001D78 3C08                      cmp al, 8
5491 00001D7A 0F84C4000000              jz .sdtWriteIDTEntry      ;al = 08
5492 00001D80 3C09                      cmp al, 9
5493 00001D82 0F84D4000000              jz .sdtNewDDP         ;al = 09
5494 00001D88 3C0A                      cmp al, 0Ah
5495 00001D8A 0F84E0000000              jz .sdtReadDDP        ;al = 0A
5496 00001D90 3C0B                      cmp al, 0Bh
5497 00001D92 0F84CE000000              jz .sdtNewfDDP        ;al = 0Bh
5498 00001D98 3C0C                      cmp al, 0Ch
5499 00001D9A 0F84DA000000              jz .sdtReadfDDP       ;al = 0Ch
5500 00001DA0 3C0D                      cmp al, 0Dh
5501 00001DA2 0F84DC000000              jz .sdtNewSysInit     ;al = 0Dh
5502 00001DA8 3C0E                      cmp al, 0Eh
5503 00001DAA 0F84E6000000              jz .sdtReadSysInit    ;al = 0Eh
5504 00001DB0 E9B5FDFFFF                jmp .badFunction
5505
5506                                .sdtDT:
5507                                ;sys data tables Descriptor Table dispatcher
5508                                ;rbx has/will have I/GDT base pointer (qword)
5509                                ;ecx has/will have I/GDT limit (word)
5510                                ;edx has/will have Number of entries in I/GDT (word)
5511 00001DB5 57                        push rdi
5512 00001DB6 56                        push rsi
5513 00001DB7 48BF-                     mov rdi, GDTlength
5513 00001DB9 [0C00000000000000]
5514 00001DC1 48BE-                     mov rsi, IDTlength
5514 00001DC3 [0000000000000000]
5515 00001DCB A801                      test al, 1  ;If al[0] = 1, want rdi to point to IDT area
5516 00001DCD 480F45FE                  cmovnz rdi, rsi ;If al[0] = 0, rdi will keep pointing to GDT
5517 00001DD1 A802                      test al, 2  ;If bit 2 is set, Get pointers
5518 00001DD3 750F                      jnz .sdtGet
5519 00001DD5 668917                    mov word [rdi], dx
5520 00001DD8 66894F02                  mov word [rdi + 2], cx
5521 00001DDC 48895F04                  mov qword [rdi + 4], rbx
5522 00001DE0 56                        push rsi
5523 00001DE1 5F                        pop rdi
5524 00001DE2 48CF                      iretq
5525                                .sdtGet:
5526 00001DE4 0FB717                    movzx edx, word [rdi]
5527 00001DE7 0FB74F02                  movzx ecx, word [rdi + 2]
5528 00001DEB 488B5F04                  mov rbx, qword [rdi + 4]
5529 00001DEF 56                        push rsi
5530 00001DF0 5F                        pop rdi
5531 00001DF1 48CF                      iretq
5532                                .sdtRegisterPage:
5533 00001DF3 48891C25[18000000]        mov qword [pageTablePtr], rbx   ;Registers pointer as new table
                                                                          space
5534 00001DFB 48CF                      iretq
5535                                .sdtGetPagePtr:
5536 00001DFD 488B1C25[18000000]        mov rbx, qword [pageTablePtr]  ;Return BIOS Page Table ptr
5537 00001E05 48CF                      iretq
5538                                .sdtDataptr:
5539 00001E07 48BB-                     mov rbx, section.data.start         ;Get BIOS Data area ptr into
                                                                          rax
5539 00001E09 [0000000000000000]
5540 00001E11 48CF                      iretq
5541                                .sdtReadIDTEntry:
5542                                ;bx = Number of interrupt handler (00h-0FFFFh), uses only bl
5543                                ;Returns pointer in rbx,
5544                                ;Segment selector in ax,
5545                                ;Attribute word in dx
5546 00001E13 480FB6DB                  movzx rbx, bl
5547 00001E17 488B1425[04000000]        mov rdx, qword [IDTpointer.Base]     ;Get base address
5548 00001E1F 48C1E304                  shl rbx, 4h           ;Multiply address number by 16
5549 00001E23 4801DA                    add rdx, rbx          ;rdx point to IDT entry
5550 00001E26 8B4208                    mov eax, dword [rdx + 8]
5551 00001E29 48C1E020                  shl rax, 20h          ;Shift dword into upper dword
5552 00001E2D 668B5A06                  mov bx, word [rdx + 6]
5553 00001E31 C1E310                    shl ebx, 10h          ;Shift word into upper word
5554 00001E34 668B1A                    mov bx, word [rdx]  ;Get final word
5555 00001E37 4809C3                    or rbx, rax           ;Add upper dword to rbx
```

```
00001E3A 668B4202                       mov ax, word [rdx + 2]   ;Get Segment selector in ax
00001E3E 668B5204                       mov dx, word [rdx + 4]   ;Get attributes word
00001E42 48CF                           iretq
                                    .sdtWriteIDTEntry:
                                    ;rbx = Pointer to new routine
                                    ;cx = Number of the interrupt handler (00h-0FFFFh), uses only cl
                                    ;dx = IDT entry attributes
                                    ;si = Segment selector
00001E44 50                             push rax
00001E45 51                             push rcx
00001E46 56                             push rsi
00001E47 53                             push rbx
00001E48 4889D8                         mov rax, rbx      ;Move pointer to new routine to rax
00001E4B 89F3                           mov ebx, esi      ;Move Segment selector from si to bx
00001E4D 480FB6F1                       movzx rsi, cl     ;Movzx low byte of interrupt number into rsi
00001E51 E875E2FFFF                     call idtWriteEntry
00001E56 5B                             pop rbx
00001E57 5E                             pop rsi
00001E58 59                             pop rcx
00001E59 58                             pop rax
00001E5A 48CF                           iretq
                                    .sdtNewDDP:
00001E5C 48891C25[AF010000]             mov qword [diskDptPtr], rbx
00001E64 48CF                           iretq
                                    .sdtNewfDDP:
00001E66 48891C25[B7010000]             mov qword [fdiskDptPtr], rbx
00001E6E 48CF                           iretq
                                    .sdtReadDDP:
00001E70 488B1C25[AF010000]             mov rbx, qword [diskDptPtr]
00001E78 48CF                           iretq
                                    .sdtReadfDDP:
00001E7A 488B1C25[B7010000]             mov rbx, qword [fdiskDptPtr]
00001E82 48CF                           iretq
                                    .sdtNewSysInit:
00001E84 48891C25[BF010000]             mov qword [nextFilePtr], rbx
00001E8C 66891425[C7010000]             mov word [numSectors], dx
00001E94 48CF                           iretq
                                    .sdtReadSysInit:
00001E96 488B1C25[BF010000]             mov rbx, qword [nextFilePtr]
00001E9E 668B1425[C7010000]             mov dx, word [numSectors]
00001EA6 48CF                           iretq
                                    .ehciFunctionDispatcher:
                                    ;EHCI function dispatcher 0F1h
                                    ; al = 00h -> EHCI get crit error handler
                                    ; al = 01h -> EHCI set crit error handler
                                    ; al = 02h -> Reserved, reset selected EHCI controller
                                    ; al = 03h -> Reserved, re-enumerate devices downstream of EHCI
                                                                Root hub
00001EA8 84C0                           test al, al
00001EAA 7411                           jz .ehciDispGetCritPtr
00001EAC FEC8                           dec al
00001EAE 7417                           jz .ehciDispSetCritPtr
00001EB0 FEC8                           dec al
00001EB2 741D                           jz .ehciDispResetCtrlr
00001EB4 FEC8                           dec al
00001EB6 7419                           jz .echiDispReEnumDevices
00001EB8 E9ADFCFFFF                     jmp .badFunction

                                    .ehciDispGetCritPtr:
                                    ;Gets the address of the current EHCI critical error handler into
                                                                rbx
00001EBD 488B1C25[36020000]             mov rbx, qword [eHCErrorHandler]
00001EC5 48CF                           iretq
                                    .ehciDispSetCritPtr:
                                    ;Sets the address of the EHCI critical error handler to the ptr in
                                                                rbx
00001EC7 48891C25[36020000]             mov qword [eHCErrorHandler], rbx
00001ECF 48CF                           iretq
                                    .ehciDispResetCtrlr:
                                    .echiDispReEnumDevices:
00001ED1 B486                           mov ah, 86h       ;Unsupported function call
00001ED3 E994FCFFFF                     jmp .badout
                                    ;------------------------End of Interrupt------------------------
                                    ;--------------------Keyboard Interrupt Int 36h------------------
                                    ; Software keyboard interrupt.
                                    ; ah = 0 -> Read the next scancode/ASCII struck from the keyboard
                                    ; ah = 1 -> Clear zero flag if there is a new char ready to be
                                    ;             read.
                                    ; ah = 2 -> Returns the current shift status in the al register
                                    ; ax and flags changed.
                                    ;-----------------------------------------------------------------
                                    kb_io:
00001ED8 53                             push rbx
```

```
00001ED9 FA                               cli             ;Interrupts off
00001EDA 84E4                             test ah, ah
00001EDC 7411                             jz .k0
00001EDE FECC                             dec ah
00001EE0 7436                             jz .k1
00001EE2 FECC                             dec ah
00001EE4 7450                             jz .k2
00001EE6 804C241801                       or byte [rsp + 3*8h], 1     ;Set CF, invalid function, skip rbx on stack
00001EEB B480                             mov ah, 80h     ;Invalid Function
00001EED EB4E                             jmp short .kexit ;ah > 2, not a valid function

                                      .k0:
                                      ;This one moves the head to catch up with the tail.
00001EEF FB                               sti
00001EF0 F390                             pause     ;Allow a keyboard interrupt to occur
00001EF2 FA                               cli
00001EF3 488B1C25[42000000]               mov rbx, qword [kb_buf_head]
00001EFB 483B1C25[4A000000]               cmp rbx, qword [kb_buf_tail]     ;Are we at the head of the buffer?
00001F03 74EA                             je .k0     ;If we are, then the buffer is empty, await a keystroke
00001F05 66678B03                         mov ax, word [ebx]          ;move the word pointed at by rbx to ax
00001F09 E833000000                       call .kb_ptr_adv      ;Advance the buffer pointer

00001F0E 48891C25[42000000]               mov qword [kb_buf_head], rbx      ;Move rbx into the buffer head variable
00001F16 EB25                             jmp short .kexit

                                      .k1:
00001F18 488B1C25[42000000]               mov rbx, qword [kb_buf_head]
00001F20 483B1C25[4A000000]               cmp rbx, qword [kb_buf_tail] ;sets flags, Z is set if equal
00001F28 660F4503                         cmovnz ax, word [rbx]      ;move head of buffer into ax, IF Z clear
00001F2C FB                               sti       ;renable interrupts
00001F2D 9C                               pushfq      ;push flags onto stack
00001F2E 5B                               pop rbx     ;pop them into rbx
00001F2F 48895C2418                       mov [rsp + 3*8h], qword rbx      ;Replace with new flags, skip pushed rbx
00001F34 EB07                             jmp short .kexit

                                      .k2:
00001F36 8A0425[62000000]                 mov al, byte [kb_flags]
                                      .kexit:
00001F3D FB                               sti
00001F3E 5B                               pop rbx
00001F3F 48CF                             iretq

                                      .kb_ptr_adv:
                                      ;Advance the pointer passed by rbx safely and return pointer!
00001F41 48FFC3                           inc rbx
00001F44 48FFC3                           inc rbx
00001F47 483B1C25[5A000000]               cmp rbx, qword [kb_buf_end]       ;Are we at the end of the buffer space
00001F4F 7508                             jne .kbpa1                         ;If not exit, if we are, wrap around space!
00001F51 488B1C25[52000000]               mov rbx, qword [kb_buf_start]
                                      .kbpa1:
00001F59 C3                               ret
                                      ;————————————————————End of Interrupt————————————————————
                                      ;————————————————————Printer Int 37h————————————————————
                                      ; Reserved for printer specific functions. Both USB and Parallel.
                                      ; Not currently supported
                                      ;——————————————————————————————————————————————————————————
                                      printer_io:
00001F5A B486                             mov ah, 86h     ;Function not supported
00001F5C 804C241001                       or byte [rsp+ 2*8h], 1     ;Set carry
00001F61 48CF                             iretq
                                      ;————————————————————End of Interrupt————————————————————
                                      ;——————————————————MCP Interrupt Int 38h——————————————————
                                      ;This interrupt superceeds the IBM BASIC routine caller.
                                      ;This is a 64 bit port of my 16 bit MCP monitor program,
                                      ; allowing users to "interactively" get sectors from devices
                                      ; and run them. I might add some nicities to this version of MCP
                                      ; such as a function to list all devices.
                                      ;——————————————————————————————————————————————————————————
                                      MCPjmptbl:   ;Function jump table
00001F63 [9D28000000000000]               dq MCP_int.dumpMemory         ;Dump
00001F6B [9E2A000000000000]               dq MCP_int.editMemory         ;Edit
00001F73 [5A2B000000000000]               dq MCP_int.singleStep         ;Single step
00001F7B [0E2B000000000000]               dq MCP_int.jumpProc           ;Go
00001F83 [692B000000000000]               dq MCP_int.proceedDefault     ;Proceed
00001F8B [7C2B000000000000]               dq MCP_int.storageRead        ;Load
```

```
00001F93 [842B000000000000]         dq MCP__int.storageWrite    ;Write
00001F9B [0E2C000000000000]         dq MCP__int.restartMcp      ;Quit   <- To call Int 40h for DOS
                                                                compatibility
00001FA3 [102C000000000000]         dq MCP__int.clearscreen     ;Clear screen
00001FAB [7823000000000000]         dq MCP__int.xchangeReg      ;Registers
00001FB3 [DB21000000000000]         dq MCP__int.debugRegs       ;Breakpoints
00001FBB [4C25000000000000]         dq MCP__int.hexCalc         ;Hex
00001FC3 [9024000000000000]         dq MCP__int.inport          ;In
00001FCB [E324000000000000]         dq MCP__int.outport         ;Out
00001FD3 [9621000000000000]         dq MCP__int.version         ;Version
00001FDB [5A2B000000000000]         dq MCP__int.singleStep      ;Single Step (Alt), temp
00001FE3 [6720000000000000]         dq MCP__int.memoryMap       ;Print memory map
00001FEB [2F21000000000000]         dq MCP__int.connect         ;Connect Debugger
00001FF3 [6121000000000000]         dq MCP__int.disconnect      ;Disconnect Debugger
                                MCP__int:
                                     ;Entry point from external programs
00001FFB 48890425[04020000]         mov qword [mcpUserRaxStore], rax
00002003 488B0425[EC010000]         mov rax, qword [mcpUserBase]
0000200B 48896008                   mov qword [rax + 08h], rsp
0000200F E8080C0000                 call .storeMainRegisters     ;Save main registers
                                .z11:
00002014 488B2425[0C020000]         mov rsp, qword [mcpStackPtr]   ;Point sp to new stack
0000201C B804130000                 mov eax, 1304h     ;Zero extends to rax
00002021 48BD-                      mov rbp, .prompt
00002023 [EA2D000000000000]
0000202B 30FF                       xor bh, bh
0000202D CD30                       int 30h
                                .z2:
0000202F 6631C0                     xor ax, ax
00002032 CD36                       int 36h
00002034 3C08                       cmp al, 08h           ;If backspace, ignore
00002036 74F7                       je .z2
00002038 E8820D0000                 call .print           ;Print input char
0000203D FD                         std
0000203E 48BF-                      mov rdi, .prompt      ;end of lst is prompt
00002040 [EA2D000000000000]
00002048 48B914000000000000-        mov rcx, .lstl + 1
00002051 00
00002052 F2AE                       repne scasb
00002054 FC                         cld
00002055 0F85AC000000               jne .bad__command      ;Char not found!
                                .prog__sel:      ;Choose program
0000205B 68[14200000]               push MCP__int.z11      ;to allow RETurning to application
00002060 FF24CD[631F0000]           jmp qword [MCPjmptbl + 8*rcx]       ;Jump to chosen function
                                .memoryMap:
00002067 66B80A0E                   mov ax,0E0Ah
0000206B CD30                       int 30h
0000206D 66B80D0E                   mov ax, 0E0Dh
00002071 CD30                       int 30h
00002073 E888DFFFFF                 call e820print   ;Print memory map
00002078 E997FFFFFF                 jmp .z11
                                .singleStepsEP:
0000207D 48890425[04020000]         mov qword [mcpUserRaxStore], rax
00002085 488B0425[EC010000]         mov rax, qword [mcpUserBase]
0000208D 48896008                   mov qword [rax + 08h], rsp
00002091 E8860B0000                 call .storeMainRegisters
00002096 488B0424                   mov rax, qword [rsp]       ;Get next instruction address
0000209A 48890425[F4010000]         mov qword [mcpUserRip], rax
000020A2 E875050000                 call .dumpReg      ;Show register state
000020A7 E802020000                 call .dumpDebugRegs
000020AC FB                         sti ;Restore interrupts
000020AD E962FFFFFF                 jmp .z11
                                .debugEpHardware:
000020B2 48890425[04020000]         mov qword [mcpUserRaxStore], rax
000020BA 488B0425[EC010000]         mov rax, qword [mcpUserBase]
000020C2 48896008                   mov qword [rax + 08h], rsp
000020C6 E8510B0000                 call .storeMainRegisters
000020CB FB                         sti ;Restore interrupts
000020CC B020                       mov al, EOI
000020CE E620                       out pic1command, al
000020D0 EB1A                       jmp short .dep1
                                .debugEp:
                                ;Return here after a single step or int 3.
                                ;Support Int 3h thru manual encoding only, not via the debugger
000020D2 48890425[04020000]         mov qword [mcpUserRaxStore], rax
000020DA 488B0425[EC010000]         mov rax, qword [mcpUserBase]
000020E2 48896008                   mov qword [rax + 08h], rsp
000020E6 E8310B0000                 call .storeMainRegisters
000020EB FB                         sti ;Restore interrupts
                                .dep1:
000020EC 488B0424                   mov rax, qword [rsp]       ;Get next instruction address
000020F0 48890425[F4010000]         mov qword [mcpUserRip], rax
000020F8 E81F050000                 call .dumpReg      ;Show register state
```

```
5790 000020FD E8AC010000                call .dumpDebugRegs
5791 00002102 E90DFFFFFF                jmp .z11
5792                                .bad_command:
5793 00002107 48B804130000000000-       mov rax, 1304h
5793 00002110 00
5794 00002111 30FF                      xor bh, bh
5795 00002113 48BD-                     mov rbp, .bc1
5795 00002115 [2421000000000000]
5796 0000211D CD30                      int 30h
5797 0000211F E9F0FEFFFF                jmp MCP_int.z11
5798 00002124 0A0D205E204572726F-   .bc1: db 0Ah,0Dh," ^ Error",0
5798 0000212D 7200
5799                                ;><><><><><><><><—Internal Commands Begin Here—><><><><><><><><
5800                                .connect:
5801 0000212F 50                        push rax
5802 00002130 55                        push rbp
5803 00002131 B801C50000                mov eax, 0C501h ;Connect Debugger
5804 00002136 CD35                      int 35h
5805 00002138 B804130000                mov eax, 1304h
5806 0000213D 48BD-                     mov rbp, .connectString
5806 0000213F [4C21000000000000]
5807 00002147 CD30                      int 30h
5808 00002149 5D                        pop rbp
5809 0000214A 58                        pop rax
5810 0000214B C3                        ret
5811 0000214C 0A0D53595344454255-   .connectString db 0Ah,0Dh,"SYSDEBUG Connected",0
5811 00002155 4720436F6E6E656374-
5811 0000215E 656400
5812                                .disconnect:
5813 00002161 50                        push rax
5814 00002162 55                        push rbp
5815 00002163 B802C50000                mov eax, 0C502h ;Disconnect Debugger
5816 00002168 CD35                      int 35h
5817 0000216A B804130000                mov eax, 1304h
5818 0000216F 48BD-                     mov rbp, .disconnectString
5818 00002171 [7E21000000000000]
5819 00002179 CD30                      int 30h
5820 0000217B 5D                        pop rbp
5821 0000217C 58                        pop rax
5822 0000217D C3                        ret
5823 0000217E 0A0D53595344454255-   .disconnectString db 0Ah,0Dh,"SYSDEBUG Disconnected",0
5823 00002187 4720446973636F6E6E-
5823 00002190 656374656400
5824                                .version:
5825 00002196 66B80413                  mov ax, 1304h
5826 0000219A 30FF                      xor bh, bh
5827 0000219C 48BD-                     mov rbp, .vstring
5827 0000219E [BE21000000000000]
5828 000021A6 CD30                      int 30h
5829 000021A8 48BE-                     mov rsi, signature + 1      ;Point to BIOS signature string (skip
                                                                      the v char)
5829 000021AA [344F000000000000]
5830                                .v1:
5831 000021B2 AC                        lodsb
5832 000021B3 3C20                      cmp al, 20h                ;Check space
5833 000021B5 7406                      je .v2
5834 000021B7 B40E                      mov ah, 0Eh
5835                                    ;xor bh, bh
5836 000021B9 CD30                      int 30h
5837 000021BB EBF5                      jmp short .v1
5838                                .v2:
5839 000021BD C3                        ret
5840 000021BE 0A0D5343502F42494F-   .vstring:      db 0Ah, 0Dh,"SCP/BIOS SYSDEBUG Version ",0
5840 000021C7 532053595344454255-
5840 000021D0 472056657273696F6E-
5840 000021D9 2000
5841                                .debugRegs:
5842 000021DB E8CE000000                call .dumpDebugRegs
5843 000021E0 66B80413                  mov ax, 1304h
5844 000021E4 48BD-                     mov rbp, .crlf      ;Newline
5844 000021E6 [F02D000000000000]
5845 000021EE CD30                      int 30h
5846
5847 000021F0 66B82E0E                  mov ax, 0E2Eh      ;Print dot byte
5848 000021F4 CD30                      int 30h
5849
5850 000021F6 66B80101                  mov ax, 0101h      ;Process one byte
5851 000021FA E81F0B0000                call .keyb
5852 000021FF 4885ED                    test rbp, rbp
5853 00002202 0F840CFEFFFF              jz .z11     ;If enter pressed, return to command line
5854 00002208 E8B10A0000                call .arg
5855 0000220D 3C01                      cmp al, 1
5856 0000220F 0F85DA070000              jne .dmbadexit
```

```
5857
5858 00002215 488B7D00                    mov rdi, qword [rbp]
5859 00002219 4881FF04000000              cmp rdi, 4
5860 00002220 7213                        jb .xr11     ;Cant edit dr4, or 5. dr6 is read only
5861 00002222 4881FF07000000              cmp rdi, 7   ;Can only edit 7
5862 00002229 0F85D8FEFFFF                jne .bad_command
5863 0000222F 48FFCF                      dec rdi      ;Is the fifth entry in the table
5864 00002232 48FFCF                      dec rdi
5865                                  .xr11:
5866 00002235 48BD-                       mov rbp, .crlf
5866 00002237 [F02D000000000000]
5867 0000223F 66B80413                    mov ax, 1304h
5868 00002243 30FF                        xor bh, bh
5869 00002245 CD30                        int 30h
5870
5871 00002247 57                          push rdi     ;Save rdi
5872 00002248 48C1E702                    shl rdi, 2     ;Multiply by 4
5873 0000224C 66B90400                    mov cx, 4     ;4 chars to print
5874                                  .xr1:    ;Print register name
5875 00002250 8A87[60230000]              mov al, byte [.dregtbl + rdi]
5876 00002256 B40E                        mov ah, 0Eh
5877 00002258 CD30                        int 30h
5878 0000225A 66FFC7                      inc di
5879 0000225D 66FFC9                      dec cx
5880 00002260 75EE                        jnz .xr1
5881                                  ;Get the qword into the keybuffer
5882 00002262 5F                          pop rdi
5883 00002263 66B80104                    mov ax, 0401h     ;Process one qword
5884 00002267 E8B20A0000                  call .keyb
5885 0000226C 4885ED                      test rbp, rbp
5886 0000226F 0F84C6010000                jz .xcnoexit
5887 00002275 E8440A0000                  call .arg
5888 0000227A 3C01                        cmp al, 1
5889 0000227C 0F856D070000                jne .dmbadexit
5890
5891 00002282 488B4500                    mov rax, qword [rbp]     ;rax has the replacement value
5892 00002286 4885FF                      test rdi, rdi
5893 00002289 7504                        jnz .xr2
5894 0000228B 0F23C0                      mov dr0, rax
5895 0000228E C3                          ret
5896                                  .xr2:
5897 0000228F 48FFCF                      dec rdi
5898 00002292 7504                        jnz .xr3
5899 00002294 0F23C8                      mov dr1, rax
5900 00002297 C3                          ret
5901                                  .xr3:
5902 00002298 48FFCF                      dec rdi
5903 0000229B 7504                        jnz .xr4
5904 0000229D 0F23D0                      mov dr2, rax
5905 000022A0 C3                          ret
5906                                  .xr4:
5907 000022A1 48FFCF                      dec rdi
5908 000022A4 7504                        jnz .xr5
5909 000022A6 0F23D8                      mov dr3, rax
5910 000022A9 C3                          ret
5911                                  .xr5:
5912 000022AA 0F23F8                      mov dr7, rax
5913 000022AD C3                          ret
5914
5915                                  .dumpDebugRegs:
5916 000022AE 48BD-                       mov rbp, .crlf
5916 000022B0 [F02D000000000000]
5917 000022B8 66B80413                    mov ax, 1304h
5918 000022BC 30FF                        xor bh, bh
5919 000022BE CD30                        int 30h
5920 000022C0 4831ED                      xor rbp, rbp
5921 000022C3 4831FF                      xor rdi, rdi
5922
5923 000022C6 0F21F8                      mov rax, dr7
5924 000022C9 50                          push rax
5925 000022CA 0F21F0                      mov rax, dr6
5926 000022CD 50                          push rax
5927 000022CE 0F21D8                      mov rax, dr3
5928 000022D1 50                          push rax
5929 000022D2 0F21D0                      mov rax, dr2
5930 000022D5 50                          push rax
5931 000022D6 0F21C8                      mov rax, dr1
5932 000022D9 50                          push rax
5933 000022DA 0F21C0                      mov rax, dr0
5934 000022DD 50                          push rax
5935
5936                                  .ddr1:
5937 000022DE 4831C9                      xor rcx, rcx
```

```
5938 000022E1 4881FF03000000          cmp rdi, 3      ;3 registers per row
5939 000022E8 7450                    je .dregcrlf
5940                                .ddr11:
5941 000022EA 8A840D[60230000]        mov al, byte [.dregtbl + rbp + rcx]
5942 000022F1 B40E                    mov ah, 0Eh
5943 000022F3 CD30                    int 30h
5944 000022F5 66FFC1                  inc cx
5945 000022F8 6681F90400              cmp cx, 4
5946 000022FD 75EB                    jnz .ddr11
5947
5948 000022FF 48B908000000000000-     mov rcx, 8
5948 00002308 00
5949                                .ddr2:
5950 00002309 5B                      pop rbx     ;Get debug register
5951 0000230A 480FCB                  bswap rbx
5952                                .ddr21:
5953 0000230D B404                    mov ah, 04h
5954 0000230F 88D8                    mov al, bl
5955 00002311 CD30                    int 30h
5956 00002313 48C1EB08                shr rbx, 8h
5957 00002317 FEC9                    dec cl
5958 00002319 75F2                    jnz .ddr21
5959 0000231B 48FFC7                  inc rdi
5960
5961 0000231E B403                    mov ah, 3
5962 00002320 CD30                    int 30h
5963 00002322 80C203                  add dl, 3
5964 00002325 B402                    mov ah, 2
5965 00002327 CD30                    int 30h
5966 00002329 4881C504000000          add rbp, 4
5967 00002330 4881FD18000000          cmp rbp, 24 ;number of chars in the below typed string
5968 00002337 72A5                    jb .ddr1
5969
5970 00002339 C3                      ret
5971                                .dregcrlf:
5972 0000233A 4831FF                  xor rdi, rdi
5973 0000233D 55                      push rbp
5974 0000233E 50                      push rax
5975 0000233F 53                      push rbx
5976 00002340 48BD-                   mov rbp, .crlf
5976 00002342 [F02D000000000000]
5977 0000234A 48B804130000000000-     mov rax, 1304h
5977 00002353 00
5978 00002354 30FF                    xor bh, bh
5979 00002356 CD30                    int 30h
5980 00002358 5B                      pop rbx
5981 00002359 58                      pop rax
5982 0000235A 5D                      pop rbp
5983 0000235B E98AFFFFFF              jmp .ddr11
5984 00002360 4452303D4452313D44-   .dregtbl db "DR0=", "DR1=", "DR2=", "DR3=", "DR6=", "DR7="
5984 00002369 52323D4452333D4452-
5984 00002372 363D4452373D
5985
5986                                .xchangeReg:
5987 00002378 E89F020000              call .dumpReg
5988 0000237D 66B80413                mov ax, 1304h
5989 00002381 48BD-                   mov rbp, .crlf      ;Newline
5989 00002383 [F02D000000000000]
5990 0000238B CD30                    int 30h
5991
5992 0000238D 66B82E0E                mov ax, 0E2Eh     ;Print dot byte
5993 00002391 CD30                    int 30h
5994
5995 00002393 66B80101                mov ax, 0101h     ;Process one byte
5996 00002397 E882090000              call .keyb
5997 0000239C 4885ED                  test rbp, rbp
5998 0000239F 0F846FFCFFFF            jz .z11     ;If enter pressed, return to command line
5999 000023A5 E814090000              call .arg
6000 000023AA 3C01                    cmp al, 1
6001 000023AC 0F853D060000            jne .dmbadexit
6002
6003 000023B2 488B7D00                mov rdi, qword [rbp]      ;move this byte into rdi
6004 000023B6 4881FF11000000          cmp rdi, 11h
6005 000023BD 0F8744FDFFFF            ja .bad_command     ;If the user chooses a value greater than
                                                              11, exit!
6006
6007 000023C3 48BD-                   mov rbp, .crlf
6007 000023C5 [F02D000000000000]
6008 000023CD 66B80413                mov ax, 1304h
6009 000023D1 30FF                    xor bh, bh
6010 000023D3 CD30                    int 30h
6011
6012 000023D5 4881FF11000000          cmp rdi, 11h
```

```
6013 000023DC 7467                          je .xcflags ;If the user typed 10, then xchange flags
6014
6015 000023DE 57                            push rdi     ;Save rdi
6016 000023DF 48C1E702                      shl rdi, 2     ;Multiply by 4
6017 000023E3 66B90400                      mov cx, 4     ;4 chars to print
6018                                    .xcr1:
6019 000023E7 8A87[37280000]                mov al, byte [.regtbl + rdi]
6020 000023ED B40E                          mov ah, 0Eh
6021 000023EF CD30                          int 30h
6022 000023F1 66FFC7                        inc di
6023 000023F4 66FFC9                        dec cx
6024 000023F7 75EE                          jnz .xcr1
6025
6026 000023F9 5F                            pop rdi
6027 000023FA 66B80104                      mov ax, 0401h     ;Process one qword
6028 000023FE E81B090000                    call .keyb
6029 00002403 4885ED                        test rbp, rbp
6030 00002406 7433                          jz .xcnoexit
6031 00002408 E8B1080000                    call .arg
6032 0000240D 3C01                          cmp al, 1
6033 0000240F 0F85DA050000                  jne .dmbadexit
6034
6035 00002415 488B4500                      mov rax, qword [rbp]
6036 00002419 4881FF10000000                cmp rdi, 10h
6037 00002420 741A                          je .xcipchange
6038 00002422 488B1C25[EC010000]            mov rbx, qword [mcpUserBase]
6039 0000242A 4881C380000000                add rbx, 80h
6040 00002431 48C1E703                      shl rdi, 3  ;Multiply by 8
6041 00002435 4829FB                        sub rbx, rdi
6042 00002438 488903                        mov qword [rbx], rax     ;Replace element with rax
6043                                    .xcnoexit:
6044 0000243B C3                            ret
6045                                    .xcipchange:
6046 0000243C 48890425[F4010000]            mov qword [mcpUserRip], rax
6047 00002444 C3                            ret
6048                                    .xcflags:
6049 00002445 48B907000000000000-           mov rcx, 7
6049 0000244E 00
6050 0000244F 4831FF                        xor rdi, rdi
6051                                    .xcf1:
6052 00002452 8A87[8D280000]                mov al, byte [.rflgs + rdi]
6053 00002458 B40E                          mov ah, 0Eh
6054 0000245A CD30                          int 30h
6055 0000245C 66FFC7                        inc di
6056 0000245F 66FFC9                        dec cx
6057 00002462 75EE                          jnz .xcf1
6058
6059 00002464 66B80104                      mov ax, 0401h     ;Process one qword
6060 00002468 E8B1080000                    call .keyb
6061 0000246D 4885ED                        test rbp, rbp
6062 00002470 74C9                          jz .xcnoexit
6063 00002472 E847080000                    call .arg
6064 00002477 3C01                          cmp al, 1
6065 00002479 0F8570050000                  jne .dmbadexit
6066 0000247F 488B4500                      mov rax, qword [rbp]
6067 00002483 488B2C25[EC010000]            mov rbp, qword [mcpUserBase]
6068 0000248B 48894500                      mov qword [rbp], rax
6069 0000248F C3                            ret
6070                                    .inport:
6071 00002490 66B80413                      mov ax, 1304h
6072 00002494 30FF                          xor bh, bh
6073 00002496 48BD-                         mov rbp, .prompt2     ;Give the user the prompt
6073 00002498 [EE2D000000000000]
6074 000024A0 CD30                          int 30h
6075
6076 000024A2 66B80101                      mov ax, 0101h     ;Get 1 byte
6077 000024A6 E873080000                    call .keyb
6078 000024AB 4885ED                        test rbp, rbp
6079 000024AE 0F8453FCFFFF                  jz .bad_command
6080 000024B4 E805080000                    call .arg
6081 000024B9 3C01                          cmp al, 1
6082 000024BB 0F852E050000                  jne .dmbadexit
6083 000024C1 488B5500                      mov rdx, qword [rbp]     ;First arg, word io addr
6084 000024C5 48BD-                         mov rbp, .crlf
6084 000024C7 [F02D000000000000]
6085 000024CF 48B804130000000000-           mov rax, 1304h
6085 000024D8 00
6086 000024D9 30FF                          xor bh, bh
6087 000024DB CD30                          int 30h
6088 000024DD EC                            in al, dx
6089 000024DE B404                          mov ah, 04h
6090 000024E0 CD30                          int 30h
6091 000024E2 C3                            ret
```

```
6092
6093                                        .outport:
6094 000024E3 66B80413                          mov ax, 1304h
6095 000024E7 48BB07000000000000-               mov rbx, 7h
6095 000024F0 00
6096 000024F1 48BD-                             mov rbp, .prompt2      ;Give the user the prompt
6096 000024F3 [EE2D000000000000]
6097 000024FB CD30                              int 30h
6098 000024FD 66B80102                          mov ax, 0201h      ;Get 1 word
6099 00002501 E818080000                        call .keyb
6100 00002506 4885ED                            test rbp, rbp
6101 00002509 0F84F8FBFFFF                      jz .bad_command
6102 0000250F E8AA070000                        call .arg
6103 00002514 3C01                              cmp al, 1
6104 00002516 0F85D3040000                      jne .dmbadexit
6105 0000251C 488B5500                          mov rdx, qword [rbp]       ;First arg, word io addr
6106 00002520 B02E                              mov al, "."
6107 00002522 E898080000                        call .print
6108 00002527 66B80101                          mov ax, 0101h      ;Get 1 byte
6109 0000252B E8EE070000                        call .keyb
6110 00002530 4885ED                            test rbp, rbp
6111 00002533 0F84CEFBFFFF                      jz .bad_command
6112 00002539 E880070000                        call .arg
6113 0000253E 3C01                              cmp al, 1
6114 00002540 0F85A9040000                      jne .dmbadexit
6115 00002546 488B4500                          mov rax, qword [rbp]
6116 0000254A EE                                out dx, al
6117 0000254B C3                                ret
6118
6119                                        .hexCalc:
6120 0000254C 66B80413                          mov ax, 1304h
6121 00002550 30FF                              xor bh, bh
6122 00002552 48BD-                             mov rbp, .prompt2      ;Give the user the prompt
6122 00002554 [EE2D000000000000]
6123 0000255C CD30                              int 30h
6124 0000255E 66B80204                          mov ax, 0402h      ;Get 2 qwords
6125 00002562 E8B7070000                        call .keyb
6126 00002567 4885ED                            test rbp, rbp
6127 0000256A 0F8497FBFFFF                      jz .bad_command
6128 00002570 E849070000                        call .arg
6129
6130 00002575 3C02                              cmp al, 2
6131 00002577 0F8572040000                      jne .dmbadexit
6132
6133 0000257D 4C8B4508                          mov r8, qword [rbp + 8]  ;First number
6134 00002581 4C8B4D00                          mov r9, qword [rbp]           ;Second number
6135 00002585 4F8D1408                          lea r10, qword [r8+r9]
6136
6137 00002589 48BD-                             mov rbp, .crlf
6137 0000258B [F02D000000000000]
6138 00002593 48B804130000000000-               mov rax, 1304h
6138 0000259C 00
6139 0000259D 30FF                              xor bh, bh
6140 0000259F CD30                              int 30h
6141
6142 000025A1 4C89C2                            mov rdx, r8
6143 000025A4 E856000000                        call .hcprintquad
6144 000025A9 B02B                              mov al, "+"
6145 000025AB E80F080000                        call .print
6146 000025B0 4C89CA                            mov rdx, r9
6147 000025B3 E847000000                        call .hcprintquad
6148 000025B8 B03D                              mov al, "="
6149 000025BA E800080000                        call .print
6150 000025BF 4C89D2                            mov rdx, r10
6151 000025C2 E838000000                        call .hcprintquad
6152
6153 000025C7 48B804130000000000-               mov rax, 1304h
6153 000025D0 00
6154 000025D1 30FF                              xor bh, bh
6155 000025D3 CD30                              int 30h
6156
6157 000025D5 4C89C2                            mov rdx, r8
6158 000025D8 E822000000                        call .hcprintquad
6159 000025DD B02D                              mov al, "-"
6160 000025DF E8DB070000                        call .print
6161 000025E4 4C89CA                            mov rdx, r9
6162 000025E7 E813000000                        call .hcprintquad
6163 000025EC B03D                              mov al, "="
6164 000025EE E8CC070000                        call .print
6165 000025F3 4D29C8                            sub r8, r9
6166 000025F6 4C89C2                            mov rdx, r8
6167 000025F9 E801000000                        call .hcprintquad
6168 000025FE C3                                ret
```

```
                                        .hcprintquad:
                                        ;Takes whats in rdx, and prints it
000025FF 480FCA                             bswap rdx
00002602 48B908000000000000-                mov rcx, 8
0000260B 00
                                        .hcpq1:
0000260C 88D0                               mov al, dl
0000260E B404                               mov ah, 04h
00002610 CD30                               int 30h
00002612 48C1EA08                           shr rdx, 8
00002616 66FFC9                             dec cx
00002619 75F1                               jnz .hcpq1
0000261B C3                                 ret

                                        .dumpReg:
0000261C 48BD-                              mov rbp, .crlf
0000261E [F02D000000000000]

00002626 66B80413                           mov ax, 1304h
0000262A 30FF                               xor bh, bh
0000262C CD30                               int 30h
0000262E 4831ED                             xor rbp, rbp
00002631 4831FF                             xor rdi, rdi
00002634 4831F6                             xor rsi, rsi
00002637 488B3425[EC010000]                 mov rsi, qword [mcpUserBase]
0000263F 4881C680000000                     add rsi, 80h
                                        .dreg1:
00002646 4831C9                             xor rcx, rcx
00002649 4881FF03000000                     cmp rdi, 3
00002650 0F84BB010000                       je .regcrlf
                                        .dreg11:     ;Print register name
00002656 8A840D[37280000]                   mov al, byte [.regtbl+rbp+rcx]
0000265D B40E                               mov ah, 0Eh
0000265F CD30                               int 30h
00002661 66FFC1                             inc cx
00002664 6681F90400                         cmp cx, 4h
00002669 75EB                               jnz .dreg11
                                        .dreg2:
0000266B 48B908000000000000-                mov rcx, 8h
00002674 00

                                        ;Now print register value
00002675 488B1E                             mov rbx, qword [rsi]     ;Get qword from storage
00002678 81EE08000000                       sub esi, 8
0000267E 480FCB                             bswap rbx     ;Change endianness
                                        .dreg21:
00002681 B404                               mov ah, 04h
00002683 88D8                               mov al, bl
00002685 CD30                               int 30h
00002687 48C1EB08                           shr rbx, 8h     ;Shift down by a byte
0000268B FEC9                               dec cl
0000268D 75F2                               jnz .dreg21
0000268F 48FFC7                             inc rdi

00002692 B403                               mov ah, 3
00002694 CD30                               int 30h
00002696 80C203                             add dl, 3
00002699 B402                               mov ah, 2
0000269B CD30                               int 30h
0000269D 4881C504000000                     add rbp, 4
000026A4 4881FD40000000                     cmp rbp, 40h
000026AB 7299                               jb .dreg1


                                        ;Print RIP
                                        .drip0:
000026AD 4831C9                             xor rcx, rcx
                                        .drip1:
                                        ;Print name
000026B0 8A840D[37280000]                   mov al, byte [.regtbl+rbp+rcx]
000026B7 B40E                               mov ah, 0Eh
000026B9 CD30                               int 30h
000026BB 66FFC1                             inc cx
000026BE 6681F90400                         cmp cx, 4h
000026C3 75EB                               jne .drip1

000026C5 48B908000000000000-                mov rcx, 8
000026CE 00
000026CF 488B3425[F4010000]                 mov rsi, qword [mcpUserRip]
000026D7 480FCE                             bswap rsi
                                        .drip2:
                                        ;Print value
000026DA B404                               mov ah, 04h
000026DC 4088F0                             mov al, sil
000026DF CD30                               int 30h
```

```
000026E1 48C1EE08                    shr rsi, 8h     ;Shift down by a byte
000026E5 FEC9                        dec cl
000026E7 75F1                        jnz .drip2
000026E9 4881C504000000              add rbp, 4     ;Offset into table

000026F0 55                          push rbp
000026F1 48BD-                       mov rbp, .ipstrg
000026F3 [9428000000000000]
000026FB 66B80413                    mov ax, 1304h
000026FF CD30                        int 30h
00002701 B107                        mov cl, 7
00002703 488B0425[EC010000]          mov rax, qword [mcpUserBase]
0000270B 488B4008                    mov rax, qword [rax + 08h]   ;Get the old stack pointer
0000270F 488B18                      mov rbx, qword [rax]      ;Get the address of 8 bytes at that
                                                               instruction
00002712 488B1B                      mov rbx, qword [rbx]      ;Get the bytes
00002715 88D8                        mov al, bl
00002717 B404                        mov ah, 04h
00002719 CD30                        int 30h
0000271B 48C1EB08                    shr rbx, 8
0000271F B40E                        mov ah, 0Eh    ;Add a space to indicate mod r/m + optionals
00002721 B02D                        mov al, '-'
00002723 CD30                        int 30h
                                 .ssep0:
00002725 88D8                        mov al, bl
00002727 B404                        mov ah, 04h
00002729 CD30                        int 30h
0000272B 48C1EB08                    shr rbx, 8
0000272F FEC9                        dec cl
00002731 75F2                        jnz .ssep0

00002733 48BD-                       mov rbp, .crlf
00002735 [F02D000000000000]
0000273D 48B804130000000000-         mov rax, 1304h
00002746 00
00002747 48BB07000000000000-         mov rbx, 7h
00002750 00
00002751 CD30                        int 30h
00002753 5D                          pop rbp

00002754 668CC8                      mov ax, cs
00002757 E87D000000                  call .dsegregwrite
0000275C 668CD8                      mov ax, ds
0000275F E875000000                  call .dsegregwrite
00002764 668CC0                      mov ax, es
00002767 E86D000000                  call .dsegregwrite
0000276C 668CD0                      mov ax, ss
0000276F E865000000                  call .dsegregwrite
00002774 668CE0                      mov ax, fs
00002777 E85D000000                  call .dsegregwrite
0000277C 668CE8                      mov ax, gs
0000277F E855000000                  call .dsegregwrite

00002784 55                          push rbp
00002785 48BD-                       mov rbp, .crlf
00002787 [F02D000000000000]
0000278F 48B804130000000000-         mov rax, 1304h
00002798 00
00002799 30FF                        xor bh, bh
0000279B CD30                        int 30h
0000279D 5D                          pop rbp
                                 .drflagwrite:
0000279E 4831C9                      xor rcx, rcx
                                 .drflg1:     ;Print register name
000027A1 8A840D[37280000]            mov al, byte [.regtbl+rbp+rcx]
000027A8 B40E                        mov ah, 0Eh
000027AA CD30                        int 30h
000027AC 48FFC1                      inc rcx
000027AF 4881F907000000              cmp rcx, 7
000027B6 75E9                        jnz .drflg1

000027B8 48FFC1                      inc rcx
000027BB 488B1425[EC010000]          mov rdx, qword [mcpUserBase]      ;Get flags into rdx
000027C3 488B12                      mov rdx, qword [rdx]
000027C6 480FCA                      bswap rdx
                                 .drflg2:
000027C9 B404                        mov ah, 04h
000027CB 88D0                        mov al, dl
000027CD CD30                        int 30h
000027CF 48C1EA08                    shr rdx, 8
000027D3 48FFC9                      dec rcx
000027D6 75F1                        jnz .drflg2

```

```
                                    .dregexit:
000027D8 C3                             ret
                                    .dsegregwrite:
000027D9 4831C9                         xor rcx, rcx
000027DC 6689C2                         mov dx, ax      ;save
                                    .dsegreg1:      ;Print register name
000027DF 8A840D[37280000]               mov al, byte [.regtbl+rbp+rcx]
                                        ;xor bh, bh
000027E6 B40E                           mov ah, 0Eh
000027E8 CD30                           int 30h
000027EA 48FFC1                         inc rcx
000027ED 4881F903000000                 cmp rcx, 3
000027F4 75E9                           jnz .dsegreg1

000027F6 88F0                           mov al, dh
000027F8 B404                           mov ah, 04h
000027FA CD30                           int 30h
000027FC 88D0                           mov al, dl
000027FE B404                           mov ah, 04h
00002800 CD30                           int 30h

00002802 4801CD                         add rbp, rcx
00002805 B403                           mov ah, 3
00002807 CD30                           int 30h
00002809 80C202                         add dl, 2
0000280C B402                           mov ah, 2
0000280E CD30                           int 30h
00002810 C3                             ret


                                    .regcrlf:
00002811 4831FF                         xor rdi, rdi
00002814 55                             push rbp
00002815 50                             push rax
00002816 53                             push rbx
00002817 48BD-                          mov rbp, .crlf
00002819 [F02D000000000000]
00002821 48B804130000000000-            mov rax, 1304h
0000282A 00
0000282B 30FF                           xor bh, bh
0000282D CD30                           int 30h
0000282F 5B                             pop rbx
00002830 58                             pop rax
00002831 5D                             pop rbp
00002832 E91FFEFFFF                     jmp .dreg11

00002837 5241583D5242583D52-        .regtbl  db "RAX=", "RBX=", "RCX=", "RDX=", "RSI=", "RDI=", "R8 =",
00002840 43583D5244583D5253-
00002849 493D5244493D523820-
00002852 3D
00002853 5239203D5231303D52-                 db "R9 =", "R10=", "R11=", "R12=", "R13=", "R14=", "R15=",
0000285C 31313D5231323D5231-
00002865 333D5231343D523135-
0000286E 3D
0000286F 5242503D5253503D52-                 db "RBP=", "RSP=", "RIP=","CS=", "DS=", "ES=", "SS=",
                                                                     "FS=",
00002878 49503D43533D44533D-
00002881 45533D53533D46533D
0000288A 47533D                              db "GS="
0000288D 52464C4147533D             .rflgs   db "RFLAGS="
00002894 20205B5249505D3D00         .ipstrg: db "  [RIP]=",0
                                    .dumpMemory:
0000289D 50                             push rax
0000289E 53                             push rbx
0000289F 51                             push rcx
000028A0 52                             push rdx
000028A1 57                             push rdi
000028A2 56                             push rsi
000028A3 55                             push rbp
000028A4 4150                           push r8
000028A6 4151                           push r9

000028A8 66B80413                       mov ax, 1304h
000028AC 48BD-                          mov rbp, .prompt2       ;Give the user the prompt
000028AE [EE2D000000000000]
000028B6 CD30                           int 30h
000028B8 66B80204                       mov ax, 0402h       ;Get 2 dwords
000028BC E85D040000                     call .keyb
000028C1 4885ED                         test rbp, rbp
000028C4 0F8442010000                   jz .dmnoargs
000028CA B002                           mov al, 2       ;Number of user inputs to convert
000028CC E8ED030000                     call .arg
000028D1 FEC8                           dec al
000028D3 0F843D010000                   jz .dmnoargs1
```

```
000028D9 FEC8                          dec al      ;More than 2 args, error
000028DB 0F850E010000                  jnz .dmbadexit
000028E1 4C8B4508                      mov r8, qword [rbp + 8]      ;First argument, #Base
000028E5 4C8B4D00                      mov r9, qword [rbp]      ;Second argument, #Number of bytes
                                   .dmmain00:
000028E9 4D85C9                        test r9, r9
000028EC 0F84FD000000                  jz .dmbadexit
000028F2 66B80413                      mov ax, 1304h
000028F6 48BD-                         mov rbp, .crlf
000028F8 [F02D000000000000]
00002900 CD30                          int 30h
00002902 4C89C2                        mov rdx, r8
00002905 E85C010000                    call .dmcsaddrprint
0000290A 30FF                          xor bh, bh
0000290C B403                          mov ah, 03h
0000290E CD30                          int 30h
00002910 B219                          mov dl, 25
00002912 B402                          mov ah, 02h
00002914 CD30                          int 30h
00002916 4C89C6                        mov rsi, r8     ;point rsi at r8
00002919 48F7C608000000                test rsi, 08h      ;If it starts between a qword and para

00002920 48F7C60F000000                test rsi, 0Fh
00002927 7430                          jz .dmmain0     ;If it starts on paragraph bndry, continue as
                                                                  normal
00002929 56                            push rsi
0000292A 4881E60F000000                and rsi, 0Fh
00002931 4881FE08000000                cmp rsi, 8
00002938 720F                          jb .dmmain01
0000293A 48B901000000000000-           mov rcx, 1
00002943 00
00002944 E8F9000000                    call .dmal1     ;Print one space
                                   .dmmain01:
00002949 5E                            pop rsi
0000294A 48B801000000000000-           mov rax, 1
00002953 00
00002954 E8D0000000                    call .dmalign


                                   .dmmain0:
00002959 4889F7                        mov rdi, rsi     ;Save start point at rdi
0000295C 4151                          push r9
                                   .dmmain1:     ;This loop prints a line
0000295E AC                            lodsb
0000295F B404                          mov ah, 4h
00002961 CD30                          int 30h
00002963 49FFC9                        dec r9
00002966 7416                          jz .dmmain2
00002968 48F7C608000000                test rsi, 08h      ;This is zero iff rsi has bit 4 set
0000296F 0F85D9000000                  jnz .dmhyphen1
00002975 48F7C60F000000                test rsi, 0Fh      ;This is zero iff lower nybble is zero
0000297C 75E0                          jnz .dmmain1
                                   .dmmain2:
                                   ;Now the numbers have been printed, get the ascii row too
                                   ;First check if numbers have stopped short of 16
0000297E 4D85C9                        test r9, r9
00002981 7500                          jnz .dmmain21      ;end of row

                                   .dmmain21:
00002983 4159                          pop r9
00002985 30FF                          xor bh, bh
00002987 B403                          mov ah, 03h
00002989 CD30                          int 30h
0000298B B23E                          mov dl, 62
0000298D B402                          mov ah, 02h
0000298F CD30                          int 30h
00002991 4889FE                        mov rsi, rdi     ;Reload value
00002994 48F7C60F000000                test rsi, 0Fh
0000299B 7408                          jz .dmmain3     ;If it starts on paragraph bndry, continue as
                                                                  normal
0000299D 4831C0                        xor rax, rax     ;no shift
000029A0 E884000000                    call .dmalign

                                   .dmmain3:
000029A5 AC                            lodsb
000029A6 49FFC9                        dec r9
000029A9 3C30                          cmp al, 30h
000029AB 660F420425-                   cmovb ax, word [.dmdot]      ;bring the dot to ax
000029B0 [8D2A0000]
000029B4 B40E                          mov ah, 0Eh
000029B6 CD30                          int 30h
000029B8 4D85C9                        test r9, r9
000029BB 7443                          jz .dmexit
000029BD 48F7C60F000000                test rsi, 0Fh      ;Check if lower nybble is 0
```

```
000029C4 75DF                           jnz .dmmain3

000029C6 48BD-                          mov rbp, .crlf
000029C8 [F02D000000000000]
000029D0 66B80413                       mov ax, 1304h
000029D4 CD30                           int 30h

000029D6 4889F2                         mov rdx, rsi
000029D9 E888000000                     call .dmcsaddrprint

000029DE B403                           mov ah, 03h
000029E0 30FF                           xor bh, bh
000029E2 CD30                           int 30h
000029E4 B219                           mov dl, 25
000029E6 B402                           mov ah, 02h
000029E8 CD30                           int 30h
000029EA E96AFFFFFF                     jmp .dmmain0

                                    .dmbadexit:
000029EF 48BD-                          mov rbp, .dmbadargs
000029F1 [8F2A000000000000]
000029F9 66B80413                       mov ax, 1304h
000029FD CD30                           int 30h
000029FF C3                             ret;Reload program, error!
                                    .dmexit:
00002A00 4159                           pop r9
00002A02 4158                           pop r8
00002A04 5D                             pop rbp
00002A05 5E                             pop rsi
00002A06 5F                             pop rdi
00002A07 5A                             pop rdx
00002A08 59                             pop rcx
00002A09 5B                             pop rbx
00002A0A 58                             pop rax
00002A0B C3                             ret
                                    .dmnoargs:
00002A0C 4C8B0425[F4010000]             mov r8, qword [mcpUserRip]
                                        ;add r8, 180h    ;Add 180 bytes, to bypass internal work areas
00002A14 EB04                           jmp short .dmnoargscommon
                                    .dmnoargs1:
00002A16 4C8B4500                       mov r8, qword [rbp]
                                    .dmnoargscommon:
00002A1A 49B980000000000000-            mov r9, 80h
00002A23 00
00002A24 E9C0FEFFFF                     jmp .dmmain00

                                    .dmalign:     ;Print blank chars for offset
                                    ;Works out from rsi
                                    ;rax contains value for shl
00002A29 56                             push rsi
00002A2A 4889F1                         mov rcx, rsi
00002A2D 4881E1F0FFFFFF                 and rcx, 0FFFFFFFFFFFFFFF0h    ;Round down
00002A34 4829CE                         sub rsi, rcx
00002A37 4887CE                         xchg rcx, rsi
00002A3A 5E                             pop rsi
00002A3B 4891                           xchg rcx, rax
00002A3D 48D3E0                         shl rax, cl
00002A40 4891                           xchg rcx, rax
                                    .dmal1:
00002A42 66B8200E                       mov ax, 0E20h
00002A46 CD30                           int 30h
00002A48 48FFC9                         dec rcx
00002A4B 75F5                           jnz .dmal1
00002A4D C3                             ret

                                    .dmhyphen1:
00002A4E 48F7C607000000                 test rsi, 07h    ;If the rest of the bits are set, go away
00002A55 0F8503FFFFFF                   jnz .dmmain1
00002A5B 66B82D0E                       mov ax, 0E2Dh    ;2dh=”–”
00002A5F CD30                           int 30h
00002A61 E9F8FEFFFF                     jmp .dmmain1
                                    .dmcsaddrprint:
00002A66 668CC8                         mov ax, cs    ;Get current code segment into ax
00002A69 88E0                           mov al, ah
00002A6B B404                           mov ah, 04h    ;print upper byte
00002A6D CD30                           int 30h
00002A6F 668CC8                         mov ax, cs
00002A72 B404                           mov ah, 04h
00002A74 CD30                           int 30h        ;print lower byte
00002A76 66B83A0E                       mov ax, 0E3Ah

00002A7A B108                           mov cl, 8
00002A7C CD30                           int 30h
```

```

                                     .dmrollprint:
                                     ;Takes whats in rdx, rols left by one byte, prints al
                                     ;repeats, cl times.
00002A7E 48C1C208                        rol rdx, 8
00002A82 88D0                            mov al, dl
00002A84 B404                            mov ah, 04h
00002A86 CD30                            int 30h
00002A88 FEC9                            dec cl
00002A8A 75F2                            jnz .dmrollprint
00002A8C C3                              ret
00002A8D 2E00                        .dmdot:     db     ".",0
00002A8F 0A0D53796E74617820-         .dmbadargs:     db 0Ah, 0Dh,"Syntax error",0
00002A98 6572726F7200


                                     .editMemory:
00002A9E 66B80413                        mov ax, 1304h
00002AA2 30FF                            xor bh, bh
00002AA4 48BD-                           mov rbp, .prompt2     ;Give the user the prompt
00002AA6 [EE2D000000000000]
00002AAE CD30                            int 30h

00002AB0 66B80104                        mov ax, 0401h      ;Get up to one qword
00002AB4 E865020000                      call .keyb
00002AB9 4885ED                          test rbp, rbp          ;No chars entered?
00002ABC 0F8445F6FFFF                    jz .bad_command
00002AC2 E8F7010000                      call .arg
00002AC7 488B7D00                        mov rdi, qword [rbp]      ;First arg, Dword Address

00002ACB 48BD-                           mov rbp, .crlf
00002ACD [F02D000000000000]
00002AD5 30FF                            xor bh, bh
00002AD7 48B80413000000000-              mov rax, 1304h
00002AE0 00
00002AE1 CD30                            int 30h

00002AE3 4889FE                          mov rsi, rdi
00002AE6 AC                              lodsb     ;Get byte into al
00002AE7 B404                            mov ah, 04
00002AE9 CD30                            int 30h
00002AEB B02E                            mov al, "."
00002AED E8CD020000                      call .print
00002AF2 66B80101                        mov ax, 0101h      ;Get 1 byte
00002AF6 E823020000                      call .keyb
00002AFB 4885ED                          test rbp, rbp          ;No chars entered?
00002AFE 0F84EBFEFFFF                    jz .dmbadexit
00002B04 E8B5010000                      call .arg
00002B09 4889EE                          mov rsi, rbp     ;Point rsi to the stack
00002B0C A4                              movsb              ;Move byte from rsi to rdi

00002B0D C3                              ret

                                     .jumpProc:
00002B0E 66B80413                        mov ax, 1304h
00002B12 30FF                            xor bh, bh
00002B14 48BD-                           mov rbp, .prompt2     ;Give the user the prompt
00002B16 [EE2D000000000000]
00002B1E CD30                            int 30h
00002B20 66B80104                        mov ax, 0401h      ;Get 1 dword (forbit going too high eh?)
00002B24 E8F5010000                      call .keyb
00002B29 4885ED                          test rbp, rbp          ;No chars entered?
00002B2C 743B                            jz .proceedDefault
00002B2E E88B010000                      call .arg
00002B33 FEC8                            dec al
00002B35 0F85B4FEFFFF                    jnz .dmbadexit
00002B3B 488B6D00                        mov rbp, qword [rbp]      ;First argument, Address of procedure
00002B3F 48892C25[F4010000]              mov qword [mcpUserRip], rbp    ;Move first argument into new Rip
00002B47 E81B010000                      call .loadMainRegisters
00002B4C 488B6008                        mov rsp, qword [rax + 08h]
00002B50 488B0425[04020000]              mov rax, qword [mcpUserRaxStore]
00002B58 48CF                            iretq
                                     .singleStep:
                                     ;When s is pressed, the program proceeds by a single step.
                                     ;Sets trap flag on
00002B5A 488B0425[EC010000]              mov rax, qword [mcpUserBase]
00002B62 48810800010000                  or qword [rax + 00h], 100h  ;Set trap flag on
                                     .proceedDefault:
00002B69 E8F9000000                      call .loadMainRegisters
00002B6E 488B6008                        mov rsp, qword [rax + 08h]
00002B72 488B0425[04020000]              mov rax, qword [mcpUserRaxStore]
00002B7A 48CF                            iretq

                                     .storageRead:
```

```
6630 00002B7C 50                         push rax
6631 00002B7D B800820000                  mov eax, 8200h ;LBA Read function
6632 00002B82 EB06                        jmp short .storageCommon
6633                                  .storageWrite:
6634 00002B84 50                          push rax
6635 00002B85 B800830000                  mov eax, 8300h ;LBA Write function
6636                                  .storageCommon:
6637                                  ;l/w [Address Buffer] [Drive] [Sector] [Count]
6638 00002B8A 53                          push rbx
6639 00002B8B 51                          push rcx
6640 00002B8C 52                          push rdx
6641 00002B8D 56                          push rsi
6642 00002B8E 57                          push rdi
6643 00002B8F 55                          push rbp
6644
6645 00002B90 89C6                        mov esi, eax          ;Save LBA r/w function number in esi
6646 00002B92 66B80413                    mov ax, 1304h
6647 00002B96 48BD-                       mov rbp, .prompt2     ;Give the user the prompt
6647 00002B98 [EE2D000000000000]
6648 00002BA0 CD30                        int 30h
6649
6650 00002BA2 66B80404                    mov ax, 0404h      ;Get 4 qwords
6651 00002BA6 E873010000                  call .keyb
6652 00002BAB 4885ED                      test rbp, rbp
6653 00002BAE 7452                        jz .storageError
6654 00002BB0 B004                        mov al, 4     ;Number of user inputs to convert
6655 00002BB2 E807010000                  call .arg
6656 00002BB7 3C04                        cmp al, 4   ;If not 4 arguments, fail
6657 00002BB9 7547                        jne .storageError
6658 00002BBB BF05000000                  mov edi, 5
6659                                  .sc0:
6660 00002BC0 89F0                        mov eax, esi                    ;Get back LBA r/w function number
                                                                         into eax
6661 00002BC2 488B5D18                    mov rbx, qword [rbp + 24]    ;First argument, Address buffer
6662 00002BC6 488B5510                    mov rdx, qword [rbp + 16]    ;dl ONLY, Second argument
6663 00002BCA 4881E2FF000000              and rdx, 0FFh
6664 00002BD1 488B4D08                    mov rcx, qword [rbp + 08]    ;LBA starting sector, third argument
6665 00002BD5 488B7500                    mov rsi, qword [rbp]         ;Sector count into rsi
6666 00002BD9 4881E6FF000000              and rsi, 0FFh                ;Sector count can be at most 255
6667 00002BE0 09F0                        or eax, esi                  ;Add the sector count to eax
6668 00002BE2 89C6                        mov esi, eax                 ;Copy the function number into esi
                                                                       for failures
6669 00002BE4 81E600FF0000                and esi, 0FF00h              ;Save only byte two of esi, the
                                                                       function number
6670 00002BEA CD33                        int 33h
6671 00002BEC 7308                        jnc .storageExit
6672
6673 00002BEE 31C0                        xor eax, eax
6674 00002BF0 CD33                        int 33h
6675 00002BF2 FFCF                        dec edi
6676 00002BF4 75CA                        jnz .sc0
6677                                  .storageExit:
6678 00002BF6 5D                          pop rbp
6679 00002BF7 5F                          pop rdi
6680 00002BF8 5E                          pop rsi
6681 00002BF9 5A                          pop rdx
6682 00002BFA 59                          pop rcx
6683 00002BFB 5B                          pop rbx
6684 00002BFC 58                          pop rax
6685 00002BFD E912F4FFFF                  jmp MCP_int.z11
6686                                  .storageError:
6687 00002C02 5D                          pop rbp
6688 00002C03 5F                          pop rdi
6689 00002C04 5E                          pop rsi
6690 00002C05 5A                          pop rdx
6691 00002C06 59                          pop rcx
6692 00002C07 5B                          pop rbx
6693 00002C08 58                          pop rax
6694 00002C09 E9F9F4FFFF                  jmp .bad_command
6695                                  .restartMcp:
6696 00002C0E CD40                        int 40h       ;To allow returning to DOS
6697                                  .clearscreen:
6698 00002C10 B307                        mov bl, 07h
6699 00002C12 E8E0D4FFFF                  call cls
6700 00002C17 E9F8F3FFFF                  jmp MCP_int.z11
6701                                  .storeMainRegisters:
6702 00002C1C 9C                          pushfq
6703 00002C1D 8F00                        pop qword [rax + 00h]         ;Flags
6704                                      ;mov qword [rax + 08h], rsp
6705 00002C1F 48896810                    mov qword [rax + 10h], rbp
6706 00002C23 4C897818                    mov qword [rax + 18h], r15
6707 00002C27 4C897020                    mov qword [rax + 20h], r14
6708 00002C2B 4C896828                    mov qword [rax + 28h], r13
```

```
00002C2F 4C896030                      mov qword [rax + 30h], r12
00002C33 4C895838                      mov qword [rax + 38h], r11
00002C37 4C895040                      mov qword [rax + 40h], r10
00002C3B 4C894848                      mov qword [rax + 48h], r9
00002C3F 4C894050                      mov qword [rax + 50h], r8
00002C43 48897858                      mov qword [rax + 58h], rdi
00002C47 48897060                      mov qword [rax + 60h], rsi
00002C4B 48895068                      mov qword [rax + 68h], rdx
00002C4F 48894870                      mov qword [rax + 70h], rcx
00002C53 48895878                      mov qword [rax + 78h], rbx
00002C57 488B1C25[04020000]            mov rbx, qword [mcpUserRaxStore]
00002C5F 48899880000000                mov qword [rax + 80h], rbx   ;Store rax
00002C66 C3                            ret

                                   .loadMainRegisters:
00002C67 488B0425[EC010000]            mov rax, qword [mcpUserBase]
00002C6F 488B5008                      mov rdx, qword [rax + 08h]   ;Get old stack pointer into rdx
00002C73 488B1C25[F4010000]            mov rbx, qword [mcpUserRip]
00002C7B 48891A                        mov qword [rdx], rbx      ;Move the userRip into rdx
00002C7E 488B18                        mov rbx, qword [rax + 00h]
00002C81 48895A10                      mov qword [rdx + 10h], rbx   ;Move new flags into position on stack
00002C85 488B5878                      mov rbx, qword [rax + 78h]
00002C89 488B4870                      mov rcx, qword [rax + 70h]
00002C8D 488B5068                      mov rdx, qword [rax + 68h]
00002C91 488B7060                      mov rsi, qword [rax + 60h]
00002C95 488B7858                      mov rdi, qword [rax + 58h]
00002C99 4C8B4050                      mov r8,  qword [rax + 50h]
00002C9D 4C8B4848                      mov r9,  qword [rax + 48h]
00002CA1 4C8B5040                      mov r10, qword [rax + 40h]
00002CA5 4C8B5838                      mov r11, qword [rax + 38h]
00002CA9 4C8B6030                      mov r12, qword [rax + 30h]
00002CAD 4C8B6828                      mov r13, qword [rax + 28h]
00002CB1 4C8B7020                      mov r14, qword [rax + 20h]
00002CB5 4C8B7818                      mov r15, qword [rax + 18h]
00002CB9 488B6810                      mov rbp, qword [rax + 10h]
00002CBD C3                            ret
                                   ;ARG    PROC    NEAR
                                   .arg:
                                   ;Number of arguments expected in buffer in al (could early terminate due to
                                   ; enter)
                                   ;Converted qwords stored on stack with al indicating how many processed
                                   ;rbp returns the base of the stack of stored arguments
                                   ;rdx is our scratch register
00002CBE 53                            push rbx
00002CBF 51                            push rcx
00002CC0 52                            push rdx
00002CC1 56                            push rsi
00002CC2 4889E5                        mov rbp, rsp      ;Preserve stack pointer
00002CC5 488B3425[FC010000]            mov rsi, qword [mcpUserkeybf]
00002CCD 30C9                          xor cl, cl           ;Keep track of how many arguments processed

                                   .a01:
00002CCF 4831D2                        xor rdx, rdx      ;Clean rdx

                                   .a1:
00002CD2 AC                            lodsb          ;Get the first byte into al
00002CD3 3C11                          cmp al, 11h      ;Offset 11h is the space key
00002CD5 740E                          jz .a2
00002CD7 3C12                          cmp al, 12h      ;Offset 12h is the enter key
00002CD9 740F                          jz .aexit          ;Anyway, enter is exit!
00002CDB 48C1E204                      shl rdx, 4      ;Go to next sig fig
00002CDF 08C2                          or dl, al      ;Put this byte into dl
00002CE1 7013                          jo .error
00002CE3 EBED                          jmp short .a1

                                   .a2:
00002CE5 52                            push rdx      ;Store argument on stack
00002CE6 FEC1                          inc cl          ;One more argument processed
00002CE8 EBE5                          jmp short .a01

                                   .aexit:
00002CEA 480FB6C1                      movzx rax, cl      ;Return #of args processed
00002CEE 4887E5                        xchg rsp, rbp      ;rbp points to bottom of argument stack
00002CF1 5E                            pop rsi
00002CF2 5A                            pop rdx
00002CF3 59                            pop rcx
00002CF4 5B                            pop rbx
00002CF5 C3                            ret

                                   .error:
00002CF6 48BD-                         mov rbp, .emsg
00002CF8 [0D2D000000000000]

00002D00 30FF                          xor bh, bh
00002D02 66B80413                      mov ax, 1304h
00002D06 CD30                          int 30h
00002D08 5E                            pop rsi
```

```
00002D09 5A                                pop rdx
00002D0A 59                                pop rcx
00002D0B 5B                                pop rbx
00002D0C C3                                ret
00002D0D 0A0D417267756D656E–         .emsg:      db 0Ah, 0Dh,"Argument error",0
00002D16 74206572726F7200
                                     ;ARG     ENDP

                                     ;KEYB       PROC       NEAR
                                     .keyb:
                                     ;Number of arguments to accept is passed in al, in units of ah
                                     ;ah=4 => Qwords, ah=3 => dwords... ah=2 => word, ah=1 => bytes
                                     ;Arguments are stored in buffer, after USB area, of size 2*al qwords
                                     ;All arguments CAN be up to qword in size, though not all subprogs,
                                     ;     may use the full qword.
                                     ;ch returns number of chars not processed
00002D1E 50                                push rax
00002D1F 53                                push rbx
                                           ;push rcx
00002D20 57                                push rdi
00002D21 52                                push rdx

00002D22 4831C9                            xor rcx, rcx
00002D25 88C1                              mov cl, al
00002D27 51                                push rcx
00002D28 88E1                              mov cl, ah
00002D2A D2E0                              shl al, cl  ;Multiply by 16 to get the number of bytes needed
                                                                   w/o spaces
00002D2C 59                                pop rcx
00002D2D 00C8                              add al, cl  ;Add space for spaces
00002D2F FEC8                              dec al      ;We reserve one space for a "non–user accessible"
                                                                   EOL at the end

00002D31 488B3C25[FC010000]                mov rdi, qword [mcpUserkeybf]     ;Data area in command tail
00002D39 50                                push rax
00002D3A 48B810000000000000–               mov rax, 10h
00002D43 00
00002D44 57                                push rdi
00002D45 F348AB                            rep stosq     ;Clear buffer space for al qwords (max 8)
00002D48 5F                                pop rdi
00002D49 58                                pop rax

00002D4A 88C5                              mov ch, al     ;Rememebr 1 Qword is 16 ASCII chars
00002D4C 88C2                              mov dl, al     ;Let dl save this number
00002D4E 4831ED                            xor rbp, rbp     ;Cheap cop out char counter

                                     .k1:
00002D51 6631C0                            xor ax, ax
00002D54 CD36                              int 36h
00002D56 3C71                              cmp al, "q"     ;Quit option
00002D58 0F84B6F2FFFF                      je .z11
00002D5E 3C08                              cmp al, 08h     ;Backspace
00002D60 7447                              je .kb2
00002D62 3C0D                              cmp al, 0Dh     ;Enter key pressed, we done
00002D64 7438                              je .kend

00002D66 84ED                              test ch, ch     ;Have we filled a 16 char buffer?
00002D68 74E7                              jz .k1         ;Yes, await control key

00002D6A 4889FB                            mov rbx, rdi     ;Save current offset into bbuffer
00002D6D 51                                push rcx
00002D6E 48BF–                             mov rdi, .ascii
00002D70 [C42D000000000000]
00002D78 48B913000000000000–               mov rcx, .asciil
00002D81 00
00002D82 F2AE                              repne scasb          ;Find the offset of the char in al in the
                                                                   table
00002D84 59                                pop rcx              ;Doesnt affect flags
00002D85 4887FB                            xchg rdi, rbx     ;Return value back to rdi
00002D88 75C7                              jne .k1              ;Not a key from our buffer, loop again
00002D8A 48FFC5                            inc rbp
00002D8D E82D000000                        call .print          ;Print typed char

00002D92 488D83(3BD2FFFF)                  lea rax, qword [rbx – .ascii –1]     ;Work out difference

00002D99 AA                                stosb              ;Store the value in storage buffer, inc rdi
00002D9A FECD                              dec ch              ;Decrement the number of typable chars
00002D9C EBB3                              jmp short .k1     ;Get next char
                                     .kend:
00002D9E 66B81112                          mov ax, 1211h     ;Store a space and EOF at the end (little
                                                                   endian!)
00002DA2 66AB                              stosw

```

```
00002DA4 5A                           pop rdx
00002DA5 5F                           pop rdi
                                      ;pop rcx     ;Return in cl the number of processed chars
00002DA6 5B                           pop rbx
00002DA7 58                           pop rax
                                  .kb1:
00002DA8 C3                           ret
                                  .kb2:
                                  ;When a backspace is entered, DONT MOVE THIS PROC!
00002DA9 68[512D0000]                 push .k1
00002DAE 38D5                         cmp ch, dl     ;If bbuf is empty, ignore backspace
00002DB0 74F6                         jz .kb1
00002DB2 48FFCF                       dec rdi         ;Decrement pointer and print the bspace char
00002DB5 FEC5                         inc ch         ;Increment the number of typable chars
00002DB7 4885ED                       test rbp, rbp
00002DBA 7403                         jz .print      ;Dont decrement if rbp is zero
00002DBC 48FFCD                       dec rbp
                                  ;KEYB    ENDP
                                  .print:     ;Print char in al
00002DBF B40E                         mov ah, 0Eh
                                      ;xor bh, bh
00002DC1 CD30                         int 30h
00002DC3 C3                           ret
00002DC4 303132333435363738-      .ascii        db     "0123456789abcdef", 08h, 20h, 0Dh ;b/space, enter
00002DCD 396162636465660820-
00002DD6 0D
                                  .asciil        equ     $ − .ascii
00002DD7 64657367706C777163-      .lst         db     'desgplwqcrbhiovamkx' ;dump, edit, go, single
                                                                  step, read, write, quit,
00002DE0 726268696F76616D6B-
00002DE9 78
                                  ;clearscreen, registers, deBug regs, hex, in, out, version, Single Step
                                                                  alt, memory map
                                  ; (k)connect, dixonnect
                                  .lstl     equ     $ − .lst
00002DEA 0A0D2D00                 .prompt        db     0Ah, 0Dh, "−", 0     ;3Eh = >
00002DEE 2000                     .prompt2     db 20h,0
00002DF0 0A0D00                   .crlf        db     0Ah, 0Dh, 0
                                  ;−−−−−−−−−−−−−−−−−−−−−−−−End of Interrupt−−−−−−−−−−−−−−−−−−−−−−−−−
                                  ;−−−−−−−−−−−−−−−−−−−−Restart Interrupt Int 39h−−−−−−−−−−−−−−−−−−−−
                                  ;This interrupt allows the user to soft reboot
                                  ;−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
                                  bootstrapInt:
                                  ;Bootstrap loader, loads sector 88 of device 0 to 7C00h and jumps
                                                                  to it
                                  ;If not found, will restart the machine, failing that, iretq with
                                                                  CF set
00002DF3 50                           push rax
00002DF4 53                           push rbx
00002DF5 51                           push rcx
00002DF6 52                           push rdx
00002DF7 56                           push rsi

00002DF8 B9000100C0                   mov ecx, 0C0000100h     ;Select fs register to load base addr
00002DFD 488B0425[CD010000]           mov rax, qword [userBase]      ;Load address to fs
00002E05 31D2                         xor edx, edx         ;Zero upper bytes
00002E07 0F30                         wrmsr                    ;Write msr to load fs base

00002E09 BE0A000000                   mov esi, 10
                                  ;Now load one sector of second prog from first device
                                  .e0:
00002E0E 6631D2                       xor dx, dx   ;This also clears carry flag so no checking ah
00002E11 48BB007C0000000000-          mov rbx, 7c00h
00002E1A 00
00002E1B 488B0C25[BF010000]           mov rcx, qword [nextFilePtr]
00002E23 668B0425[C7010000]           mov ax, word [numSectors]
00002E2B B482                         mov ah, 82h ;LBA Sector Read
00002E2D CD33                         int 33h       ;Read one sector
00002E2F 730C                         jnc .e1

00002E31 FFCE                         dec esi
00002E33 742F                         jz .efail

00002E35 30D2                         xor dl, dl
00002E37 30E4                         xor ah, ah   ;Reset the device
00002E39 CD33                         int 33h
00002E3B EBD1                         jmp short .e0
                                  .e1:
00002E3D 31D2                         xor edx, edx   ;Device number 0!
00002E3F 66813C25007C000055-          cmp word [7c00h], 0AA55h ;The Boot signature
00002E48 AA
00002E49 7519                         jne .efail
                                  ;State when system transferred:
                                  ; RSP = DFF8h, 1FFh qword stack from DFFFh to 7C00H + 42*200h sectors = D000h
                                  ; FS MSR = userbase pointer, can be used for segment override.
                                  ; DX = Int 33h boot device number
                                  ; RBX = LBA of first Logical Block after SCP/BIOS
                                  ; BDA and BIOS ready to go
00002E4B 48BCF8DF0000000000-            mov rsp, 0DFF8h ;Move Stack pointer to default init stack position
00002E54 00
00002E55 31D2                           xor edx, edx    ;Device boot number
00002E57 488B1C25[BF010000]             mov rbx, qword [nextFilePtr]    ;First sector on device after SCP/BIOS
00002E5F E9(027C0000)                   jmp 7C02h       ;New sector entry point
                                  .efail:
00002E64 5E                             pop rsi
00002E65 5A                             pop rdx
00002E66 59                             pop rcx
00002E67 5B                             pop rbx
00002E68 58                             pop rax
00002E69 804C241001                     or byte [rsp + 2*8h], 1 ;Set carry flag
00002E6E 48CF                           iretq
                                  ;------------------------------End of Interrupt------------------------------
                                  ;----------------------System Timer Interrupt Int 3Ah----------------------
                                  ;System Timer functions:
                                  ; ah=0 -> Get tick count
                                  ; ah=1 -> Set tick count
                                  ; ah=2 -> Read RTC time
                                  ; ah=3 -> Set RTC time
                                  ; ah=4 -> Read RTC date
                                  ; ah=5 -> Set RTC date
                                  ; ah=6 -> Set RTC alarm
                                  ; ah=7 -> Reset RTC alarm
                                  ; ah=80h -> Get PIT divisor
                                  ; ah=81h -> Set PIT divisor
                                  ;----------------------------------------------------------------------------
                                  timerInt:
00002E70 F6C480                         test ah, 80h
00002E73 747B                           jz .tiext
00002E75 84E4                           test ah, ah
00002E77 7444                           jz .gett
00002E79 80FC01                         cmp ah, 1
00002E7C 745E                           jz .sett
00002E7E 80FC02                         cmp ah, 2
00002E81 0F8497000000                   jz .readRTCtime
00002E87 80FC03                         cmp ah, 3
00002E8A 0F84D8000000                   jz .setRTCtime
00002E90 80FC04                         cmp ah, 4
00002E93 0F8431010000                   jz .readRTCdate
00002E99 80FC05                         cmp ah, 5
00002E9C 0F845C010000                   jz .setRTCdate
00002EA2 80FC06                         cmp ah, 6
00002EA5 0F84B4010000                   jz .setRTCalarm
00002EAB 80FC07                         cmp ah, 7
00002EAE 0F84F6010000                   jz .resetRTCalarm
                                  .bad:
00002EB4 804C241001                     or byte [rsp + 2*8h], 1     ;Set Carry flag on for invalid function
00002EB9 B480                           mov ah, 80h
                                  .exit:
00002EBB 48CF                           iretq
                                  .gett:
                                  ;Returns:
                                  ; al=Rolled over flag (0=not rolled)
                                  ; cx=Hi count
                                  ; dx=Lo count
00002EBD 8B0425[37010000]               mov eax, dword [pit_ticks]
00002EC4 6689C2                         mov dx, ax      ;Lo count
00002EC7 C1E810                         shr eax, 10h    ;Bring high word down
00002ECA 30ED                           xor ch, ch
00002ECC 88C1                           mov cl, al
00002ECE 88E0                           mov al, ah
00002ED0 0FB6C0                         movzx eax, al   ;Zero upper bytes
00002ED3 882425[3A010000]               mov byte [pit_ticks + 3], ah    ;Move 0 into day OF counter
00002EDA 48CF                           iretq
                                  .sett:
                                  ;Called with:
                                  ; cx=Hi count (bzw. cl)
                                  ; dx=Lo count
                                  ;Returns: Nothing
00002EDC 66891425[37010000]             mov word [pit_ticks], dx
00002EE4 30ED                           xor ch, ch      ;Reset the OF counter
00002EE6 66890C25[39010000]             mov word [pit_ticks + 2], cx
00002EEE 48CF                           iretq
```

```
                                       .tiext:     ;Extended Timer functions
00002EF0 80EC80                            sub ah, 80h
00002EF3 7406                              jz .getpitdiv
00002EF5 FECC                              dec ah
00002EF7 740C                              jz .setpitdiv
00002EF9 EBB9                              jmp short .bad
                                       .getpitdiv:
                                       ;Returns:
                                       ; ax=PIT divisor
00002EFB 668B0425[35010000]                mov ax, word [pit_divisor]
00002F03 48CF                              iretq
                                       .setpitdiv:
                                       ;Called with:
                                       ; dx=divsor
                                       ;Returns: Nothing
00002F05 66891425[35010000]                mov word [pit_divisor], dx
00002F0D 50                                push rax
00002F0E B036                              mov al, 36h ;Bitmap for frequency write to channel 0 of PIT
00002F10 E643                              out PITcommand, al
00002F12 6689D0                            mov ax, dx
00002F15 E640                              out PIT0, al     ;Send low byte of new divisor
00002F17 88E0                              mov al, ah
00002F19 E640                              out PIT0, al     ;Send high byte of new divisor
00002F1B 58                                pop rax
00002F1C 48CF                              iretq

                                       .readRTCtime:
                                       ; dh = Seconds
                                       ; cl = Minutes
                                       ; ch = Hours
                                       ; dl = Daylight Savings
00002F1E 50                                push rax
00002F1F 51                                push rcx
00002F20 31C9                              xor ecx, ecx     ;Long counter
                                       .rrt0:
00002F22 FFC9                              dec ecx
00002F24 743C                              jz .rrtbad
00002F26 B08A                              mov al, 8Ah ;Disable NMI and and read bit 7. When 0, read
00002F28 E89A010000                        call .readRTC
00002F2D A880                              test al, 80h     ;Check bit 7 is zero
00002F2F 75F1                              jnz .rrt0    ;If zero, fall and read RTC registers

00002F31 59                                pop rcx          ;Pop upper word of ecx back
00002F32 B080                              mov al, 80h      ;Get seconds
00002F34 E88E010000                        call .readRTC
00002F39 88C6                              mov dh, al       ;Pack seconds in dh
00002F3B B082                              mov al, 82h      ;Get minutes
00002F3D E885010000                        call .readRTC
00002F42 88C1                              mov cl, al       ;Pack minutes in cl
00002F44 B084                              mov al, 84h      ;Get Hours
00002F46 E87C010000                        call .readRTC
00002F4B 88C5                              mov ch, al       ;Pack Hours in ch
00002F4D B08B                              mov al, 8Bh      ;Get Status B for Daylight Savings
00002F4F E873010000                        call .readRTC
00002F54 2401                              and al, 1        ;Isolate bit 0
00002F56 88C2                              mov dl, al       ;Pack Daylight Savings bit in dl
00002F58 B00D                              mov al, 0Dh      ;Enable NMI
00002F5A E868010000                        call .readRTC
00002F5F 58                                pop rax
00002F60 48CF                              iretq
                                       .rrtbad:
00002F62 59                                pop rcx
00002F63 58                                pop rax
00002F64 F9                                stc
00002F65 C20800                            ret 8    ;Set carry and return

                                       .setRTCtime:
                                       ; dh = Seconds
                                       ; cl = Minutes
                                       ; ch = Hours
                                       ; dl = Daylight Savings
00002F68 50                                push rax
00002F69 51                                push rcx
00002F6A 31C9                              xor ecx, ecx
                                       .srt0:
00002F6C FFC9                              dec ecx
00002F6E 74F2                              jz .rrtbad
00002F70 B08A                              mov al, 8Ah ;Disable NMI and and read bit 7. When 0, write
00002F72 E850010000                        call .readRTC
00002F77 A880                              test al, 80h     ;Check bit 7 is zero
00002F79 75F1                              jnz .srt0    ;If zero, fall and write RTC registers
```

```
00002F7B 59                 pop rcx
00002F7C B08B               mov al, 8Bh
00002F7E E844010000         call .readRTC
00002F83 80E201             and dl, 1    ;Ensure we only have the low bit of dl
00002F86 08D0               or al, dl    ;Set the daylight savings bit of Status B
00002F88 0C80               or al, 80h   ;Stop RTC updates
00002F8A 88C4               mov ah, al
00002F8C B08B               mov al, 8Bh ;Reset Status B Register, and daylight savings
00002F8E E83D010000         call .writeRTC

00002F93 88F4               mov ah, dh   ;Pack seconds
00002F95 B080               mov al, 80h
00002F97 E834010000         call .writeRTC
00002F9C 88CC               mov ah, cl   ;Pack minutes
00002F9E B082               mov al, 82h
00002FA0 E82B010000         call .writeRTC
00002FA5 88EC               mov ah, ch   ;Pack hours
00002FA7 B084               mov al, 84h
00002FA9 E822010000         call .writeRTC

00002FAE B08B               mov al, 8Bh
00002FB0 E812010000         call .readRTC
00002FB5 247F               and al, 7Fh ;Clear the top bit
00002FB7 88C4               mov ah, al   ;Pack byte to send in ah
00002FB9 B08B               mov al, 8Bh
00002FBB E810010000         call .writeRTC   ;Restart RTC

00002FC0 B00D               mov al, 0Dh    ;Enable NMI
00002FC2 E800010000         call .readRTC

00002FC7 58                 pop rax
00002FC8 48CF               iretq

                        .readRTCdate:
                        ; ch = Reserved, Century (19/20/21...), fixed 20h for now
                        ; cl = Year
                        ; dh = Month
                        ; dl = Day
00002FCA 50                 push rax
00002FCB 51                 push rcx
00002FCC 31C9               xor ecx, ecx
                        .rrd0:
00002FCE FFC9               dec ecx
00002FD0 7490               jz .rrtbad
00002FD2 B08A               mov al, 8Ah      ;Disable NMI and and read bit 7. When 0, write
00002FD4 E8EE000000         call .readRTC
00002FD9 A880               test al, 80h     ;Check bit 7 is zero
00002FDB 75F1               jnz .rrd0        ;If zero, fall and read RTC registers

00002FDD 59                 pop rcx
00002FDE B087               mov al, 87h      ;Get Day of the Month
00002FE0 E8E2000000         call .readRTC
00002FE5 88C2               mov dl, al       ;Pack Day of the Month
00002FE7 B088               mov al, 88h      ;Get Month of the Year
00002FE9 E8D9000000         call .readRTC
00002FEE 88C6               mov dh, al       ;Pack Month of the Year
00002FF0 B089               mov al, 89h      ;Get bottom two digits of year
00002FF2 E8D0000000         call .readRTC
00002FF7 88C1               mov cl, al       ;Pack Year
00002FF9 B514               mov ch, 20       ;BCD value for 20

00002FFB 58                 pop rax
00002FFC 48CF               iretq

                        .setRTCdate:
                        ; ch = Reserved, Century (19/20/21...), fixed 20h for now
                        ; cl = Year
                        ; dh = Month
                        ; dl = Day
00002FFE 50                 push rax
00002FFF 51                 push rcx
00003000 31C9               xor ecx, ecx
                        .srd0:
00003002 FFC9               dec ecx
00003004 0F8458FFFFFF       jz .rrtbad
0000300A B08A               mov al, 8Ah      ;Disable NMI and and read bit 7. When 0, write
0000300C E8B6000000         call .readRTC
00003011 A880               test al, 80h     ;Check bit 7 is zero
00003013 75ED               jnz .srd0        ;If zero, fall and write RTC registers

00003015 59                 pop rcx
00003016 B08B               mov al, 8Bh
00003018 E8AA000000         call .readRTC
```

```
0000301D 0C80                   or al, 80h          ;Stop RTC updates
0000301F 88C4                   mov ah, al
00003021 B08B                   mov al, 8Bh
00003023 E8A8000000             call .writeRTC
00003028 88D4                   mov ah, dl          ;Pack Day of the Month
0000302A B087                   mov al, 87h
0000302C E89F000000             call .writeRTC
00003031 88F4                   mov ah, dh          ;Pack Month of the Year
00003033 B088                   mov al, 88h
00003035 E896000000             call .writeRTC
0000303A 88CC                   mov ah, cl          ;Pack Year
0000303C B089                   mov al, 89h
0000303E E88D000000             call .writeRTC

00003043 B08B                   mov al, 8Bh
00003045 E87D000000             call .readRTC
0000304A 247F                   and al, 7Fh ;Clear the top bit
0000304C 88C4                   mov ah, al  ;Pack byte to send in ah
0000304E B08B                   mov al, 8Bh
00003050 E87B000000             call .writeRTC   ;Restart RTC

00003055 B00D                   mov al, 0Dh    ;Enable NMI
00003057 E86B000000             call .readRTC

0000305C 58                     pop rax
0000305D 48CF                   iretq

                            .setRTCalarm:
                            ; dh = Seconds for alarm
                            ; cl = Minutes for alarm
                            ; ch = Hours for alarm
0000305F 50                     push rax
00003060 B08B                   mov al, 8BH ;Get status B
00003062 E860000000             call .readRTC
00003067 A820                   test al, 20h
00003069 7537                   jnz .srabad ;If The alarm bit is already set, exit CF=CY

0000306B 88F4                   mov ah, dh          ;Pack Seconds for alarm
0000306D B081                   mov al, 81h
0000306F E85C000000             call .writeRTC
00003074 88CC                   mov ah, cl          ;Pack Minutes for alarm
00003076 B083                   mov al, 83h
00003078 E853000000             call .writeRTC
0000307D 88EC                   mov ah, ch          ;Pack Hours for alarm
0000307F B085                   mov al, 85h
00003081 E84A000000             call .writeRTC

00003086 B08B                   mov al, 8Bh         ;Get Status B
00003088 E83A000000             call .readRTC
0000308D 0C20                   or al, 20h          ;Set Bit 5 - Alarm Interrupt Enable
0000308F 88C4                   mov ah, al          ;Pack new Status B
00003091 B08B                   mov al, 8Bh
00003093 E838000000             call .writeRTC

00003098 B00D                   mov al, 0Dh         ;Enable NMI
0000309A E828000000             call .readRTC

0000309F 58                     pop rax
000030A0 48CF                   iretq
                            .srabad:
000030A2 58                     pop rax
000030A3 804C241001             or byte [rsp + 2*8], 1  ;Set Carry Flag
000030A8 48CF                   iretq
                            .resetRTCalarm:
000030AA 50                     push rax
000030AB B08B                   mov al, 8Bh         ;Get Status B
000030AD E815000000             call .readRTC
000030B2 24DF                   and al, 0DFh        ;Clear Alarm Interrupt Enable
000030B4 88C4                   mov ah, al
000030B6 B08B                   mov al, 8Bh
000030B8 E813000000             call .writeRTC

000030BD B00D                   mov al, 0Dh         ;Enable NMI
000030BF E803000000             call .readRTC
000030C4 58                     pop rax
000030C5 48CF                   iretq

                            .readRTC:
                            ;Reads an RTC port, interrupts disabled throughout
                            ;Input: al = I/O port to read
                            ;Output: al = I/O data
000030C7 FA                     cli
000030C8 E670                   out cmos_base, al
```

```
000030CA E680                      out waitp, al
000030CC E471                      in al, cmos_data
000030CE FB                        sti
000030CF C3                        ret
                               .writeRTC:
                               ;Writes to an RTC port, interrupts disabled throughout
                               ;Input: al = I/O port to read, ah = Data byte to send
000030D0 FA                        cli
000030D1 E670                      out cmos_base, al
000030D3 E680                      out waitp, al
000030D5 88E0                      mov al, ah
000030D7 E671                      out cmos_data, al
000030D9 FB                        sti
000030DA C3                        ret
                               ;----------------------------End of Interrupt----------------------------
                               ;---------------------CTRL+BREAK Interrupt Int 3Bh---------------------
                               ;CTRL+Break will call this!
                               ;------------------------------------------------------------------------
                               ctrlbreak_io:
000030DB 48CF                      iretq
                               ;----------------------------End of Interrupt----------------------------
                               ;--------------Screen Mode Parameters Interrupt Int 3Dh--------------
                               ;This Interrupt returns in r8 the pointer to screen mode
                               ; parameters. It replaces the nice pointers in the IVT of yore.
                               ;Returns in r8 to not conflict with ported apps
                               ;------------------------------------------------------------------------
                               scr_params_io:
000030DD 49B8-                     mov r8, scr_mode_params
000030DF [6001000000000000]
000030E7 48CF                      iretq
                               ;----------------------------End of Interrupt----------------------------
                               ;---------------------Disk Params Interrupt Int 3Eh---------------------
                               disk_params_io:
000030E9 4C8B0425[AF010000]        mov r8, qword [diskDptPtr]
000030F1 4C8B0C25[B7010000]        mov r9, qword [fdiskDptPtr]
000030F9 48CF                      iretq
                               ;----------------------------End of Interrupt----------------------------
                               ;----------------------CGA font Interrupt Int 3Fh----------------------
                               ;This Interrupt returns in r8 the pointer to the CGA font.
                               ;It replaces the nice pointers in the IVT of yore.
                               ;Returns in r8 to not conflict with ported apps
                               ;------------------------------------------------------------------------
                               cga_ret_io: ;Get first pointer in list
000030FB 4C0FB70425-               movzx r8, word [scr_vga_ptrs]
00003100 [68010000]
00003104 49C1E004                  shl r8, 4
00003108 6644030425-               add r8w, word [scr_vga_ptrs + 2]
0000310D [6A010000]
00003111 48CF                      iretq
                               ;----------------------------End of Interrupt----------------------------
                               ;------------------------IDE Driver and data area------------------------
                               IDE:
                               .addControllerTable:
                               ;Adds a PCI IDE controller to the internal data tables, if there is space
                               ; If there is no space, returns with carry set.
                               ;Input: eax = BAR5 address
                               ;       ebx = PCI IO address
                               ;Output: CF=NC, all ok, CF=CY, device not added.
00003113 56                        push rsi
00003114 803C25[5A030000]02        cmp byte [ideNumberOfControllers], 2
0000311C 7428                      je .actfail ;If it is 2, fail
0000311E FE0425[5A030000]          inc byte [ideNumberOfControllers]
00003125 48BE-                     mov rsi, ideControllerTable
00003127 [5B03000000000000]
0000312F 803E00                    cmp byte [rsi], 0    ;Is the first entry empty?
00003132 7407                      jz .act0      ;If yes, write entry
00003134 4881C610000000            add rsi, ideTableEntrySize  ;Else, goto second entry space
                               .act0:
0000313B 891E                      mov dword [rsi], ebx      ;Move first PCI IO addr
0000313D C60600                    mov byte [rsi], 0         ;Zero the register index
00003140 894604                    mov dword [rsi + 4], eax      ;Move next data
00003143 F8                        clc
                               .actexit:
00003144 5E                        pop rsi
00003145 C3                        ret
                               .actfail:
00003146 F9                        stc
00003147 EBFB                      jmp short .actexit
                               .identifyDevice:
                               ;dx should contain the base register
                               ;al should contain either A0/B0 for master/slave
                               ;rdi points to the buffer
```

```
                                ;Carry set if failed.
00003149 50                         push rax            ;save the master/slave bit temporarily
0000314A 6681C20700                 add dx, 7           ;dx at base + 7
                                .l1:
0000314F EC                         in al, dx
00003150 3CFF                       cmp al, 0FFh
00003152 7447                       je .exitfail
00003154 A880                       test al, 10000000b
00003156 75F7                       jnz .l1

00003158 EB00                       jmp short $ + 2             ;IO cycle kill
0000315A FA                         cli
                                .l2:
0000315B EC                         in al, dx
0000315C 84C0                       test al, al
0000315E 743B                       jz .exitfail
00003160 A840                       test al, 01000000b
00003162 74F7                       jz .l2

00003164 30C0                       xor al, al
00003166 6681EA0500                 sub dx, 5           ;dx at base + 2
0000316B EE                         out dx, al
0000316C 66FFC2                     inc dx              ;dx at base + 3
0000316F EE                         out dx, al
00003170 66FFC2                     inc dx              ;dx at base + 4
00003173 EE                         out dx, al
00003174 66FFC2                     inc dx              ;dx at base + 5
00003177 EE                         out dx, al
00003178 66FFC2                     inc dx              ;dx at base + 6
0000317B 58                         pop rax             ;Get the master/slave bit back
0000317C EE                         out dx, al
0000317D 66FFC2                     inc dx              ;dx at base + 7
00003180 B0EC                       mov al, 0ECh        ;ECh = Identify drive command
00003182 EE                         out dx, al

00003183 EB00                       jmp short $ + 2     ;IO cycle kill
                                .l3:
00003185 EC                         in al, dx           ;get status byte
00003186 A808                       test al, 00001000b  ;Check DRQ, to be set for data ready
00003188 74FB                       jz .l3

0000318A 6681EA0700                 sub dx, 7           ;dx at base + 0
0000318F 51                         push rcx
00003190 66B90001                   mov cx, 100h        ;100h words to be copied
00003194 F3666D                     rep insw
00003197 F8                         clc
00003198 FB                         sti
00003199 EB01                       jmp short .exit


                                .exitfail:
0000319B F9                         stc
                                .exit:
0000319C 58                         pop rax
0000319D C3                         ret
                                ;————————————————USB Driver and data area————————————————
                                USB:
                                ;————————————————————EHCI functions————————————————————
                                ;eActiveCtrlr must be set with the offset of the controller
                                ; IFF the controller is about to enter a state in which it could
                                ; fire an interrupt. These functions must safeguard against it by
                                ; checking that this byte is −1 first and then setting the byte
                                ; with the selected controller index, ending by resetting this
                                ; byte to −1 (even on fail).
                                ;
                                ;Certain functions may be called to act upon the CURRENT ACTIVE
                                ; controller, these functions dont need these safeguards, though
                                ; they may need to ensure that there is a valid controller number
                                ; in the eActiveCtrlr byte.
                                ;————————————————————————————————————————————————————————
                                .ehciCriticalErrorWrapper:
                                ;Currently just jumps to the installed address.
                                ;Conditional error calls MUST call this wrapper to allow for
                                ; host operating systems to install their own USB error handlers
                                ; and have the system continue working.
0000319E FF2425[36020000]           jmp qword [eHCErrorHandler]
                                .ehciCriticalErrorHandler:
                                ;Currently just halts the system
000031A5 BB07000000                 mov ebx, 07h
000031AA E848CFFFFF                 call cls
000031AF 48BD-                      mov rbp, .ecehmsg
000031B1 [CC31000000000000]
000031B9 66B80413                   mov ax, 1304h
000031BD CD30                       int 30h
```

```
000031BF B0FF                                  mov al, 0FFh
000031C1 E621                                  out pic1data, al
000031C3 E6A1                                  out pic2data, al
000031C5 FA                                    cli
000031C6 F4                                    hlt
000031C7 E9F9FFFFFF                            jmp $ - 2
000031CC 454843492043686563-               .ecehmsg db ”EHCI Check 1”, 0
000031D5 6B203100

                                           .setupEHCIcontroller:
                                           ;Resets, initialises variables to default
                                           ;Input: al = Controller to setup (0 based)
                                           ;Output: CF=CY - Controller failed to reset
                                           ;         CF=NC - No problems
                                           ; al = Controller that was reset
000031D9 51                                    push rcx
000031DA 53                                    push rbx
000031DB 55                                    push rbp
000031DC E80F010000                            call .ehciResetCtrlr     ;Reset the controller
000031E1 7215                                  jc .secexit
000031E3 6631DB                                xor bx, bx ;No schedule, no interrupts
000031E6 31C9                                  xor ecx, ecx
000031E8 48BD-                                 mov rbp, ehciAschedule
000031EA [0000000000000000]

000031F2 E8EB010000                            call .ehciInitCtrlrRegs     ;Initialise controller registers
000031F7 F8                                    clc
                                           .secexit:
000031F8 5D                                    pop rbp
000031F9 5B                                    pop rbx
000031FA 59                                    pop rcx
000031FB C3                                    ret

                                           .ehciResetControllerPort:
                                           ;A function that enacts an EHCI reset on a port.
                                           ;Works ONLY on the current active controller.
                                           ;Input:
                                           ; al = Port number [0,N-1] (Checked against ctrlr struc params
                                                                       entry)
                                           ;Returns:
                                           ; CF set if failed, clear if success
                                           ; ax=Error code, 0h=No active controller
                                           ;               1h=Invalid port number
                                           ;               2h=No device on port
                                           ;               3h=Port not enabled (Low speed device)
                                           ;               4h=Device not entering reset
                                           ;               5h=Device not clearing reset
                                           ;               6h=Port not enabled (Full speed device)
                                           ; rax destroyed
000031FC 53                                    push rbx
000031FD 51                                    push rcx
000031FE 52                                    push rdx
000031FF 55                                    push rbp

00003200 6631ED                                xor bp, bp
00003203 0FB6D0                                movzx edx, al     ;Save port number into dl (edx)
00003206 0FB61C25[47020000]                    movzx ebx, byte [eActiveCtrlr]
0000320E 80FBFF                                cmp bl, -1
00003211 0F84D3000000                          je .ercperr     ;Error, No active controller (ec=0)
00003217 66FFC5                                inc bp          ;Inc error counter
0000321A 8B1CDD[19020000]                      mov ebx, dword [eControllerList + 4 + 8*rbx]      ;get mmiobase
                                                                       into ebx
00003221 678B4304                              mov eax, dword [ebx+ehcistrucparams]      ;Get # of ports in al
00003225 247F                                  and al, 7Fh     ;al contains port number, clear upper bit
00003227 FEC8                                  dec al          ;Zero based port number
00003229 0FB6C0                                movzx eax, al
0000322C 38C2                                  cmp dl, al      ;dl contains called port number
0000322E 0F87B6000000                          ja .ercperr     ;Error, invalid port number (ec=1)
00003234 66FFC5                                inc bp          ;Inc error counter


00003237 670FB603                              movzx eax, byte [ebx]      ;Byte access for caplength!
0000323B 01C3                                  add ebx, eax      ;eax now points to opregs
0000323D 66B90A00                              mov cx, 10
                                           .erclp0:      ;Remember ebx=opregs, edx=port number
00003241 67814C934400100000                    or dword [ebx+4*edx+ehciportsc], 1000h ;Set power bit

0000324A 51                                    push rcx
0000324B B90A000000                            mov ecx, 10
00003250 B486                                  mov ah, 86h
00003252 CD35                                  int 35h          ;Wait for 10 ms
00003254 59                                    pop rcx

                                           .erclp1:
00003255 66FFC9                                dec cx
```

```
00003258 0F848C000000               jz .ercperr ;Error, No device on port (ec=2)
0000325E 67F744934401000000         test dword [ebx+4*edx+ehciportsc], 1h     ;Test device on port
00003267 74D8                       jz .erclp0
00003269 66FFC5                     inc bp          ;Inc error counter

0000326C 678B449344                 mov eax, dword [ebx+4*edx+ehciportsc]
00003271 6625000C                   and ax, 0C00h
00003275 662D0004                   sub ax, 400h
00003279 66FFC8                     dec ax
0000327C 746C                       jz .ercperr     ;Error, Low speed device (ec=3)
0000327E 66FFC5                     inc bp          ;Inc error counter

00003281 66B90A00                   mov cx, 10
                                .erclp2:
00003285 66FFC9                     dec cx
00003288 7460                       jz .ercperr ;Error, Device not entering reset (ec=4)
0000328A 67814C934400010000         or dword [ebx+4*edx+ehciportsc], 100h     ;Set bit 8, port reset bit

00003293 51                         push rcx
00003294 B90A000000                 mov ecx, 10
00003299 B486                       mov ah, 86h
0000329B CD35                       int 35h          ;Wait for 10 ms
0000329D 59                         pop rcx

0000329E 67F744934400010000         test dword [ebx+4*edx+ehciportsc], 100h     ;Check if entered reset
000032A7 74DC                       jz .erclp2

000032A9 66FFC5                     inc bp          ;Inc error counter
000032AC 66B90A00                   mov cx, 10
000032B0 6781649344FFFEFFFF         and dword [ebx+4*edx+ehciportsc], 0FFFFFEFFh     ;Clear reset bit
                                .erclp3:
000032B9 FFC9                       dec ecx
000032BB 742D                       jz .ercperr ;Error, Device not leaving reset (ec=5)

000032BD 51                         push rcx
000032BE B90A000000                 mov ecx, 10
000032C3 B486                       mov ah, 86h
000032C5 CD35                       int 35h          ;Wait for 10 ms
000032C7 59                         pop rcx

000032C8 67F744934400010000         test dword [ebx+4*edx+ehciportsc], 100h
000032D1 75E6                       jnz .erclp3
000032D3 66FFC5                     inc bp          ;Inc error counter

000032D6 67F744934404000000         test dword [ebx+4*edx+ehciportsc], 4h     ;Bit 2 is the port enabled bit
000032DF 7409                       jz .ercperr     ;Error, Full speed device (ec=6)
                                ;We get here IFF device on port is high speed


                                ;High Speed Device successfully reset. Now print message or whatever
000032E1 4831C0                     xor rax, rax
000032E4 F8                         clc
                                .ercpexit:
000032E5 5D                         pop rbp
000032E6 5A                         pop rdx
000032E7 59                         pop rcx
000032E8 5B                         pop rbx
000032E9 C3                         ret
                                .ercperr:
000032EA 6689E8                     mov ax, bp     ;Get error code in ax
000032ED F9                         stc
000032EE EBF5                       jmp short .ercpexit

                                .ehciResetCtrlr:
                                ;A function that resets a controller.
                                ;No other controllers may be running during a ctrlr reset
                                ;Input:
                                ; al = Offset into the ehci controller table
                                ;Returns:
                                ; CF=CY if failed, CF=NC if reset
                                ;All registers preserved
000032F0 50                         push rax
000032F1 51                         push rcx
                                    ;cmp byte [eActiveCtrlr], -1
                                    ;jne .erc2     ;A controller already active, exit fail (ec=0)
                                    ;mov byte [eActiveCtrlr], al     ;For added security (may be removed later)
000032F2 E800180000                 call .ehciGetOpBase
000032F7 67C7400800000000           mov dword [eax + ehciintr], 0h     ;No interrupts
000032FF 67C740043F000000           mov dword [eax + ehcists], 3Fh     ;Clear any outstanding interrupts
```

```
                                  ;Set the reset bit, check to see if run bit has cleared first!
00003307 31C9                     xor ecx, ecx
                              .erc0:
00003309 678120FEFFFFFF           and dword [eax + ehcicmd], 0FFFFFFFEh     ;Force stop the controller
00003310 FFC9                     dec ecx
00003312 743D                     jz .erc2      ;Controller not resetting, exit fail  (ec=1)

00003314 67F7400400100000         test dword [eax + ehcists], 1000h     ;Test if bit 12 has been set
0000331C 74EB                     jz .erc0
0000331E 67810802000000           or dword [eax + ehcicmd], 02h ;Set bit 1, reset HC
                                  ;Spin and wait to give device time to respond and reset.
00003325 6631C9                   xor cx, cx
                              .erc1:
00003328 66FFC9                   dec cx          ;Wait for reset to happen
0000332B 7424                     jz .erc2      ;Not resetting, exit fail (ec=2)

0000332D 50                       push rax
0000332E 51                       push rcx
0000332F B486                     mov ah, 86h
00003331 B905000000               mov ecx, 5      ;5ms wait
00003336 CD35                     int 35h
00003338 59                       pop rcx
00003339 58                       pop rax

0000333A 67F70002000000           test dword [eax + ehcicmd], 2h      ;Whilst this bit is set, keep looping
00003341 75E5                     jnz .erc1
00003343 31C0                     xor eax, eax
00003345 F8                       clc
                              .ercexit:
00003346 C60425[47020000]FF       mov byte [eActiveCtrlr], -1     ;No controllers active
0000334E 59                       pop rcx
0000334F 58                       pop rax
00003350 C3                       ret
                              .erc2:
00003351 F9                       stc
00003352 EBF2                     jmp short .ercexit

                              .ehciRunCtrlr:
                              ;A function that runs a controller to process set schedules
                              ;Input:
                              ;    al = Offset into the controller table
                              ;Returns:
                              ; CF = CY if failed, CF = NC if success
00003354 50                       push rax
00003355 51                       push rcx
00003356 E89C170000               call .ehciGetOpBase
0000335B 67F7400400100000         test dword [eax + ehcists], 1000h     ;bit 12 must be set to write 1 in cmd
00003363 741E                     jz .esc2
00003365 67810801000000           or dword [eax + ehcicmd], 1h ;Set bit 0 to run
0000336C 31C9                     xor ecx, ecx
                              .esc0:
0000336E 66FFC9                   dec cx
00003371 7410                     jz .esc2
00003373 67F7400400100000         test dword [eax + ehcists], 1000h     ;bit 12 must be clear
0000337B 75F1                     jnz .esc0
0000337D 31C0                     xor eax, eax
0000337F F8                       clc
                              .esc1:
00003380 59                       pop rcx
00003381 58                       pop rax
00003382 C3                       ret
                              .esc2:     ;Bad exit
00003383 F9                       stc
00003384 EBFA                     jmp short .esc1

                              .ehciStopCtrlr:
                              ;A function that stops current active controller from running
                              ;Input:
                              ; al=Controller to stop processing
                              ;Returns:
                              ; CF set if failed to stop, clear if success
00003386 50                       push rax
00003387 51                       push rcx
00003388 480FB60425-              movzx rax, byte [eActiveCtrlr]
0000338D [47020000]

00003391 E861170000               call .ehciGetOpBase
00003396 678120FEFFFFFF           and dword [eax + ehcicmd], 0FFFFFFFEh     ;Stop controller
0000339D 31C9                     xor ecx, ecx
                              .estc0:
```

```
0000339F 66FFC9                     dec cx
000033A2 740E                       jz .estc1
000033A4 67F7400400100000           test dword [eax + ehcists], 1000h     ;test hchalted until set
000033AC 74F1                       jz .estc0
000033AE F8                         clc
                                .estcexit:
000033AF 59                         pop rcx
000033B0 58                         pop rax
000033B1 C3                         ret
                                .estc1:
000033B2 F9                         stc
000033B3 EBFA                       jmp short .estcexit
                                .ehciAdjustAsyncSchedCtrlr:
                                ;This function checks the currently online controller and compares it to
                                ; the value provided in al.
                                ;If they are equal, do nothing.
                                ;If not, turn off controller, update active ctrlr byte and indicate a new bus
                                ; was activated.
                                ;If no controller active, update active ctrlr byte and indicate which bus
                                ; has been activated.
                                ;
                                ; Input: al = Controller to activate, preserved.
                                ; Output: CF=CY: Error, turn off all controllers
                                ;         CF=NC: All ok, proceed
000033B5 3A0425[47020000]           cmp al, byte [eActiveCtrlr]
000033BC 7420                       je .eacOkExit
000033BE 803C25[47020000]FF         cmp byte [eActiveCtrlr], -1
000033C6 7407                       je .eacStart
000033C8 E8D8020000                 call .ehciStopAsyncSchedule ;Stop currently transacting controller
000033CD 7211                       jc .eacBad
                                .eacStart:
000033CF 880425[47020000]           mov byte [eActiveCtrlr], al ;Set new active controller
000033D6 C60425[46020000]01         mov byte [eNewBus], 1    ;Set flag that a new bus has been selected
                                .eacOkExit:
000033DE F8                         clc
000033DF C3                         ret
                                .eacBad:
000033E0 F9                         stc
000033E1 C3                         ret
                                .ehciInitCtrlrRegs:
                                ;A function that initialises a given controllers registers as needed.
                                ;Controller is left ready to process data start schedules
                                ;MUST NOT BE CALLED ON A RUNNING CONTROLLER
                                ;Input:
                                ; al = Offset into the ehci controller table
                                ; bl = ehciintr mask
                                ; bh = Schedule mask, bits [7:2] reserved
                                ;         00b = No schedule, 01b=Periodic, 10b=Async, 11b=Both
                                ; ecx = Frame Index
                                ; rbp = Schedule address
                                ;Returns:
                                ; Nothing
000033E2 50                         push rax
000033E3 53                         push rbx
000033E4 51                         push rcx
000033E5 53                         push rbx
000033E6 E80C170000                 call .ehciGetOpBase     ;Get opbase
000033EB 0FB7DB                     movzx ebx, bx
000033EE 67C7400800000000           mov dword [eax + ehciintr], 0
000033F6 6789480C                   mov dword [eax + ehcifrindex], ecx
000033FA 67896818                   mov dword [eax + ehciasyncaddr], ebp
000033FE 48C1CD20                   ror rbp, 20h     ;Get upper dword low
00003402 67896810                   mov dword [eax + ehcictrlseg], ebp
00003406 5B                         pop rbx     ;Get back bh
00003407 30DB                       xor bl, bl     ;Zero lo byte
00003409 66C1EB04                   shr bx, 4     ;Shift to hi nybble of lo byte
0000340D 678120CF000000             and dword [eax + ehcicmd], 0CFh     ;Clear schedule enable bits
00003414 670B18                     or ebx, dword [eax + ehcicmd]     ;Add ehcicmd to schedule mask
00003417 81E3F3FF00FF               and ebx, 0FF00FFF3h     ;Clear the Int Threshold and Frame List bits
0000341D 81CB00000800               or ebx, 000080000h ;Set 8 microframes (1 ms) per interrupt
00003423 678918                     mov dword [eax + ehcicmd], ebx     ;Write back
00003426 67C7404001000000           mov dword [eax + ehciconfigflag], 1h     ;Route all ports to EHCI ctrlr
0000342E 59                         pop rcx
0000342F 5B                         pop rbx
00003430 58                         pop rax
```

```
00003431 C3                              ret
                                     .ehciCtrlrGetNumberOfPorts:
                                     ;Gets the number of ports on a Host Controller.
                                     ;Ports are zero addressed so ports numbers are 0 to NUMBER_OF_PORTS - 1
                                     ;Input:  al = Offset into the controller table
                                     ;Output: rax = Number of ports on controller.
                                     ;Warning, input NOT bounds checked.
00003432 0FB6C0                          movzx eax, al
00003435 8B04C5[19020000]                mov eax, dword [eControllerList + 4 + 8*rax]
0000343C 678B4004                        mov eax, dword [eax + ehcistrucparams]
00003440 257F000000                      and eax, 7Fh       ;Clear upper bits
00003445 C3                              ret
                                     .ehciGetNewQHeadAddr:
                                     ;Picks which QHead position to put the new Qhead into
                                     ;Input: Nothing
                                     ;Output: rdi = Position in RAM for QHead
                                     ;          r8  = Link to next QHead
                                     ;              r8 NEEDS to be or'ed with 2 when used as a QHead pointer
00003446 49B8-                           mov r8, ehciQHead1
00003448 [8000000000000000]
00003450 48BF-                           mov rdi, ehciQHead0
00003452 [0000000000000000]
0000345A 483B3C25[3E020000]              cmp rdi, qword [eCurrAsyncHead]     ;Compare head to start of buffer
00003462 7503                            jne .egnqaexit
00003464 4987F8                          xchg rdi, r8
                                     .egnqaexit:
00003467 C3                              ret

                                     .ehciToggleTransactingQHead:
                                     ;Toggles the transacting Qhead position
                                     ;This is called AFTER the old Qhead has been delinked from the AsynchSchedule
00003468 48813C25[3E020000]-             cmp qword [eCurrAsyncHead], ehciQHead0
00003470 [00000000]
00003474 750D                            jne .ettqh0
00003476 48C70425[3E020000]-             mov qword [eCurrAsyncHead], ehciQHead1
0000347E [80000000]
00003482 C3                              ret
                                     .ettqh0:
00003483 48C70425[3E020000]-             mov qword [eCurrAsyncHead], ehciQHead0
0000348B [00000000]
0000348F C3                              ret

                                     .ehciDelinkOldQHead:
                                     ;Delinks the old Qhead from the list async list
00003490 57                              push rdi
00003491 4150                            push r8
00003493 E8AEFFFFFF                      call .ehciGetNewQHeadAddr
00003498 4989F8                          mov r8, rdi
0000349B 4981C802000000                  or r8, 2
000034A2 448907                          mov dword [rdi], r8d      ;Point the new qhead to itself
000034A5 814F0400800000                  or dword [rdi + 4], 8000h    ;Toggle H-bit in the current transacting QHead
000034AC 4158                            pop r8
000034AE 5F                              pop rdi
000034AF C3                              ret

                                     .ehciLinkNewQHead:
                                     ;Links the inserted qhead into the async list
000034B0 57                              push rdi
000034B1 4150                            push r8
000034B3 E88EFFFFFF                      call .ehciGetNewQHeadAddr     ;Get bus addresses
000034B8 803C25[46020000]01              cmp byte [eNewBus], 1
000034C0 740F                            je .elnqadjusted     ;If equal, exit
000034C2 4881CF02000000                  or rdi, 2
000034C9 418938                          mov dword [r8], edi
                                     .elnqhexit:
000034CC F8                              clc
000034CD 4158                            pop r8
000034CF 5F                              pop rdi
000034D0 C3                              ret
                                     ;Only here if a new bus was Adjusted
                                     .elnqadjusted:
                                     ;The first qhead in a new queue must always point to itself and be
                                     ; the head of the reclaim list.
                                     ;The same address is provided to the function which writes the qhead
                                     ; and in the above function call into rdi, thus allowing us to point
                                     ; the new qhead to itself and set the H-bit on, in ALL instances
000034D1 4989F8                          mov r8, rdi
000034D4 4981C802000000                  or r8, 2
000034DB 448907                          mov dword [rdi], r8d      ;Point the QHead to itself
```

```
000034DE 814F0400800000               or dword [rdi + 4], 8000h    ;Set H bit on
000034E5 50                           push rax
000034E6 8A0425[47020000]             mov al, byte [eActiveCtrlr]
000034ED E805160000                   call .ehciGetOpBase
000034F2 67897818                     mov dword [eax + ehciasyncaddr], edi ;Set the address in the
                                                                  ctrlr register
000034F6 58                           pop rax
000034F7 E87D010000                   call .ehciStartAsyncSchedule     ;Start schedule
000034FC 7209                         jc .elnqhbad
000034FE FE0C25[46020000]             dec byte [eNewBus]   ;Reset back to zero if successfully onlined
00003505 EBC5                         jmp short .elnqhexit
                                  .elnqhbad:   ;If Async fails to start, exit
00003507 4158                         pop r8
00003509 5F                           pop rdi
0000350A F9                           stc
0000350B C3                           ret

                                  .ehciSetNoData:
                                  ;A function that does a set request with no data phase to the device
                                  ;at address al.
                                  ;Input:
                                  ; al = Address number (7 bit value)
                                  ; rbx = Setup packet
                                  ; cx = Max Packet Length
                                  ;Returns:
                                  ; CF = NC if no Host error, CF = CY if Host error
                                  ; Caller MUST check the schedule to ensure that the transfer was
                                                                  successful,
                                  ; and without transaction errors as these dont constitute Host
                                                                  system errors.
                                  ;
                                  ; All registers except for CF preserved
0000350C 57                           push rdi
0000350D 4150                         push r8
0000350F 4151                         push r9
00003511 4152                         push r10
00003513 4153                         push r11
00003515 51                           push rcx
00003516 52                           push rdx
00003517 FC                           cld     ;Set right direction for string ops

                                      ;Write setup packet
00003518 48891C25[80030000]           mov qword [ehciDataOut], rbx
00003520 E821FFFFFF                   call .ehciGetNewQHeadAddr
00003525 4981C802000000               or r8, 2     ;Process qH TDs
0000352C 41B900600080                 mov r9d, 80006000h   ;Bit 15 not set here!!!!! Important
00003532 0FB7C9                       movzx ecx, cx
00003535 C1E110                       shl ecx, 8*2
00003538 4109C9                       or r9d, ecx
0000353B 247F                         and al, 7Fh     ;Force clear upper bit of al
0000353D 4108C1                       or r9b, al     ;Set lower 8 bits of r9 correctly
00003540 41BA00000040                 mov r10d, 40000000h     ;1 transaction/ms
00003546 49BB-                        mov r11, ehciTDSpace   ;First TD is the head of the buffer
00003548 [0001000000000000]

00003550 E827080000                   call .ehciWriteQHead

00003555 4C89DF                       mov rdi, r11     ;Move pointer to TD buffer head
00003558 4C8D4740                     lea r8, qword [rdi + ehciSizeOfTD]     ;Point to next TD
0000355C 49B901000000000000-          mov r9, 1
00003565 00
00003566 41BA800E0800                 mov r10d, 00080E80h ;Active TD, SETUP EP, Error ctr = 3, 8 byte
                                                                  transfer
0000356C 49BB-                        mov r11, ehciDataOut ; Data out buffer
0000356E [8003000000000000]

00003576 E826080000                   call .ehciWriteQHeadTD

0000357B 4881C740000000               add rdi, ehciSizeOfTD       ;Go to next TD space
00003582 49B801000000000000-          mov r8, 1
0000358B 00
0000358C 4D89C1                       mov r9, r8
0000358F 41BA808D0080                 mov r10d, 80008D80h         ;Status stage opposite direction of
                                                                  last transfer
00003595 49BB-                        mov r11, msdCSW          ;Nothing should be returned but use
                                                                  this point
00003597 [C005000000000000]

0000359F E8FD070000                   call .ehciWriteQHeadTD
000035A4 B103                         mov cl, 011b    ;Lock out internal buffer
000035A6 E9BD000000                   jmp .egddproceed

                                  .ehciGetRequest:
                                  ;A function which does a standard get request from a device at
                                  ;address al.
                                  ;Input:
                                  ; al = Address number (7 bit value)
                                  ; rbx = Setup packet
                                  ; ecx = Max Packet Length
                                  ;Returns:
                                  ; CF = NC if no Host error, CF = CY if Host error
                                  ; Caller MUST check the schedule to ensure that the transfer was
                                                                  successful,
                                  ; and without transaction errors as these dont constitute Host
                                                                  system errors.
                                  ;
                                  ; All registers except for CF preserved
000035AB 57                                push rdi
000035AC 4150                              push r8
000035AE 4151                              push r9
000035B0 4152                              push r10
000035B2 4153                              push r11
000035B4 51                                push rcx
000035B5 52                                push rdx
000035B6 FC                                cld      ;Ensure right direction

                                      ;Write setup packet
000035B7 48891C25[80030000]                mov qword [ehciDataOut], rbx
000035BF E882FEFFFF                        call .ehciGetNewQHeadAddr
000035C4 4981C802000000                    or r8, 2      ;Process qH TDs
000035CB 41B900600080                      mov r9d, 80006000h   ;Bit 15 not set here!!!!! Important
000035D1 0FB7C9                            movzx ecx, cx
000035D4 C1E110                            shl ecx, 8*2
000035D7 4109C9                            or r9d, ecx
000035DA 247F                              and al, 7Fh     ;Force clear upper bit of al
000035DC 4108C1                            or r9b, al      ;Set lower 8 bits of r9 correctly
000035DF 41BA00000040                      mov r10d, 40000000h     ;1 transaction/ms
000035E5 49BB-                             mov r11, ehciTDSpace   ;First TD is the head of the buffer
000035E7 [0001000000000000]

000035EF E888070000                        call .ehciWriteQHead

000035F4 4C89DF                            mov rdi, r11      ;Move pointer to TD buffer head
000035F7 4C8D4740                          lea r8, qword [rdi + ehciSizeOfTD]      ;Point to next TD
000035FB 49B901000000000000-               mov r9, 1
00003604 00
00003605 41BA800E0800                      mov r10d, 00080E80h ;Active TD, SETUP EP, Error ctr = 3, 8 byte
                                                                  transfer
0000360B 49BB-                             mov r11, ehciDataOut ; Data out buffer
0000360D [8003000000000000]

00003615 E887070000                        call .ehciWriteQHeadTD

0000361A 4881C740000000                    add rdi, ehciSizeOfTD      ;Go to next TD space
00003621 4C8D4740                          lea r8, qword [rdi + ehciSizeOfTD]
00003625 4D89C1                            mov r9, r8      ;Alt pointer also points to next TD since this is
                                                                  expected!
00003628 41BA800D4080                      mov r10d, 80400D80h ;Active TD, IN EP, Error ctr = 3, max 64
                                                                  byte transfer
0000362E 49BB-                             mov r11, ehciDataIn
00003630 [C003000000000000]

00003638 E864070000                        call .ehciWriteQHeadTD

0000363D 4881C740000000                    add rdi, ehciSizeOfTD       ;Go to next TD space
00003644 49B801000000000000-               mov r8, 1
0000364D 00
0000364E 4D89C1                            mov r9, r8
00003651 41BA808C0080                      mov r10d, 80008C80h
00003657 49BB-                             mov r11, msdCSW
00003659 [C005000000000000]

00003661 E83B070000                        call .ehciWriteQHeadTD

00003666 B103                              mov cl, 11b      ;Lock out internal buffer, ignore one interrupt
                                  ;Now set controller to process the schedule
                                  .egddproceed:
00003668 E867000000                        call .ehciProcessCommand
                                  ;The carry status of the previous function will propagate
                                  .egddexit:
0000366D 5A                                pop rdx
0000366E 59                                pop rcx
0000366F 415B                              pop r11
00003671 415A                              pop r10
00003673 4159                              pop r9
00003675 4158                              pop r8
```

```
7951 00003677 5F                        pop rdi
7952 00003678 C3                        ret
7953
7954                                .ehciStartAsyncSchedule:
7955 00003679 50                        push rax
7956 0000367A 51                        push rcx
7957
7958 0000367B 8A0425[47020000]          mov al, byte [eActiveCtrlr]     ;Deals with current active controller
7959 00003682 E870140000                call .ehciGetOpBase             ;Return opregs ADDRESS in eax
7960 00003687 67810820000000            or dword [eax + ehcicmd], 20h     ;Process asyncschedule
7961 0000368E 31C9                      xor ecx, ecx
7962                                .esas0:
7963 00003690 FFC9                      dec ecx
7964 00003692 740E                      jz .esasfail
7965 00003694 67F7400400800000          test dword [eax + ehcists], 08000h ;Asyncschedule bit should be on
7966 0000369C 74F2                      jz .esas0
7967
7968 0000369E F8                        clc
7969                                .esasok:
7970 0000369F 59                        pop rcx
7971 000036A0 58                        pop rax
7972 000036A1 C3                        ret
7973                                .esasfail:
7974 000036A2 F9                        stc
7975 000036A3 EBFA                      jmp short .esasok
7976
7977                                .ehciStopAsyncSchedule:
7978                                ;This function stops the processing of the current active Async Schedule
7979                                ;Output: CF=CY: Failed to stop Async Schedule CF=NC: Stopped Async Schedule
7980 000036A5 50                        push rax
7981 000036A6 51                        push rcx
7982 000036A7 8A0425[47020000]          mov al, byte [eActiveCtrlr]     ;Deals with current active controller
7983 000036AE E844140000                call .ehciGetOpBase             ;Return opregs ADDRESS in eax
7984 000036B3 6631C9                    xor cx, cx
7985 000036B6 678120DFFFFFFF            and dword [eax + ehcicmd], 0FFFFFFDFh ;Stop processing async
7986                                .espc0:
7987 000036BD 66FFC9                    dec cx
7988 000036C0 740E                      jz .espcfail
7989 000036C2 67F7400400800000          test dword [eax + ehcists], 08000h
7990 000036CA 75F1                      jnz .espc0
7991
7992 000036CC F8                        clc
7993 000036CD 59                        pop rcx
7994 000036CE 58                        pop rax
7995 000036CF C3                        ret
7996                                .espcfail:
7997 000036D0 F9                        stc
7998 000036D1 59                        pop rcx
7999 000036D2 58                        pop rax
8000 000036D3 C3                        ret
8001
8002                                .ehciProcessCommand:
8003                                ; Allows EHCI async schedule to process commands.
8004                                ; Preserves all registers except CF
8005                                ; Returns: CF=CY if error detected
8006                                ;           CF=NC if no error detected
8007                                ;
8008                                ; If returned with CF=CY, caller must read the msdStatus byte
8009 000036D4 50                        push rax
8010 000036D5 53                        push rbx
8011 000036D6 51                        push rcx
8012 000036D7 57                        push rdi
8013
8014 000036D8 880C25[49020000]          mov byte [eAsyncMutex], cl   ;Set mutex
8015 000036DF 8A0425[47020000]          mov al, byte [eActiveCtrlr]     ;Deals with current active controller
8016 000036E6 E80C140000                call .ehciGetOpBase             ;Return opregs ADDRESS in eax
8017 000036EB 4889C3                    mov rbx, rax
8018 000036EE 66BF8813                  mov di, 5000
8019 000036F2 E8B9FDFFFF                call .ehciLinkNewQHead
8020 000036F7 0F82A1000000              jc .epcfailedstart
8021                                .epc1:
8022 000036FD 67F7430413000000          test dword [ebx + ehcists], 13h
8023 00003705 7516                      jnz .epc2      ;If bits we care about are set, call IRQ proceedure
8024 00003707 F390                      pause
8025 00003709 66FFCF                    dec di
8026 0000370C 0F849F000000              jz .epcfailtimeout
```

```
8027 00003712 B486                      mov ah, 86h
8028 00003714 B901000000                mov ecx, 1      ;Max 5s in 1ms chunks
8029 00003719 CD35                      int 35h
8030 0000371B EBE0                      jmp short .epc1
8031                                .epc2:
8032 0000371D 89D8                      mov eax, ebx      ;Get opreg base into eax before we proceed into IRQ handler
8033 0000371F E8A3D4FFFF                call ehci_IRQ.nonIRQep ;Manually call IRQ
8034 00003724 F60425[48020000]10        test byte [eActiveInt], 10h ;HC error bit
8035 0000372C 7578                      jnz .epcHostError     ;HC error detected
8036 0000372E F60425[49020000]00        test byte [eAsyncMutex], 0
8037 00003736 75C5                      jnz .epc1      ;If the mutex isnt cleared, go back to sts check
8038 00003738 E853FDFFFF                call .ehciDelinkOldQHead     ;Perform delink
8039 0000373D E826FDFFFF                call .ehciToggleTransactingQHead      ;Toggle the active Qheads
8040                                ;Now set doorbell
8041 00003742 67810B40000000            or dword [ebx + ehcicmd], 40h     ;Ring Doorbell
8042 00003749 66BF8813                  mov di, 5000
8043                                .epc3:
8044 0000374D 67F7430420000000          test dword [ebx + ehcists], 20h ;Test for doorbell set high
8045 00003755 7512                      jnz .epc4
8046 00003757 F390                      pause
8047 00003759 66FFCF                    dec di
8048 0000375C 7440                      jz .epcfaildelinked
8049 0000375E B486                      mov ah, 86h
8050 00003760 B901000000                mov ecx, 1      ;Max 5s in 1ms chunks
8051 00003765 CD35                      int 35h
8052 00003767 EBE4                      jmp short .epc3
8053                                .epc4:
8054                                ;Clear once more to clear the doorbell bit
8055 00003769 678B4B04                  mov ecx, dword [ebx + ehcists]
8056 0000376D 67094B04                  or dword  [ebx + ehcists], ecx      ;WC high bits
8057                                ;Check if it was a stall
8058 00003771 F60425[48020000]02        test byte [eActiveInt], 2h   ;Check USBError bit
8059 00003779 7509                      jnz .epcexit
8060 0000377B C60425[A9010000]00        mov byte [msdStatus], 00h    ;No error... yet
8061 00003783 F8                        clc
8062                                .epcexit:
8063 00003784 5F                        pop rdi
8064 00003785 59                        pop rcx
8065 00003786 5B                        pop rbx
8066 00003787 58                        pop rax
8067 00003788 C3                        ret
8068                                .epcStall:
8069 00003789 C60425[A9010000]21        mov byte [msdStatus], 21h    ;General Controller Failure - Stall
8070 00003791 F9                        stc
8071 00003792 EBF0                      jmp short .epcexit
8072                                .epcfail:
8073 00003794 E8F7FCFFFF                call .ehciDelinkOldQHead     ;Perform delink
8074 00003799 E8CAFCFFFF                call .ehciToggleTransactingQHead      ;Toggle the active Qheads
8075                                .epcfailedstart: ;No need to delink as that data structure is considered garbage
8076                                .epcfaildelinked:
8077 0000379E 678B4B04                  mov ecx, dword [ebx + ehcists]
8078 000037A2 67094B04                  or dword  [ebx + ehcists], ecx      ;WC selected bits
8079                                .epcHostError:  ;Host error detected in interrupt register
8080 000037A6 C60425[A9010000]20        mov byte [msdStatus], 20h    ;General Controller Error
8081 000037AE F9                        stc
8082 000037AF EBD3                      jmp short .epcexit
8083                                .epcfailtimeout:
8084                                ;Called in the event that the schedule fails to process the QHead.
8085                                ;Emergency stops the currently transacting schedule
8086 000037B1 E8DAFCFFFF                call .ehciDelinkOldQHead     ;Perform delink
8087 000037B6 E8ADFCFFFF                call .ehciToggleTransactingQHead      ;Toggle the active Qheads
8088 000037BB 678B4B04                  mov ecx, dword [ebx + ehcists]
8089 000037BF 67094B04                  or dword  [ebx + ehcists], ecx      ;WC selected bits
8090 000037C3 C60425[A9010000]80        mov byte [msdStatus], 80h    ;Timeout Error
8091 000037CB F9                        stc
8092 000037CC EBB6                      jmp short .epcexit   ;Delink
8093
8094                                .ehciEnumerateRootPort:
8095                                ;This function discovers whether a device is of a valid type
8096                                ;or not.
8097                                ;Input: dl=port number - 1 (0 based), dh = bus [0-3]
8098                                ;        r10b = Host hub address (if the device is on a hub, 0 else)
8099                                ;Output:       CF=CY if error, CF=NC if bus transaction occured
8100                                ;              ZF=ZR if passed enum: ah = bus number, al = Address number
8101                                ;               ZF=NZ if the device failed enumeration: ax=error code
8102                                ;                     ah = Enum stage, al = Sub function stage
8103 000037CE 53                        push rbx
8104 000037CF 51                        push rcx
8105 000037D0 52                        push rdx
8106 000037D1 55                        push rbp
```

```
000037D2 4150                           push r8
000037D4 4151                           push r9
000037D6 4152                           push r10
000037D8 4153                           push r11

                                    .eebinit:
000037DA 6631ED                         xor bp, bp     ;Use as error counter     (Stage 0)
000037DD 88D0                           mov al, dl
000037DF E818FAFFFF                     call .ehciResetControllerPort     ;Reset port
000037E4 0F828C010000                   jc .ehciedbadnotimeout
                                    ;Power on debounce!
000037EA B9C8000000                     mov ecx, debounceperiod     ;debounce period
000037EF B486                           mov ah, 86h
000037F1 CD35                           int 35h

000037F3 66FFC5                         inc bp     ;Increment Error Counter     (Stage 1)
                                    .eeb0:
000037F6 48BB80060001000008-            mov rbx, 00008000001000680h     ;Pass get minimal device
                                                                        descriptor
000037FF 00
00003800 48891C25[80030000]             mov qword [ehciDataOut], rbx
00003808 66B94000                       mov cx, 40h     ;Pass default endpoint size
0000380C 30C0                           xor al, al
0000380E E898FDFFFF                     call .ehciGetRequest
00003813 0F8245010000                   jc .ehciedexit  ;Fast exit with carry set
                                    .eeb1:
00003819 66FFC5                         inc bp     ;Increment Error Counter     (Stage 2)
0000381C 30C0                           xor al, al     ;Increment Error subcounter     (Substage 0)
0000381E 48BB-                          mov rbx, ehciDataIn
00003820 [C003000000000000]
00003828 807B0101                       cmp byte [rbx + 1], 01h     ;Verify this is a valid dev
                                                                        descriptor
0000382C 0F8539010000                   jne .ehciedbad
00003832 FEC0                           inc al     ;Increment Error subcounter     (Substage 1)
00003834 66817B020002                   cmp word [rbx + 2], 0200h     ;Verify this is a USB 2.0 device
                                                                        or above
0000383A 0F822B010000                   jb .ehciedbad
00003840 FEC0                           inc al     ;Increment Error subcounter     (Substage 2)
00003842 807B0400                       cmp byte [rbx + 4], 0     ;Check interfaces
00003846 7410                           je .eeb2
00003848 807B0408                       cmp byte [rbx + 4], 08h     ;MSD?
0000384C 740A                           je .eeb2
0000384E 807B0409                       cmp byte [rbx + 4], 09h     ;Hub?
00003852 0F8513010000                   jne .ehciedbad
                                    .eeb2:
00003858 66FFC5                         inc bp     ;Increment Error Counter     (Stage 3)
0000385B 440FB64307                     movzx r8d, byte [rbx + 7]     ;Byte 7 is MaxPacketSize0, save in
                                                                        r8b
00003860 88D0                           mov al, dl

00003862 E895F9FFFF                     call .ehciResetControllerPort     ;Reset port again
00003867 0F82FE000000                   jc .ehciedbad
0000386D 49BB0A000000000000-            mov r11, 10
00003876 00
                                    .ehciEnumCommonEp:
00003877 66FFC5                         inc bp     ;Increment Error Counter     (Stage 4)
0000387A 88F0                           mov al, dh     ;Put bus number into al

0000387C E825030000                     call .ehciGiveValidAddress     ;Get a valid address for device
00003881 3C80                           cmp al, 80h
00003883 0F83E2000000                   jae .ehciedbad     ;Invalid address

00003889 66FFC5                         inc bp     ;Increment Error Counter     (Stage 5)
0000388C 4188C1                         mov r9b, al          ;Save the new device address number in r9b
                                    .eeb3:
0000388F BB00050000                     mov ebx, 0500h     ;Set address function
00003894 410FB6C9                       movzx ecx, r9b     ;move new address into ecx
00003898 C1E110                         shl ecx, 8*2
0000389B 09CB                           or ebx, ecx     ;Add address number to ebx
0000389D 664489C1                       mov cx, r8w     ;Move endpoint size into cx
000038A1 30C0                           xor al, al     ;Device still talks on address 0, ax not preserved
000038A3 E864FCFFFF                     call .ehciSetNoData     ;Set address
000038A8 0F82B0000000                   jc .ehciedexit  ;Fast exit with carry set
                                    .eeb4:
000038AE B486                           mov ah, 86h
000038B0 4C89D9                         mov rcx, r11
000038B3 CD35                           int 35h

000038B5 66FFC5                         inc bp     ;Increment Error Counter     (Stage 6)
                                    .eeb5:
000038B8 48BB80060001000012-            mov rbx, 00012000001000680h     ;Now get full device descriptor
000038C1 00
000038C2 4488C8                         mov al, r9b     ;Get address
```

```
8182 000038C5 664489C1                  mov cx, r8w
8183 000038C9 E8DDFCFFFF                call .ehciGetRequest     ;Get full device descriptor and discard
8184 000038CE 0F828A000000              jc .ehciedexit  ;Fast exit with carry set
8185 000038D4 66FFC5                    inc bp     ;Increment Error Counter     (Stage 7/0Bh)
8186                                .eeb6:
8187 000038D7 48BB80060002000000-       mov rbx, 00000000002000680h ;Get config descriptor
8187 000038E0 00
8188 000038E1 4489C1                    mov ecx, r8d     ;Adjust the packet data with bMaxPacketSize0
8189 000038E4 48C1E130                  shl rcx, 8*6     ;cx contains bMaxPacketSize0
8190 000038E8 4809CB                    or rbx, rcx
8191 000038EB 4488C8                    mov al, r9b    ;Get address
8192 000038EE 664489C1                  mov cx, r8w    ;Move endpoint size into cx
8193 000038F2 E8B4FCFFFF                call .ehciGetRequest
8194 000038F7 7265                      jc .ehciedexit  ;Fast exit with carry set
8195                                .eeb7:
8196 000038F9 66FFC5                    inc bp     ;Increment Error Counter     (Stage 8/0Ch)
8197                                ;Find a valid interface in this config
8198 000038FC E8CB020000                call .ehciFindValidInterface
8199 00003901 7268                      jc .ehciedbad     ;Dont set config, exit bad
8200                                ;If success, ah has device type (0=msd, 1=hub), al = Interface to
                                                                          use
8201                                ;rbx points to interface descriptor
8202 00003903 66FFC5                    inc bp     ;Increment Error Counter     (Stage 9/0Dh)
8203 00003906 E889000000                call .ehciAddDeviceToTables
8204 0000390B 725E                      jc .ehciedbad     ;Failed to be added to internal tables
8205 0000390D FE0425[35020000]          inc byte [usbDevices]    ;Device added successfully, inc byte
8206                                ;Set configuration 1 (wie OG Windows, consider upgrading soon)
8207 00003914 66FFC5                    inc bp     ;Increment Error Counter     (Stage 0Ah/0Ch)
8208                                .eeb8:
8209 00003917 48BB00090100000000-       mov rbx, 00000000000010900h     ;Set configuration 1 (function
                                                                          09h)
8209 00003920 00
8210 00003921 4488C8                    mov al, r9b    ;Get address
8211 00003924 664489C1                  mov cx, r8w    ;Move endpoint size into cx
8212 00003928 E8DFFBFFFF                call .ehciSetNoData
8213 0000392D 722F                      jc .ehciedexit  ;Fast exit with carry set
8214                                .eeb9:
8215 0000392F 66FFC5                    inc bp     ;Increment Error Counter     (Stage 0Bh/0Dh)
8216                                .eeb10:
8217 00003932 48BB80080000000001-       mov rbx, 0001000000000880h  ;Get device config (sanity check)
8217 0000393B 00
8218 0000393C 410FB7C8                  movzx ecx, r8w                ;bMaxPacketSize0
8219 00003940 4488C8                    mov al, r9b                   ;Get device address
8220 00003943 E863FCFFFF                call .ehciGetRequest
8221 00003948 7214                      jc .ehciedexit  ;Fast exit with carry set
8222                                .eeb11:
8223 0000394A 66FFC5                    inc bp     ;Increment Error Counter     (Stage 0Ch/0Eh)
8224 0000394D 803C25[C0030000]01        cmp byte [ehciDataIn], 01
8225 00003955 7531                      jne .ehcibadremtables
8226                                ;Device is now configured and ready to go to set/reset
8227 00003957 88F4                      mov ah, dh  ;Move bus number
8228 00003959 4488C8                    mov al, r9b ;Move address number
8229 0000395C 31D2                      xor edx, edx  ;This will always set the zero flag
8230                                .ehciedexit:
8231 0000395E 415B                      pop r11
8232 00003960 415A                      pop r10
8233 00003962 4159                      pop r9
8234 00003964 4158                      pop r8
8235 00003966 5D                        pop rbp
8236 00003967 5A                        pop rdx
8237 00003968 59                        pop rcx
8238 00003969 5B                        pop rbx
8239 0000396A C3                        ret
8240                                .ehciedbad:
8241                                .ehciedbadnoport:
8242 0000396B 50                        push rax
8243 0000396C B486                      mov ah, 86h
8244 0000396E B9F4010000                mov ecx, 500     ;500 ms wait between failed attempts
8245 00003973 CD35                      int 35h
8246 00003975 58                        pop rax
8247                                .ehciedbadnotimeout:
8248 00003976 88C4                      mov ah, al     ;Save subproc error code
8249 00003978 30C0                      xor al, al     ;Zero byte
8250 0000397A 6609E8                    or ax, bp    ;Add proc error stage code into al
8251 0000397D 86E0                      xchg ah, al
8252 0000397F 6631ED                    xor bp, bp
8253 00003982 66FFC5                    inc bp        ;This will always clear the Zero flag
8254 00003985 F8                        clc           ;This will force clear the Carry flag
8255 00003986 EBD6                      jmp short .ehciedexit
8256                                .ehcibadremtables:
8257 00003988 4488C8                    mov al, r9b ;Get address low
8258 0000398B 88F4                      mov ah, dh
8259 0000398D E87F010000                call .ehciRemoveDevFromTables
```

```
8260 00003992 EBE2                      jmp short .ehciedbadnotimeout
8261
8262                                .ehciAddDeviceToTables:
8263                                ;This function adds a valid device to the internal tables.
8264                                ;Interrupts are off for this to avoid dead entries
8265                                ;Input: ah = device type (0=msd, 1=hub)
8266                                ;       al = Interface Value to use (USB bInterfaceNumber)
8267                                ;       rbx = Ptr to valid Interface descriptor
8268                                ;       r8b = MaxPacketSize0
8269                                ;       r9b = Device Address
8270                                ;       dh = Bus number
8271                                ;       dl = Physical Port number - 1
8272                                ;       r10b = Host hub address
8273 00003994 4153                      push r11
8274 00003996 55                        push rbp     ;Error counter
8275 00003997 57                        push rdi
8276 00003998 53                        push rbx
8277 00003999 52                        push rdx
8278 0000399A 9C                        pushfq
8279 0000399B FEC2                      inc dl       ;Add one to the Physical port number (kludge for
                                                       root hub enum)
8280 0000399D 6631ED                    xor bp, bp   ;Zero error counter (Stage 0)
8281 000039A0 B90A000000                mov ecx, usbMaxDevices
8282 000039A5 380C25[35020000]          cmp byte [usbDevices], cl    ;Max number of devices, check
8283 000039AC 0F8458010000              je .eadttbad          ;If max, fail
8284 000039B2 66FFC5                    inc bp       ;Increment error counter (Stage 1)
8285 000039B5 48BF-                     mov rdi, usbDevTbl
8285 000039B7 [4C02000000000000]
8286 000039BF B10A                      mov cl, usbDevTblE   ;Within the length of the table
8287                                ;Write Common table first
8288                                .eadtt0:
8289 000039C1 800F00                    or byte [rdi], 0    ;Check if there exists a free entry
8290 000039C4 7411                      jz .eadtt1
8291 000039C6 4881C703000000            add rdi, usbDevTblEntrySize ;Go to next entry
8292 000039CD FEC9                      dec cl
8293 000039CF 0F8435010000              jz .eadttbad
8294 000039D5 EBEA                      jmp short .eadtt0
8295                                .eadtt1:
8296 000039D7 66FFC5                    inc bp       ;Increment error counter (Stage 2)
8297 000039DA 80C408                    add ah, 08h ;hub is 09h
8298                                ;Add device here, rdi points to entry
8299 000039DD 44880F                    mov byte [rdi], r9b
8300 000039E0 887701                    mov byte [rdi + 1], dh
8301 000039E3 886702                    mov byte [rdi + 2], ah
8302                                ;Entry written
8303 000039E6 66FFC5                    inc bp       ;Increment error counter (Stage 3)
8304                                ;Individual Device table writing
8305 000039E9 80FC08                    cmp ah, 08h
8306 000039EC 740E                      je .eadttmsd
8307 000039EE 80FC09                    cmp ah, 09h
8308 000039F1 0F84C9000000              je .eadtthub
8309 000039F7 E90E010000                jmp .eadttbad
8310                                .eadttmsd:
8311 000039FC 48BF-                     mov rdi, msdDevTbl
8311 000039FE [BA02000000000000]
8312 00003A06 B10A                      mov cl, msdDevTblE   ;Max entries possible
8313 00003A08 66FFC5                    inc bp       ;Increment error counter (Stage 4)
8314                                .eadttmsd0:
8315 00003A0B 800F00                    or byte [rdi], 0
8316 00003A0E 7411                      jz .eadttmsd1
8317 00003A10 4881C710000000            add rdi, msdDevTblEntrySize
8318 00003A17 FEC9                      dec cl
8319 00003A19 0F84EB000000              jz .eadttbad
8320 00003A1F EBEA                      jmp short .eadttmsd0
8321                                .eadttmsd1:
8322                                ;rdi points to correct offset into table
8323                                ;rbx points to interface
8324 00003A21 8A4B04                    mov cl, byte [rbx + 4]    ;Get number of endpoints to check
8325 00003A24 88CD                      mov ch, cl
8326 00003A26 66FFC5                    inc bp       ;Increment error counter (Stage 5)
8327 00003A29 4989DB                    mov r11, rbx     ;Save Interface Pointer in r11
8328 00003A2C 4881C309000000            add rbx, 9  ;Go to first IF
8329                                .eadttmsd11:
8330 00003A33 50                        push rax
8331 00003A34 668B4302                  mov ax, word [rbx + 2]
8332 00003A38 66C1E804                  shr ax, 4    ;Remove low 4 bits
8333 00003A3C 663D2800                  cmp ax, 28h       ;Bulk/In bits
8334 00003A40 58                        pop rax           ;Doesnt ruin flags
8335 00003A41 7411                      je .eadttmsd2     ;Not zero only if valid
8336 00003A43 4881C307000000            add rbx, 7    ;Go to next endpoint
8337 00003A4A FEC9                      dec cl
8338 00003A4C 0F84B8000000              jz .eadttbad
8339 00003A52 EBDF                      jmp short .eadttmsd11
```

```
                                     .eadttmsd2:
00003A54 44880F                          mov byte [rdi], r9b       ;Device Address
00003A57 887701                          mov byte [rdi + 1], dh    ;Root hub/bus
00003A5A 44885702                        mov byte [rdi + 2], r10b ;Address of parent device if not root
00003A5E 885703                          mov byte [rdi + 3], dl    ;Port number we are inserted in
00003A61 884704                          mov byte [rdi + 4], al    ;Save Interface number
00003A64 418A4306                        mov al, byte [r11 + 6]    ;bInterfaceSubclass is +6
00003A68 884705                          mov byte [rdi + 5], al
00003A6B 418A4307                        mov al, byte [r11 + 7]    ;Protocol
00003A6F 884706                          mov byte [rdi + 6], al
00003A72 44884707                        mov byte [rdi + 7], r8b   ;MaxPacketSize0
                                     ;Valid In EP found, write table entries
00003A76 8A4302                          mov al, byte [rbx + 2]   ;Get address
00003A79 884708                          mov byte [rdi + 8], al
00003A7C 668B4304                        mov ax, word [rbx + 4]   ;Get maxPacketSizeIn
00003A80 66894709                        mov word [rdi + 9], ax

00003A84 498D5B09                        lea rbx, qword [r11 + 9]    ;Return rbx to first IF
00003A88 66FFC5                          inc bp        ;Increment error counter (Stage 6)
                                     .eadttmsd21:
00003A8B 668B4302                        mov ax, word [rbx + 2]   ;Bulk/Out bits
00003A8F 66C1E804                        shr ax, 4
00003A93 663D2000                        cmp ax, 20h
00003A97 740D                            je .eadttmsd3    ;Not zero only if valid
00003A99 4881C307000000                  add rbx, 7    ;Go to next endpoint
00003AA0 FECD                            dec ch
00003AA2 7466                            jz .eadttbad
00003AA4 EBE5                            jmp short .eadttmsd21
                                     .eadttmsd3:
00003AA6 8A4302                          mov al, byte [rbx + 2]   ;Get address
00003AA9 88470B                          mov byte [rdi + 11], al
00003AAC 668B4304                        mov ax, word [rbx + 4]   ;Get maxPacketSizeIn
00003AB0 6689470C                        mov word [rdi + 12], ax
00003AB4 6631C0                          xor ax, ax   ;Zero ax
00003AB7 6689470E                        mov word [rdi + 14], ax ;Make dt bits for I/O EPs zero
                                     ;Table entry written for MSD device
00003ABB E93F000000                      jmp .eadttpass
                                     .eadtthub:
00003AC0 48BF-                           mov rdi, hubDevTbl
00003AC2 [6A02000000000000]
00003ACA B10A                            mov cl,  hubDevTblE ;Max entries possible
00003ACC 66BD0700                        mov bp, 7        ;Increment error counter (Stage 7)
                                     .eadtthub0:
00003AD0 800F00                          or byte [rdi], 0
00003AD3 740D                            jz .eadtthub1
00003AD5 4881C708000000                  add rdi, hubDevTblEntrySize
00003ADC FEC9                            dec cl
00003ADE 742A                            jz .eadttbad
00003AE0 EBEE                            jmp short .eadtthub0
                                     .eadtthub1:
                                     ;Valid table space found
00003AE2 44880F                          mov byte [rdi], r9b       ;Device Address
00003AE5 887701                          mov byte [rdi + 1], dh    ;Root hub/bus
00003AE8 44885702                        mov byte [rdi + 2], r10b ;Address of parent device if not root
00003AEC 885703                          mov byte [rdi + 3], dl    ;Port number we are inserted in
00003AEF 44884704                        mov byte [rdi + 4], r8b   ;MaxPacketSize0
00003AF3 66B800FF                        mov ax, 0FF00h  ;Res byte is 0FFh, Num ports (byte 6) is 0
00003AF7 66894705                        mov word [rdi + 5], ax    ;Number of ports and PowerOn2PowerGood
00003AFB C64707FF                        mov byte [rdi + 7], 0FFh      ;EP address, currently reserved
                                     .eadttpass:
00003AFF 9D                              popfq    ;If IF was clear, it will be set clear by popf
00003B00 6631C0                          xor ax, ax   ;Clear ax and clc
                                     .eadttexit:
00003B03 5A                              pop rdx
00003B04 5B                              pop rbx
00003B05 5F                              pop rdi
00003B06 5D                              pop rbp
00003B07 415B                            pop r11
00003B09 C3                              ret
                                     .eadttbad:
00003B0A 9D                              popfq    ;If IF was clear, it will be set clear by popf
00003B0B F9                              stc
00003B0C 6689E8                          mov ax, bp
00003B0F EBF2                            jmp short .eadttexit
                                     .ehciRemoveDevFromTables:
                                     ;This function removes a function from internal tables
                                     ;Input: al = Address number, ah = Bus number
                                     ;Output: Internal tables zeroed out, ax destroyed, Carry clear
                                     ;       If invalid argument, Carry set
00003B11 57                              push rdi
00003B12 51                              push rcx
00003B13 53                              push rbx
00003B14 48BF-                           mov rdi, usbDevTbl
```

```
00003B16 [4C02000000000000]
00003B1E B10A                           mov cl, usbDevTblE      ;10 entries possible
                                    .erdft0:
00003B20 66AF                           scasw
00003B22 7409                           je .erdft1      ;Device signature found
00003B24 48FFC7                         inc rdi
00003B27 FEC9                           dec cl
00003B29 7478                           jz .erdftbad
00003B2B EBF3                           jmp short .erdft0
                                    .erdft1:
00003B2D 4881EF02000000                 sub rdi, 2  ;scasw pointers to the next word past the comparison
00003B34 8A6702                         mov ah, byte [rdi + 2]      ;Save class code in ah
00003B37 80FC08                         cmp ah, 08h ;USB MSD Class device
00003B3A 7507                           jne .erdft11      ;Skip the dec if it is a hub class device
00003B3C FE0C25[4B020000]               dec byte [numMSD]     ;Device is being removed from tables,
                                                                  decrement count
                                    .erdft11:
                                    ;Clear usbDevTbl entry for usb device
00003B43 50                             push rax
00003B44 B903000000                     mov ecx, usbDevTblEntrySize      ;Table entry size
00003B49 30C0                           xor al, al
00003B4B F3AA                           rep stosb      ;Store zeros for entry
00003B4D 58                             pop rax

00003B4E 48BB-                          mov rbx, hubDevTbl
00003B50 [6A02000000000000]
00003B58 48B9-                          mov rcx, msdDevTbl
00003B5A [BA02000000000000]
00003B62 80FC09                         cmp ah, 09h
00003B65 480F44CB                       cmove rcx, rbx ;If 09h (Hub), change table pointed to by rcx
00003B69 4889CF                         mov rdi, rcx      ;Point rdi to appropriate table
00003B6C BB08000000                     mov ebx, hubDevTblEntrySize      ;Size of hub table entry
00003B71 B910000000                     mov ecx, msdDevTblEntrySize      ;Size of msd table entry
00003B76 80FC09                         cmp ah, 09h
00003B79 0F44CB                         cmove ecx, ebx      ;If hub, move size into cx
                                    ;cx has entry size, rdi points to appropriate table
00003B7C 4889FB                         mov rbx, rdi
00003B7F 31FF                           xor edi, edi
00003B81 29CF                           sub edi, ecx
00003B83 B411                           mov ah, 11h
                                    .erdft2:
00003B85 FECC                           dec ah
00003B87 741A                           jz .erdftbad      ;Somehow, address not found
00003B89 01CF                           add edi, ecx
00003B8B 3A043B                         cmp al, byte [rbx + rdi]
00003B8E 75F5                           jne .erdft2
00003B90 4801DF                         add rdi, rbx      ;point rdi to table entry
00003B93 30C0                           xor al, al
00003B95 F3AA                           rep stosb      ;ecx contains table entry size in bytes
00003B97 FE0C25[35020000]               dec byte [usbDevices]     ;Decrement total usb devices
00003B9E F8                             clc
                                    .erdftexit:
00003B9F 5B                             pop rbx
00003BA0 59                             pop rcx
00003BA1 5F                             pop rdi
00003BA2 C3                             ret
                                    .erdftbad:
00003BA3 F9                             stc
00003BA4 EBF9                           jmp short .erdftexit
                                    .ehciGiveValidAddress:
                                    ;This function will return a valid value to use as an address
                                    ;for a new device.
                                    ;Input: al = Controller number [0-3]
                                    ;Output: al = Address, or 80h => No valid available address
00003BA6 57                             push rdi
00003BA7 51                             push rcx
00003BA8 88C4                           mov ah, al      ;Move bus number high
00003BAA B000                           mov al, 0 ;Address 0, start at addr 1
                                    .egva0:
00003BAC FEC0                           inc al
00003BAE 3C80                           cmp al, 80h
00003BB0 7317                           jae .egvaexit
00003BB2 48BF-                          mov rdi, usbDevTbl
00003BB4 [4C02000000000000]
00003BBC B10A                           mov cl, usbDevTblE      ;10 entries possible
                                    .egva1:
00003BBE 66AF                           scasw
00003BC0 74EA                           je .egva0
00003BC2 48FFC7                         inc rdi      ;Pass third byte in table entry
00003BC5 FEC9                           dec cl
00003BC7 75F5                           jnz .egva1      ;Check every entry for any addresses being used
                                    .egvaexit:
00003BC9 59                             pop rcx
```

```
00003BCA 5F                         pop rdi
00003BCB C3                         ret
                                .ehciFindValidInterface:
                                ;A proc to check a valid interface descriptor is present.
                                ;Input: Nothing [Assumes Get Config was called in standard buffer]
                                ;Output: Carry set if invalid. Carry clear if valid.
                                ;     On success: ah = device type (0 is msd, 1 is hub)
                                ;                 al = interface number to set
                                ;                 rbx = Pointer to Interface Descriptor
                                ;    On fail: al contains error code, registers rbx, cx, dx destroyed
00003BCC 56                         push rsi
00003BCD 57                         push rdi
00003BCE 51                         push rcx
00003BCF 52                         push rdx

00003BD0 48BE-                      mov rsi, ehciDataIn     ;Shift to buffer
00003BD2 [C003000000000000]
00003BDA 30D2                       xor dl, dl    ;Error code counter
00003BDC 807E0102                   cmp byte [rsi + 1], 02h    ;Check if valid config descriptor
00003BE0 753F                       jne .ecvifail
00003BE2 FEC2                       inc dl
                                ;cl counts ep's per interface, ch counts possible interfaces
00003BE4 8A6E05                     mov ch, byte [rsi + 5]         ;Get number of interfaces
                                .ecvi0:
00003BE7 84ED                       test ch, ch
00003BE9 7436                       jz .ecvifail     ;Zero interfaces is invalid for us
00003BEB FEC2                       inc dl

00003BED 4889F3                     mov rbx, rsi     ;Save this descriptor in rbx
00003BF0 480FB633                   movzx rsi, byte [rbx]     ;get the size of the config to skip
00003BF4 4801DE                     add rsi, rbx     ;point rsi to head of first interface descriptor
00003BF7 807E0104                   cmp byte [rsi + 1], 04h     ;Check if valid interface descriptor
00003BFB 7524                       jne .ecvifail
00003BFD FEC2                       inc dl
00003BFF 8A4E04                     mov cl, byte [rsi + 4]
                                ;Cmp IF has valid class/prototcol
00003C02 4831C0                     xor rax, rax     ;Device signature, 0 is msd, 1 is hub
00003C05 E859000000                 call .ehciCheckMsdIf
00003C0A 7309                       jnc     .ecviif     ;Not clear => valid interface
00003C0C FEC4                       inc ah     ;Device signature, 0 is msd, 1 is hub
00003C0E E831000000                 call .ehciCheckHubIf
00003C13 7213                       jc     .ecvibadif     ;Clear => bad interface
                                .ecviif:     ;Valid interface found
00003C15 8A4602                     mov al, byte [rsi + 2]     ;Get interface number into al
00003C18 4889F3                     mov rbx, rsi     ;Save pointer in rbx for return
00003C1B F8                         clc ;Clear carry
                                .ecviexit:
00003C1C 5A                         pop rdx
00003C1D 59                         pop rcx
00003C1E 5F                         pop rdi
00003C1F 5E                         pop rsi
00003C20 C3                         ret
                                .ecvifail:
00003C21 31DB                       xor ebx, ebx     ;Zero rbx for bad returns
00003C23 F9                         stc
00003C24 88D0                       mov al, dl     ;Move error code
00003C26 EBF4                       jmp short .ecviexit
                                .ecvibadif:     ;Bad interface, goto next interface
00003C28 84C9                       test cl, cl
00003C2A 740B                       jz .ecvibadif1
00003C2C FEC9                       dec cl
00003C2E 4881C607000000             add rsi, 7
00003C35 EBF1                       jmp short .ecvibadif
                                .ecvibadif1:
00003C37 4881C609000000             add rsi, 9
00003C3E FECD                       dec ch
00003C40 B201                       mov dl, 1
00003C42 EBA3                       jmp short .ecvi0
                                .ehciCheckHubIf:
                                ;Input: rsi points to interface descriptor
                                ;Output: All registers preserved, carry set if NOT valid hub
00003C44 56                         push rsi
00003C45 807E0509                   cmp byte [rsi + 5], 09h
00003C49 7515                       jne .ecdhfail
00003C4B 807E0600                   cmp byte [rsi + 6], 0
00003C4F 750F                       jne .ecdhfail
00003C51 807E0702                   cmp byte [rsi + 7], 2
00003C55 7709                       ja .ecdhfail
00003C57 807E0401                   cmp byte [rsi + 4], 1     ;One endpoint to rule them all
00003C5B 7503                       jne .ecdhfail
00003C5D F8                         clc
                                .ecdhexit:
00003C5E 5E                         pop rsi
```

```
00003C5F C3                          ret
                                 .ecdhfail:
00003C60 F9                          stc
00003C61 EBFB                        jmp short .ecdhexit
                                 .ehciCheckMsdIf:
                                 ;Input: rsi points to interface descriptor
                                 ;Output: Carry set if fail, ax destroyed
                                 ;     rsi points to good descriptor if all ok
                                 ;Note we only accept 09/00/50 and 09/06/50
00003C63 56                          push rsi
00003C64 53                          push rbx
00003C65 51                          push rcx
00003C66 807E0508                    cmp byte [rsi + 5], 08h     ;MSD class
00003C6A 7517                        jne .ecdmfail
                                 ;Subclass check
00003C6C 807E0606                    cmp byte [rsi + 6], 06h     ;SCSI actual
00003C70 7406                        je .ecdmprot
00003C72 807E0600                    cmp byte [rsi + 6], 00h     ;SCSI defacto
00003C76 750B                        jne .ecdmfail
                                 .ecdmprot:
00003C78 807E0750                    cmp byte [rsi + 7], 50h     ;BBB
00003C7C 7505                        jne .ecdmfail
                                 .ecdmprotUAF:    ;Dummy label to find where to add this later
                                 .ecdmpass:
00003C7E F8                          clc
                                 .ecdmexit:
00003C7F 59                          pop rcx
00003C80 5B                          pop rbx
00003C81 5E                          pop rsi
00003C82 C3                          ret
                                 .ecdmfail:
00003C83 F9                          stc
00003C84 EBF9                        jmp short .ecdmexit
                                 .ehciGetDevicePtr:
                                 ;Gets address/bus pair and returns in rax a pointer to the data
                                 ;structure of the device, in the data table.
                                 ;Input: ah = bus number, al = Address number
                                 ;Output: ax = Preserved, rsi = Pointer to table structure, bl = USB
                                                                Class Code
00003C86 51                          push rcx
00003C87 52                          push rdx
00003C88 55                          push rbp
00003C89 B90A000000                  mov ecx, usbMaxDevices
00003C8E 48BE-                       mov rsi, usbDevTbl
00003C90 [4C02000000000000]

                                 .egdp0:
00003C98 663B06                      cmp ax, word [rsi]
00003C9B 740E                        je .egdp1    ;Device found
00003C9D 4881C603000000              add rsi, usbDevTblEntrySize
00003CA4 66FFC9                      dec cx
00003CA7 7447                        jz .egdpfail     ;Got to the end with no dev found, exit
00003CA9 EBED                        jmp short .egdp0
                                 .egdp1:
00003CAB 48BD-                       mov rbp, hubDevTbl
00003CAD [6A02000000000000]

00003CB5 B908000000                  mov ecx, hubDevTblEntrySize
00003CBA 0FB65E02                    movzx ebx, byte [rsi + 2]   ;Return bl for device type
00003CBE 80FB09                      cmp bl, 09h ;Are we hub?
00003CC1 48BE-                       mov rsi, msdDevTbl   ;Set to msd
00003CC3 [BA02000000000000]

00003CCB BA10000000                  mov edx, msdDevTblEntrySize
00003CD0 480F44F5                    cmove rsi, rbp   ;If hub, reload rsi pointer to hub table
00003CD4 0F44D1                      cmove edx, ecx     ;If hub, reload dx with hub table size
00003CD7 B90A000000                  mov ecx, usbMaxDevices
                                 .egdp2:
00003CDC 663B06                      cmp ax, word [rsi]
00003CDF 740A                        je .egdp3
00003CE1 4801D6                      add rsi, rdx     ;rdx contains size of entry for either table
00003CE4 66FFC9                      dec cx
00003CE7 7407                        jz .egdpfail
00003CE9 EBF1                        jmp short .egdp2
                                 .egdp3:
00003CEB F8                          clc
                                 .egdpexit:
00003CEC 5D                          pop rbp
00003CED 5A                          pop rdx
00003CEE 59                          pop rcx
00003CEF C3                          ret
                                 .egdpfail:
00003CF0 6631DB                      xor bx, bx
00003CF3 F9                          stc
00003CF4 EBF6                        jmp short .egdpexit

```

```
                                          .ehciProbeQhead:
                                          ;A proc that returns a Queue Heads' status byte in bl.
                                          ;Input:
                                          ;    rbx = Address of QHead to probe
                                          ;Output:
                                          ;    bl = Status byte, if 0, successful transfer!
00003CF6 8A5B18                               mov bl, byte [rbx + 18h]   ;08h is offset in qTD
00003CF9 C3                                   ret
                                          .ehciStandardErrorHandler:
                                          ;Attempts to verify if something went wrong in previous transaction.
                                          ;May only be called if eActiveInt has bit USBSTS bit set
                                          ;Input:  al = Device Address
                                          ;        cx = Default Endpoint Size
                                          ;Output: CF=CY: Host error, Reset host system
                                          ;        CF=NC: Proceed with below
                                          ;        al = 0 => Benign error, Make request again/Verify data.
                                          ;        al = 1 => Stall, Transaction error or Handshake error, corrected.
                                          ;        al = 80h => Fatal error, EPClear errored out, but no clear reason why
                                          ;        al > 80h => Bits 6-0 give the status byte for the error on EP Clear.
                                          ;                    Bit 7 is the fatal error bit.
                                          ;                    If set, recommend device is port reset.
                                          ;All other registers preserved
00003CFA 53                                   push rbx
00003CFB 4150                                 push r8
00003CFD 4151                                 push r9

00003CFF 4989C0                               mov r8, rax
00003D02 4989C9                               mov r9, rcx
00003D05 30C0                                 xor al, al                    ;Set error counter and clear CF
00003D07 F60425[48020000]02                   test byte [eActiveInt], 2     ;Error Interrupt
00003D0F 744B                                 jz .esehexit                  ;No error found, should not have been called
00003D11 488B1C25[3E020000]                   mov rbx, qword [eCurrAsyncHead] ;Get the current transacting QHead address
00003D19 E8D8FFFFFF                           call .ehciProbeQhead     ;Ret in bl status byte
00003D1E 80E378                               and bl, 01111000b        ;Check if it is something we should clear EP for
00003D21 7439                                 jz .esehexit             ;If it is not, benign error. al = 0

00003D23 488B1C25[3E020000]                   mov rbx, qword [eCurrAsyncHead] ;Get current AsyncHead again
00003D2B 4488C0                               mov al, r8b          ;Device Address
00003D2E 664489C9                             mov cx, r9w          ;EP size
00003D32 8A5B05                               mov bl, byte [rbx + 05h]   ;Get Endpoint to reset
00003D35 80E30F                               and bl, 0Fh ;Lo nybble only
00003D38 E828000000                           call .ehciClearEpStallHalt
00003D3D 721D                                 jc .esehexit           ;HC error!
00003D3F B001                                 mov al, 1              ;Stall cleared
00003D41 F60425[48020000]02                   test byte [eActiveInt], 2     ;Check if interrupt returned an error
00003D49 7411                                 jz .esehexit                  ;No error found, return al=1, stall cleared
00003D4B B080                                 mov al, 80h                   ;Fatal error indication
00003D4D 488B1C25[3E020000]                   mov rbx, qword [eCurrAsyncHead] ;Get the current transacting QHead address
00003D55 E89CFFFFFF                           call .ehciProbeQhead
00003D5A 08D8                                 or al, bl            ;Add error bits to al for Fatal error indication.
                                          .esehexit:
00003D5C 4C89C9                               mov rcx, r9
00003D5F 4159                                 pop r9
00003D61 4158                                 pop r8
00003D63 5B                                   pop rbx
00003D64 C3                                   ret

                                          .ehciClearEpStallHalt:
                                          ;Clears a halt or stall on an endpoint.
                                          ;Input: bl=Endpoint (0 for control)
                                          ;       al=Device Address
                                          ;       cx=Ctrl Endpoint Size
                                          ;Output:
                                          ;    CF=CY: Host error, Reset host system
                                          ;    CF=NC: Transaction succeeded, check interrupt error bit for confirmation
00003D65 53                                   push rbx
00003D66 480FB6DB                             movzx rbx, bl
00003D6A 48C1E320                             shl rbx, 2*10h   ;Shift wIndex by two words
00003D6E 4881CB02010000                       or rbx, 0102h    ;01=bRequest(CLEAR_FEATURE) 02h=bmRequestType (Endpoint)
00003D75 E892F7FFFF                           call .ehciSetNoData
00003D7A 5B                                   pop rbx ;Get original bx
```

```
00003D7B C3                      ret

                              .ehciWriteQHead:
                              ;Writes a Queue head at rdi, and clears the space for the transfer descriptor
                              ;rdi points at the head of the qhead on return
                              ;All non argument registers preserved
                              ;r8d=Horizontal Ptr + Typ + T
                              ;r9d=Endpoint Characteristics
                              ;r10d=Endpoint Capabilities
                              ;r11d=Next qTD Pointer
00003D7C 50                      push rax
00003D7D 51                      push rcx
00003D7E 57                      push rdi
00003D7F 57                      push rdi
00003D80 31C0                    xor eax, eax
00003D82 B911000000              mov ecx, 17
00003D87 F3AB                    rep stosd
00003D89 5F                      pop rdi
00003D8A 4489C0                  mov eax, r8d
00003D8D AB                      stosd
00003D8E 4489C8                  mov eax, r9d
00003D91 AB                      stosd
00003D92 4489D0                  mov eax, r10d
00003D95 AB                      stosd
00003D96 31C0                    xor eax, eax
00003D98 AB                      stosd           ;Enter 0 for the current qTD pointer entry
00003D99 4489D8                  mov eax, r11d
00003D9C AB                      stosd
00003D9D 5F                      pop rdi
00003D9E 59                      pop rcx
00003D9F 58                      pop rax
00003DA0 C3                      ret
                              .ehciWriteQHeadTD:
                              ;Writes a transfer descriptor at the location pointed at by rdi
                              ;rdi points at the head of the qheadTD on return
                              ;All registers except passed arguments, preserved
                              ;rdi=location for current linked list element
                              ;r8d=Next qTD ptr
                              ;r9d=Alternate Next qTD ptr
                              ;r10d=Transfer Descriptor Token
                              ;r11=Buffer Ptr 0 + Current Offset
00003DA1 50                      push rax
00003DA2 57                      push rdi
00003DA3 4489C0                  mov eax, r8d
00003DA6 AB                      stosd
00003DA7 4489C8                  mov eax, r9d
00003DAA AB                      stosd
00003DAB 4489D0                  mov eax, r10d
00003DAE AB                      stosd
00003DAF 4489D8                  mov eax, r11d
00003DB2 AB                      stosd
00003DB3 2500F0FFFF              and eax, 0FFFFF000h
00003DB8 0500100000              add eax, 1000h
00003DBD AB                      stosd
00003DBE 0500100000              add eax, 1000h
00003DC3 AB                      stosd
00003DC4 0500100000              add eax, 1000h
00003DC9 AB                      stosd
00003DCA 0500100000              add eax, 1000h
00003DCF AB                      stosd

00003DD0 4C89D8                  mov rax, r11
00003DD3 48C1C820                ror rax, 20h
00003DD7 AB                      stosd
00003DD8 48C1C820                ror rax, 20h
00003DDC 482500F0FFFF            and rax, 0FFFFFFFFFFFFF000h
00003DE2 480500100000            add rax, 1000h
00003DE8 48C1C820                ror rax, 20h
00003DEC AB                      stosd
00003DED 48C1C820                ror rax, 20h
00003DF1 480500100000            add rax, 1000h
00003DF7 48C1C820                ror rax, 20h
00003DFB AB                      stosd
00003DFC 48C1C820                ror rax, 20h
00003E00 480500100000            add rax, 1000h
00003E06 48C1C820                ror rax, 20h
00003E0A AB                      stosd
00003E0B 48C1C820                ror rax, 20h
00003E0F 480500100000            add rax, 1000h
00003E15 48C1C820                ror rax, 20h
00003E19 AB                      stosd
00003E1A 5F                      pop rdi
```

```
8814 00003E1B 58                            pop rax
8815 00003E1C C3                            ret
8816
8817
8818                                    .ehciDevSetupHub:
8819                                    ;Device specific setup. Takes rsi as a ptr to the
8820                                    ; specific device parameter block.
8821 00003E1D 53                            push rbx
8822 00003E1E 51                            push rcx
8823 00003E1F 56                            push rsi
8824 00003E20 55                            push rbp
8825 00003E21 6631ED                        xor bp, bp     ;Error Stage 0
8826                                    .edshub:
8827 00003E24 E839030000                    call .ehciHubClassDescriptor
8828 00003E29 0F82DA000000                  jc .edsfail
8829 00003E2F 66FFC5                        inc bp  ;Error Stage 1
8830 00003E32 8A4E05                        mov cl, byte [rsi + 5]  ;Get number of ports here
8831 00003E35 B201                          mov dl, 1    ;Start port number to begin enum on (hub ports
                                                                 start at 1)
8832                                    .edshub1:
8833 00003E37 49BC03000000000000-           mov r12, 3
8833 00003E40 00
8834                                    .edshub11:
8835 00003E41 E8CB000000                    call .ehciEnumerateHubPort     ;dl for port to scan/enumerate
8836 00003E46 7413                          jz .edshub13     ;If ZF=ZR, valid device found!
8837 00003E48 803C25[A9010000]20            cmp byte [msdStatus], 20h  ;General Controller Failure
8838 00003E50 0F8448F3FFFF                  je USB.ehciCriticalErrorWrapper
8839 00003E56 49FFCC                        dec r12
8840 00003E59 75E6                          jnz .edshub11   ;Still not zero but failed, try again.
8841                                    .edshub13:
8842 00003E5B FEC2                          inc dl  ;Start with port 1
8843 00003E5D 38D1                          cmp cl, dl
8844 00003E5F 73D6                          jae .edshub1
8845                                    .edshub2:
8846                                    ;Need to write bHostHub for any detected devices here
8847 00003E61 F8                            clc     ;Common success exit
8848 00003E62 5D                            pop rbp
8849 00003E63 5E                            pop rsi
8850 00003E64 59                            pop rcx
8851 00003E65 5B                            pop rbx
8852 00003E66 C3                            ret
8853
8854                                    .ehciDeviceSetupMsd:
8855                                    ; Input:  rsi = MSD Device Parameter Block
8856                                    ; Output: CF=CY if catastrophic host error.
8857                                    ;         CF=NC then ax = Return code
8858                                    ;         ax = 0 if successful setup
8859                                    ;         ax = 1 if device did not reset the first time
8860                                    ;         ax = 2 if device did not return a valid LUN
8861                                    ;         ax = 3 if device did not reset the second time
8862                                    ;         Device must me removed from tables and port reset if ax
                                                                 != 0
8863 00003E67 51                            push rcx
8864 00003E68 55                            push rbp
8865 00003E69 4150                          push r8
8866 00003E6B 6631ED                        xor bp, bp     ;Error Stage 0
8867                                    .edsmsd:
8868 00003E6E 49B810000000000000-           mov r8, 10h ;Loop counter setup
8868 00003E77 00
8869                                    .edsm1:
8870 00003E78 E8BD030000                    call .ehciMsdDeviceReset
8871 00003E7D 0F8281000000                  jc .edsexit
8872                                    ;Check eActiveInterrupt for confirmation if we need to handle error
8873 00003E83 F60425[48020000]02            test byte [eActiveInt], 2    ;If this is set, handle error
8874 00003E8B 7416                          jz .edsms2
8875 00003E8D 668B4E07                      mov cx, word [rsi + 7]     ;Pass endpoint size
8876 00003E91 8A06                          mov al, byte [rsi]  ;Device address
8877 00003E93 E862FEFFFF                    call .ehciStandardErrorHandler
8878 00003E98 A880                          test al, 80h
8879 00003E9A 756D                          jnz .edsfail    ;If bit 7 is set, something is seriously wrong,
                                                                 fail dev!
8880 00003E9C 49FFC8                        dec r8                  ;Dec loop counter
8881 00003E9F 7468                          jz .edsfail             ;Fatal error if after 16 goes nothing was
                                                                 resolved
8882 00003EA1 EBD5                          jmp short .edsm1
8883                                    .edsms2:
8884 00003EA3 66FFC5                        inc bp  ;Error Stage 1
8885                                    .edsms3:
8886 00003EA6 E8B9030000                    call .ehciMsdGetMaxLun  ;If stall, clear endpoint and proceed.
                                                                 No loop
8887 00003EAB 7257                          jc .edsexit
8888 00003EAD F60425[48020000]02            test byte [eActiveInt], 2    ;If this is set, handle error
8889 00003EB5 740F                          jz .edsms4
```

```

00003EB7 668B4E07                   mov cx, word [rsi + 7]     ;Pass endpoint size
00003EBB 8A06                       mov al, byte [rsi]  ;Device address
00003EBD E838FEFFFF                 call .ehciStandardErrorHandler
00003EC2 A880                       test al, 80h
00003EC4 7543                       jnz .edsfail   ;If bit 7 is set, something is seriously wrong,
                                                         fail dev!
                                .edsms4:
00003EC6 66FFC5                     inc bp  ;Error Stage 2
00003EC9 49B810000000000000-        mov r8, 10h ;Loop counter setup
00003ED2 00

                                .edsms5:
00003ED3 E862030000                 call .ehciMsdDeviceReset  ;Reset once again to clear issues
00003ED8 722A                       jc .edsexit
00003EDA F60425[48020000]02         test byte [eActiveInt], 2   ;If this is set, handle error
00003EE2 7416                       jz .edsms6

00003EE4 668B4E07                   mov cx, word [rsi + 7]     ;Pass endpoint size
00003EE8 8A06                       mov al, byte [rsi]  ;Device address
00003EEA E80BFEFFFF                 call .ehciStandardErrorHandler
00003EEF A880                       test al, 80h
00003EF1 7516                       jnz .edsfail   ;If bit 7 is set, something is seriously wrong,
                                                         fail dev!
00003EF3 49FFC8                     dec r8                ;Dec loop counter
00003EF6 7411                       jz .edsfail            ;Fatal error if after 16 goes nothing was
                                                         resolved
00003EF8 EBD9                       jmp short .edsms5
                                .edsms6:
00003EFA FE0425[4B020000]           inc byte [numMSD]
00003F01 6631C0                     xor ax, ax  ;Note that xor also clears CF
                                .edsexit:
00003F04 4158                       pop r8
00003F06 5D                         pop rbp
00003F07 59                         pop rcx
00003F08 C3                         ret
                                .edsfail:
                                ;If a fail occurs, then the entry needs to be removed from the data
                                                         tables
00003F09 6689E8                     mov ax, bp
00003F0C E9F3FFFFFF                 jmp .edsexit

                                .ehciEnumerateHubPort:
                                ;Enumerates devices on an external Hub.
                                ;Use rsi to get device properties
                                ;Input: rsi = ptr to hub device block
                                ;        dl = Port number to reset
                                ;Output: None, CF

00003F11 53                         push rbx
00003F12 51                         push rcx
00003F13 52                         push rdx
00003F14 55                         push rbp
00003F15 4150                       push r8
00003F17 4151                       push r9
00003F19 4152                       push r10
00003F1B 4153                       push r11

00003F1D 0FB6D2                     movzx edx, dl
00003F20 48C1E220                   shl rdx, 4*8     ;Shift port number to right bits
                                .eehdeinit:
00003F24 6631ED                     xor bp, bp  ;Error counter
00003F27 4C0FB70E                   movzx r9, word [rsi]         ;Save hub bus/addr in r9w
00003F2B 4C0FB64604                 movzx r8, byte [rsi + 4]     ;Get MaxPacketSize0

                                .eehde0:
00003F30 48BB23030800000000-        mov rbx, 0000000000080323h  ;Set port power feature
00003F39 00

00003F3A 4809D3                     or rbx, rdx ;Add port number into descriptor
00003F3D 664489C1                   mov cx, r8w
00003F41 4488C8                     mov al, r9b
00003F44 E8C3F5FFFF                 call .ehciSetNoData    ;Turn on power to port on device in addr
                                                         al
00003F49 0F829F010000               jc .eehdecritical  ;Fast exit with carry set
                                .eehde1:
                                ;Power on debounce!
00003F4F B486                       mov ah, 86h
00003F51 0FB64E06                   movzx ecx, byte [rsi + 6]    ;poweron2powergood
00003F55 D1E1                       shl ecx, 1
00003F57 CD35                       int 35h

00003F59 66FFC5                     inc bp       ;Increment Error Counter    (Stage 1)
                                .eehde2:
00003F5C 48BB23011000000000-        mov rbx, 0000000000100123h  ;Clear port set connection bit
```

```
8965 00003F65 00
8966 00003F66 4809D3                      or rbx, rdx ;Add port number into descriptor
8967 00003F69 4488C1                      mov cl, r8b
8968 00003F6C 4488C8                      mov al, r9b
8969 00003F6F E898F5FFFF                  call .ehciSetNoData
8970 00003F74 0F8274010000                jc .eehdecritical  ;Fast exit with carry set
8971                                  .eehde3:
8972
8973 00003F7A 66FFC5                      inc bp        ;Increment Error Counter    (Stage 2)
8974                                  .eehde31:
8975 00003F7D 48BBA3000000000004-         mov rbx, 00040000000000A3h ;Get port status
8975 00003F86 00
8976 00003F87 4809D3                      or rbx, rdx
8977 00003F8A 4488C1                      mov cl, r8b
8978 00003F8D 4488C8                      mov al, r9b
8979 00003F90 E816F6FFFF                  call .ehciGetRequest
8980 00003F95 0F8253010000                jc .eehdecritical  ;Fast exit with carry set
8981                                  .eehde4:
8982 00003F9B 66FFC5                      inc bp        ;Increment Error Counter    (Stage 3)
8983
8984 00003F9E 8A0C25[C0030000]            mov cl, byte [ehciDataIn]    ;Get the first byte in into cx
8985 00003FA5 F6C101                      test cl, 1  ;Check device in port
8986 00003FA8 0F8436010000                jz .eehdebadnotimeout
8987
8988                                  .eehde41:   ;EP for first port reset state
8989 00003FAE 66FFC5                      inc bp        ;Increment Error Counter    (Stage 4)
8990 00003FB1 E83D010000                  call .eehdereset     ;First port reset
8991 00003FB6 0F8232010000                jc .eehdecritical  ;Fast exit with carry set
8992
8993 00003FBC 66FFC5                      inc bp        ;Increment Error Counter    (Stage 5)
8994
8995 00003FBF 49BB10000000000000-         mov r11, 10h
8995 00003FC8 00
8996                                  .eehde5:
8997 00003FC9 48BBA3000000000004-         mov rbx, 00040000000000A3h ;Get port status again
8997 00003FD2 00
8998 00003FD3 4809D3                      or rbx, rdx
8999 00003FD6 4488C1                      mov cl, r8b
9000 00003FD9 4488C8                      mov al, r9b
9001 00003FDC E8CAF5FFFF                  call .ehciGetRequest
9002 00003FE1 0F8207010000                jc .eehdecritical  ;Fast exit with carry set
9003                                  .eehde6:
9004 00003FE7 66FFC5                      inc bp        ;Increment Error Counter    (Stage 6)
9005                                  ;Now check for high speed
9006
9007 00003FEA 668B0C25[C0030000]          mov cx, word [ehciDataIn]
9008 00003FF2 6681E1FF07                  and cx, 7FFh     ;Zero upper bits
9009 00003FF7 66C1E909                    shr cx, 9    ;Bring bits [10:9] low
9010 00003FFB 6681F90200                  cmp cx, 2    ;2 is High Speed device
9011 00004000 0F85DE000000                jne .eehdebadnotimeout
9012 00004006 48C70425[C0030000]-         mov qword [ehciDataIn], 0
9012 0000400E 00000000
9013
9014 00004012 66FFC5                      inc bp        ;Increment Error Counter    (Stage 7)
9015
9016 00004015 57                          push rdi
9017 00004016 48BF-                       mov rdi, ehciDataIn
9017 00004018 [C003000000000000]
9018 00004020 B908000000                  mov ecx, 8
9019 00004025 31C0                        xor eax, eax
9020 00004027 F348AB                      rep stosq
9021 0000402A 5F                          pop rdi
9022                                  .eehde7:
9023 0000402B 48BB23011200000000-         mov rbx, 0000000000120123h   ;Clear port suspend
9023 00004034 00
9024 00004035 4809D3                      or rbx, rdx ;Add port number into descriptor
9025 00004038 4488C1                      mov cl, r8b
9026 0000403B 4488C8                      mov al, r9b
9027 0000403E E8C9F4FFFF                  call .ehciSetNoData
9028 00004043 0F82A5000000                jc .eehdecritical  ;Fast exit with carry set
9029
9030                                  .eehde10:
9031 00004049 48BB80060001000008-         mov rbx, 0000800000100068h0h      ;Pass get minimal device
                                                                         descriptor
9031 00004052 00
9032 00004053 66B94000                    mov cx, 40h     ;Pass default endpoint size
9033 00004057 30C0                        xor al, al
9034 00004059 E84DF5FFFF                  call .ehciGetRequest
9035 0000405E 0F828A000000                jc .eehdecritical  ;Fast exit with carry set
9036                                  .eehde101:
9037 00004064 66FFC5                      inc bp        ;Increment Error Counter    (Stage 8)
9038
9039 00004067 803C25[C1030000]01          cmp byte [ehciDataIn + 1], 01h     ;Verify this is a valid dev
```

```
                                                                descriptor
0000406F 756E                            jne .eehdebad          ;ehciDataIn contains error signature

                                     ;Sanity check the returned descriptor here
                                     .eehde11:
00004071 66813C25[C2030000]-             cmp word [ehciDataIn + 2], 0200h     ;Verify this is a USB 2.0+
                                                                device or
00004079 0002
0000407B 7262                            jb .eehdebad
0000407D 803C25[C4030000]00              cmp byte [ehciDataIn + 4], 0     ;Check interfaces
00004085 7414                            je .eehde12
00004087 803C25[C4030000]08              cmp byte [ehciDataIn + 4], 08h     ;MSD?
0000408F 740A                            je .eehde12
00004091 803C25[C4030000]09              cmp byte [ehciDataIn + 4], 09h     ;Hub?
00004099 7544                            jne .eehdebad

                                     .eehde12:     ;Valid device detected
0000409B 440FB60425-                     movzx r8d, byte [ehciDataIn + 7]     ;Save attached device max ep
                                                                size
000040A0 [C7030000]
                                     .eehde13:
000040A4 E84A000000                      call .eehdereset     ;Do second reset
000040A9 7243                            jc .eehdecritical   ;Fast exit with carry set
                                     ;Clear the data in buffer
000040AB 57                              push rdi
000040AC 48BF-                           mov rdi, ehciDataIn
000040AE [C003000000000000]
000040B6 B908000000                      mov ecx, 8
000040BB 31C0                            xor eax, eax
000040BD F348AB                          rep stosq
000040C0 5F                              pop rdi

                                     ;Device on port now ready to have an address set to it, and be
                                                                enumerated
000040C1 48C1EA20                        shr rdx, 4*8     ;Shift port number back down to dl
000040C5 668B06                          mov ax, word [rsi]   ;Get hub bus/addr pair
000040C8 88E6                            mov dh, ah           ;Move the bus number into dh
000040CA 440FB6D0                        movzx r10d, al       ;Move hub address into r10b
                                     ;Ensure dl=port number - 1, dh=Root hub (Bus) number, r10b=Host hub
                                                                number
                                     ;          r8b=Max Control EP endpoint size
000040CE 49BB64000000000000-             mov r11, 100     ;Address settle time
000040D7 00
000040D8 FECA                            dec dl
000040DA E998F7FFFF                      jmp .ehciEnumCommonEp

                                     .eehdebad:
                                     .eehdebadnoport:     ;EP if done without disabling port
000040DF E987F8FFFF                      jmp .ehciedbadnoport
                                     .eehdebadnotimeout:
000040E4 E98DF8FFFF                      jmp .ehciedbadnotimeout
                                     .eehdebadremtables:
000040E9 E99AF8FFFF                      jmp .ehcibadremtables
                                     .eehdecritical:
000040EE E96BF8FFFF                      jmp .ehciedexit   ;Fast exit with carry set
                                     .eehdereset:
                                     ;rsi must point to valid Hub device block
000040F3 48BB23030400000000-             mov rbx, 0000000000040323h  ;Reset port
000040FC 00
000040FD 4809D3                          or rbx, rdx ;Add device address
00004100 4488C1                          mov cl, r8b
00004103 4488C8                          mov al, r9b
00004106 E801F4FFFF                      call .ehciSetNoData
0000410B 7254                            jc .eehcritexit

0000410D 49BB8813000000000000-           mov r11, 5000 ;Just keep trying
00004116 00
                                     .eehder1:
00004117 B486                            mov ah, 86h
00004119 B914000000                      mov ecx, 20      ;20 ms is max according to USB 2.0 standard
0000411E CD35                            int 35h

00004120 48BBA300000000000004-           mov rbx, 00040000000000A3h ;Get port status
00004129 00
0000412A 4809D3                          or rbx, rdx
0000412D 4488C1                          mov cl, r8b
00004130 4488C8                          mov al, r9b
00004133 E873F4FFFF                      call .ehciGetRequest
00004138 8A0C25[C0030000]                mov cl, byte [ehciDataIn]    ;Get low byte of in data
0000413F F6C110                          test cl, 10h     ;If bit not set, reset over, proceed
00004142 7405                            jz .eehder2
00004144 49FFCB                          dec r11
00004147 75CE                            jnz .eehder1
```

```
                                       .eehder2:
00004149 48BB23011400000000-                mov rbx, 0000000000140123h ;Clear port reset bit
00004152 00
00004153 4809D3                             or rbx, rdx
00004156 4488C1                             mov cl, r8b
00004159 4488C8                             mov al, r9b
0000415C E8ABF3FFFF                         call .ehciSetNoData
                                       .eehcritexit:
00004161 C3                                 ret

                                       .ehciHubClassDescriptor:
                                       ;Gets the Hub class descriptor
                                       ;Get Hub descriptor for device pointed to by rsi
                                       ;If invalid data, returns error
                                       ;Input: rsi = Ptr to hub data block
                                       ;Output:
                                       ;     Carry Clear if success
                                       ;     Carry Set if fail, al contains error code
00004162 53                                 push rbx
00004163 51                                 push rcx
00004164 55                                 push rbp
00004165 66BD0300                           mov bp, 3

00004169 48BBA0060029000007-                mov rbx, 00070000290006A0h  ;Get Hub descriptor (only first 7
                                                                          bytes)
00004172 00
00004173 0FB64E04                           movzx ecx, byte [rsi + 4]   ;bMaxPacketSize0
00004177 8A06                               mov al, byte [rsi]          ;Get device address
00004179 E82DF4FFFF                         call .ehciGetRequest
0000417E 7226                               jc .ehcdfail        ;Errors 0-2 live here

00004180 66FFC5                             inc bp
00004183 803C25[C1030000]29                 cmp byte [ehciDataIn + 1], 29h  ;Is this a valid hub descriptor
0000418B 7519                               jne .ehcdfail

0000418D 8A0C25[C2030000]                   mov cl, byte [ehciDataIn + 2]    ;Get number of downstream ports
00004194 884E05                             mov byte [rsi + 5], cl   ;Store in variable, marking device as
                                                                      configured

00004197 8A0C25[C5030000]                   mov cl, byte [ehciDataIn + 5]    ;Get PowerOn2PowerGood
0000419E 884E06                             mov byte [rsi + 6], cl   ;Store in variable
000041A1 F8                                 clc
                                       .ehcdexit:
000041A2 5D                                 pop rbp
000041A3 59                                 pop rcx
000041A4 5B                                 pop rbx
000041A5 C3                                 ret
                                       .ehcdfail:
000041A6 4088E8                             mov al, bpl
000041A9 F9                                 stc
000041AA EBF6                               jmp short .ehcdexit
                                       ;                    ----------MSD functions----------
                                       .ehciMsdInitialise:
                                       ;Initialises an MSD device.
                                       ;Input: rsi = Valid MSD device block
                                       ;Output: CF=CY: Init did not complete
                                       ;          al = 0 => Device initialised
                                       ;          al = 1 => Host/Schedule error
                                       ;          al = 2 => Device failed to initialise
                                       ;          CF=NC: Init complete, rsi points to complete USB MSD
                                                                      device block
000041AC 51                                 push rcx
000041AD 8A4601                             mov al, byte [rsi + 1]   ;Get the bus number into al
000041B0 E800F2FFFF                         call .ehciAdjustAsyncSchedCtrlr
000041B5 B001                               mov al, 1
000041B7 7249                               jc .ehciMsdInitFail
000041B9 E8A9FCFFFF                         call .ehciDeviceSetupMsd
000041BE B002                               mov al, 2
000041C0 7240                               jc .ehciMsdInitFail
000041C2 E8EB040000                         call .ehciMsdBOTInquiry
000041C7 7239                               jc .ehciMsdInitFail
000041C9 B905000000                         mov ecx, 5
                                       .emi0:
000041CE E82C050000                         call .ehciMsdBOTReadFormatCapacities
000041D3 803C25[A9010000]20                 cmp byte [msdStatus], 20h    ;Host error
000041DB 7439                               je .ehciMsdInitialisePfail   ;Protocol fail
000041DD E88C010000                         call .ehciMsdBOTCheckTransaction
000041E2 6685C0                             test ax, ax
000041E5 7538                               jnz .emipf0
000041E7 E8F6060000                         call .ehciMsdBOTModeSense6
000041EC 803C25[A9010000]20                 cmp byte [msdStatus], 20h    ;Host error
000041F4 7420                               je .ehciMsdInitialisePfail   ;Protocol fail
000041F6 E873010000                         call .ehciMsdBOTCheckTransaction
```

```
000041FB 6685C0                     test ax, ax       ;Also clears CF if zero
000041FE 751F                       jnz .emipf0
                                .ehciMsdInitExit:
00004200 59                         pop rcx
00004201 C3                         ret
                                .ehciMsdInitFail:
00004202 668B06                     mov ax, word [rsi]
00004205 E807F9FFFF                 call .ehciRemoveDevFromTables
0000420A FE0C25[4B020000]           dec byte [numMSD]     ;Device was removed from tables, decrement
00004211 F9                         stc
00004212 B002                       mov al, 2
00004214 EBEA                       jmp short .ehciMsdInitExit
                                .ehciMsdInitialisePfail:
00004216 E873000000                 call .ehciMsdBOTResetRecovery
0000421B FFC9                       dec ecx
0000421D 74E3                       jz .ehciMsdInitFail
                                .emipf0:
0000421F E836060000                 call .ehciMsdBOTRequestSense
00004224 803C25[A9010000]20         cmp byte [msdStatus], 20h
0000422C 74E8                       je .ehciMsdInitialisePfail
0000422E E83B010000                 call .ehciMsdBOTCheckTransaction
00004233 6685C0                     test ax, ax
00004236 7496                       jz .emi0
00004238 EBDC                       jmp short .ehciMsdInitialisePfail


                                .ehciMsdDeviceReset:
                                ;Reset an MSD device on current active EHCI bus
                                ;Input: rsi = Pointer to table data structure
                                ;Output:
                                ;    CF=CY: Host error, Reset host system
                                ;    CF=NC: Transaction succeeded, check reset occurred successfully
                                ;             (If eActiveIntr AND 2 != 0, then error in transfer)
0000423A 51                         push rcx
0000423B 52                         push rdx
0000423C 53                         push rbx
0000423D 50                         push rax

0000423E 0FB64E07                   movzx ecx, byte [rsi + 7]   ;Get bMaxPacketSize0
00004242 480FB65604                 movzx rdx, byte [rsi + 4]   ;Get Interface Number
00004247 48C1E228                   shl rdx, 5*8 ;Send to 5th byte
0000424B 48BB21FF0000000000-        mov rbx, 0FF21h             ;MSD Reset
00004254 00
00004255 4809D3                     or rbx, rdx                 ;And those bytes
00004258 8A06                       mov al, byte [rsi]
0000425A E8ADF2FFFF                 call .ehciSetNoData

0000425F 58                         pop rax
00004260 5B                         pop rbx
00004261 5A                         pop rdx
00004262 59                         pop rcx
00004263 C3                         ret

                                .ehciMsdGetMaxLun:
                                ;Get max LUN of an MSD device on current active EHCI bus
                                ;Input: rsi = Pointer to table data structure
                                ;         al = Address
                                ;Output:
                                ;    CF=CY: Host error, Reset host system
                                ;    CF=NC: Transaction succeeded, check data transferred
                                                                        successfully
                                ;    Max Lun saved at DataIn Buffer (first byte)
                                ;    Check this was transferred, by checking total data transferred
                                                                        value
00004264 51                         push rcx
00004265 52                         push rdx
00004266 53                         push rbx
00004267 50                         push rax

00004268 0FB64E07                   movzx ecx, byte [rsi + 7]   ;Get bMaxPacketSize0
0000426C 480FB65604                 movzx rdx, byte [rsi + 4]   ;Get Interface Number
00004271 48C1E228                   shl rdx, 5*8 ;Send to 5th byte
00004275 48BBA1FE0000000001-        mov rbx, 000100000000FEA1h              ;MSD Get Max LUN
0000427E 00
0000427F 4809D3                     or rbx, rdx                 ;And those bytes
00004282 8A06                       mov al, byte [rsi]
00004284 E822F3FFFF                 call .ehciGetRequest

00004289 58                         pop rax
0000428A 5B                         pop rbx
0000428B 5A                         pop rdx
0000428C 59                         pop rcx
0000428D C3                         ret

```

```
                                        .ehciMsdBOTResetRecovery:
                                        ;------------------------------------------------------------------
                                        ;Calls the reset recovery procedure on a device ptd to by rsi    :
                                        ;Input:  rsi = Pointer to MSD device parameter block              :
                                        ;Output: CF=CY if something went wrong. Else CF=NC                :
                                        ;------------------------------------------------------------------
                                        ;Calls an MSDBBB reset then calls StandardErrorHandler AFTER      :
                                        ; writing the Qhead for each Bulk EP.                             :
                                        ;------------------------------------------------------------------
0000428E 50                                 push rax
0000428F 53                                 push rbx
00004290 51                                 push rcx
00004291 66C7460E0000                       mov word [rsi + 14], 00h      ;Reset clear both endpoint dt bits

00004297 E89EFFFFFF                         call .ehciMsdDeviceReset      ;Call the device reset
0000429C 721E                               jc .embrrexit
                                        ;Now clear stall on IN EP
0000429E 8A06                               mov al, byte [rsi]            ;Get the address
000042A0 8A5E08                             mov bl, byte [rsi + 8]        ;Get the 4 byte EP address
000042A3 0FB64E07                           movzx ecx, byte [rsi + 7]     ;Get the Max packet size for the
                                                                          ctrl EP
000042A7 E8B9FAFFFF                         call .ehciClearEpStallHalt
000042AC 720E                               jc .embrrexit
                                        ;Now clear stall on OUT EP
000042AE 8A06                               mov al, byte [rsi]            ;Get the address
000042B0 8A5E0B                             mov bl, byte [rsi + 11]       ;Get the 4 byte EP address
000042B3 0FB64E07                           movzx ecx, byte [rsi + 7]     ;Get the Max packet size for the
                                                                          ctrl EP
000042B7 E8A9FAFFFF                         call .ehciClearEpStallHalt
                                        .embrrexit:
000042BC 59                                 pop rcx
000042BD 5B                                 pop rbx
000042BE 58                                 pop rax
000042BF C3                                 ret
                                        .ehciMsdBOTCheckValidCSW:
                                        ; This function checks that the recieved CSW was valid.
                                        ; If this function returns a non-zero value in al,
                                        ; a reset recovery of the device is required
                                        ; Output: al = 0 : valid CSW
                                        ;         If CSW not valid, al contains a bitfield describing what
                                                                          failed
                                        ;         al = 1h   : CSW is not 13 bytes in length
                                        ;         al = 2h   : dCSWSignature is not equal to 053425355h
                                        ;         al = 4h   : dCSWTag does not match the dCBWTag
                                        ;         al = 0F8h : Reserved
                                        ;   rax destroyed
000042C0 53                                 push rbx
000042C1 51                                 push rcx
000042C2 31C0                               xor eax, eax
000042C4 66B90100                           mov cx, 1
000042C8 668B1C25[8A010000]                 mov bx, word [ehciTDSpace + 2*ehciSizeOfTD + 0Ah]
                                        ;Get total bytes to transfer from third QHeadTD to see if 13h bytes
                                                                          were
                                        ; transferred
000042D0 6681E3FF7F                         and bx, 7FFFh   ;Clear upper bit
000042D5 660F45C1                           cmovnz ax, cx   ;If the result for the and is not zero, <>13
                                                                          bytes were sent


000042D9 66D1E1                             shl cx, 1
000042DC 6609C1                             or cx, ax
000042DF 813C25[C0050000]55-                cmp dword [msdCSW], CSWSig
000042E7 534253
000042EA 660F45C1                           cmovne ax, cx

000042EE 66B90400                           mov cx, 4h
000042F2 6609C1                             or cx, ax
000042F5 0FB61C25[4A020000]                 movzx ebx, byte [cbwTag]
000042FD FFCB                               dec ebx
000042FF 3B1C25[C4050000]                   cmp ebx, dword [msdCSW + 4h]
00004306 660F45C1                           cmovne ax, cx

0000430A 59                                 pop rcx
0000430B 5B                                 pop rbx
0000430C C3                                 ret

                                        .ehciMsdBOTCheckMeaningfulCSW:
                                        ; This function checks if the CSW was meaningful.
                                        ; If this function returns a non-zero value in al, it is up to the
                                        ; caller to decide what action to take. The possible set of actions
                                                                          that
                                        ; can be taken is outlined in Section 6.7 of the USB MSC BOT
                                                                          Revision 1.0
                                        ; specification.
```

```
                                 ; Output :   al = 0h   : Invalid
                                 ;            al = 1h   : bCSWStatus = 0
                                 ;            al = 2h   : bCSWStatus = 1
                                 ;            al = 4h   : bCSWStatus = 2
                                 ;            al = 8h   : bCSWStatus > 2
                                 ;            al = 10h  : dCSWDataResidue = 0
                                 ;            al = 20h  : dCSWDataResidue < dCBWDataTransferLength
                                 ;            al = 40h  : dCSWDataResidue > dCBWDataTransferLength
                                 ;            al = 80h  : Reserved
                                 ;    rax destroyed
0000430D 53                          push rbx
0000430E 51                          push rcx

0000430F 31C0                        xor eax, eax   ;In the event that things go completely wrong
00004311 66BB0800                    mov bx, 8h
00004315 8A0C25[CC050000]            mov cl, byte [msdCSW + 0Ch]

0000431C 80F902                      cmp cl, 2
0000431F 660F47C3                    cmova ax, bx
00004323 7718                        ja .embcmcResidueCheck

00004325 66D1EB                      shr bx, 1          ;Shift it down to 4
00004328 660F44C3                    cmove ax, bx       ;If bCSWStatus = 2, move it in
0000432C 740F                        je .embcmcResidueCheck

0000432E 66D1EB                      shr bx, 1          ;Shift down to 2
00004331 80F901                      cmp cl, 1
00004334 660F44C3                    cmove ax, bx       ;If bCSWStatus = 1, move bx into ax
00004338 7403                        je .embcmcResidueCheck

0000433A 66FFC0                      inc ax             ;Otherwise bCSWStatus = 0
                                 .embcmcResidueCheck:
0000433D 8B0C25[C8050000]            mov ecx, dword [msdCSW + 8]  ;Get dCSWDataResidue

00004344 66BB1000                    mov bx, 10h
00004348 6609C3                      or bx, ax
0000434B 85C9                        test ecx, ecx
0000434D 660F44C3                    cmovz ax, bx       ;If its zero, move bx with added bit from ax
                                                             into ax
00004351 7418                        jz .embcmcExit

00004353 66BB2000                    mov bx, 20h
00004357 6609C3                      or bx, ax
0000435A 3B0C25[88030000]            cmp ecx, dword [ehciDataOut + 8];ehciDataOut + 8 =
                                                             dCBWDataTransferLength
00004361 660F42C3                    cmovb ax, bx
00004365 7204                        jb .embcmcExit

00004367 660D4000                    or ax, 40h  ;Else, it must be above, fail
                                 .embcmcExit:
0000436B 59                          pop rcx
0000436C 5B                          pop rbx
0000436D C3                          ret

                                 .ehciMsdBOTCheckTransaction:
                                 ;Check successful return data here
                                 ;Output: ax = 0                                  : CSW Valid and
                                                             Meaningful
                                 ;          ah = 1, al = CSW Validity bitfield      : CSW NOT valid
                                 ;          ah = 2, al = CSW Meaningful bitfield    : CSW NOT meaningful
                                 ;    rax destroyed
0000436E 30E4                        xor ah, ah
00004370 E84BFFFFFF                  call .ehciMsdBOTCheckValidCSW
00004375 84C0                        test al, al
00004377 7407                        jz .embhiehcswmeaningful
00004379 B401                        mov ah, 1          ; CSW Not Valid signature
0000437B E90B000000                  jmp .embhiehexit
                                 .embhiehcswmeaningful:
00004380 E888FFFFFF                  call .ehciMsdBOTCheckMeaningfulCSW
00004385 244C                        and al, 4Ch        ;Check bad bits first and bCSWStatus=02
                                                             40h|08h|04h
00004387 7402                        jz .embhiehexit
00004389 B402                        mov ah, 2          ; CSW Not Meaningful signature
                                 .embhiehexit:
0000438B C3                          ret
                                 .ehciMsdBOTOO64I:     ;For devices with 64 byte max packet size
                                 .ehciMsdBOTOI64I:     ;For devices with 64 byte max packet size
0000438C C60425[A9010000]BB          mov byte [msdStatus], 0BBh    ;Undefined error
00004394 C3                          ret
                                 .ehciMsdBOTOOI:       ;Out Out In transfer
                                 ;Input − rsi = MSD device parameter block
                                 ;          rbx = Input buffer for Data In
                                 ;          ecx = Number of milliseconds to wait between Out and In
```

```
                                                                   packets
                                     ;          r8  = Number of bytes to be transferred (for the DATA
                                                                   phase)
                                     ;          r10 = LUN Value
                                     ;          r11 = Length of CBW command block
00004395 57                              push rdi
00004396 4150                            push r8
00004398 4151                            push r9
0000439A 4152                            push r10
0000439C 4153                            push r11
0000439E 4154                            push r12
000043A0 51                              push rcx
000043A1 FC                              cld

000043A2 4D89C4                          mov r12, r8      ;Save number of bytes to transfer to MSD device
000043A5 51                              push rcx
                                     ;Write QHead for CBW
000043A6 49BB-                           mov r11, ehciTDSpace ;First TD is the head of the Out buffer
000043A8 [0001000000000000]
000043B0 E840020000                      call .ehciMsdWriteOutQHead
                                     ;Write TD for CBW send
000043B5 4C89DF                          mov rdi, r11     ;Move pointer to TD buffer head
000043B8 49B801000000000000-             mov r8, 1
000043C1 00
000043C2 4D89C1                          mov r9, r8
000043C5 440FB6560F                      movzx r10d, byte [rsi + 15]    ;Get Out EP dt bit
000043CA 80760F01                        xor byte [rsi + 15], 1  ;Toggle bit
000043CE 41D1CA                          ror r10d, 1 ;Roll dt bit to upper bit of dword
000043D1 4181CA808C1F00                  or r10d, 001F8C80h
                                     ; Active TD, OUT EP, Error ctr = 3, 01Fh = 31 byte transfer
000043D8 49BB-                           mov r11, ehciDataOut ; Data out buffer
000043DA [8003000000000000]
000043E2 E8BAF9FFFF                      call .ehciWriteQHeadTD

000043E7 B103                            mov cl, 11b      ;Lock out internal buffer
000043E9 E8E6F2FFFF                      call .ehciProcessCommand          ;Run controller
000043EE 59                              pop rcx     ;Wait ecx ms for "motors to spin up"
000043EF 0F824B010000                    jc .emboexit      ;If catastrophic Host system error, exit!

000043F5 50                              push rax
000043F6 B486                            mov ah, 86h
000043F8 CD35                            int 35h
000043FA 58                              pop rax
                                     ;Write Qhead to Send data
000043FB 49BB80000000000000-             mov r11, ehciSizeOfTD + ehciSizeOfTD
00004404 00
00004405 E8EB010000                      call .ehciMsdWriteOutQHead
                                     ;Write TD for data send
0000440A 4C89DF                          mov rdi, r11
0000440D 49B801000000000000-             mov r8, 1
00004416 00
00004417 4D89C1                          mov r9, r8
0000441A 4D89E2                          mov r10, r12       ;Get back number of bytes to transfer
0000441D 49C1E210                        shl r10, 8*2      ;Shift into 3rd byte
00004421 4181CA808C0000                  or r10d, 00008C80h ;Add control bits: Active TD, OUT EP, Error
                                                                   ctr = 3
00004428 0FB64E0F                        movzx ecx, byte [rsi + 15]  ;Get Out EP dt bit in r9d
0000442C 80760F01                        xor byte [rsi + 15], 1  ;Toggle bit
00004430 D1C9                            ror ecx, 1 ;Roll dt bit to upper bit of dword
00004432 4109CA                          or r10d, ecx      ;Add dt bit to r10d
00004435 4989DB                          mov r11, rbx      ;Get the address of Data buffer
00004438 E864F9FFFF                      call .ehciWriteQHeadTD

0000443D B103                            mov cl, 11b      ;Lock out internal buffer
0000443F E890F2FFFF                      call .ehciProcessCommand          ;Run controller
00004444 0F82F6000000                    jc .emboexit      ;If catastrophic Host system error, exit!
                                     ;Write Qhead for CSW
0000444A 49BB-                           mov r11, ehciTDSpace + 2*ehciSizeOfTD  ;Third TD
0000444C [8001000000000000]
00004454 E8DB010000                      call .ehciMsdWriteInQHead
00004459 4C89DF                          mov rdi, r11
0000445C E9A9000000                      jmp .emboiicommonep
                                     .ehciMsdBOTOII: ;Out In In transfer
                                     ;Input - rsi = MSD device parameter block
                                     ;          rbx = Input buffer for Data In
                                     ;          ecx = Number of milliseconds to wait between Out and In
                                                                   packets
                                     ;          r8  = Number of bytes to be transferred (for the DATA
                                                                   phase)
                                     ;          r10 = LUN Value
                                     ;          r11 = Length of CBW command block

00004461 57                              push rdi
```

```
00004462 4150                          push r8
00004464 4151                          push r9
00004466 4152                          push r10
00004468 4153                          push r11
0000446A 4154                          push r12
0000446C 51                            push rcx
0000446D FC                            cld

0000446E 4D89C4                        mov r12, r8 ;Save the number of bytes to be transferred
00004471 51                            push rcx

                                   ;Write the OUT Queue Head
00004472 49BB-                         mov r11, ehciTDSpace ;First TD is the head of the Out buffer
00004474 [0001000000000000]
0000447C E874010000                    call .ehciMsdWriteOutQHead

00004481 4C89DF                        mov rdi, r11      ;Move pointer to TD buffer head
00004484 49B801000000000000-           mov r8, 1
0000448D 00
0000448E 4D89C1                        mov r9, r8
00004491 440FB6560F                    movzx r10d, byte [rsi + 15]    ;Get Out EP dt bit
00004496 80760F01                      xor byte [rsi + 15], 1  ;Toggle bit
0000449A 41D1CA                        ror r10d, 1 ;Roll dt bit to upper bit of dword
0000449D 4181CA808C1F00                or r10d, 001F8C80h
                                   ; Active TD, OUT EP, Error ctr = 3, 01Fh = 31 byte transfer
000044A4 49BB-                         mov r11, ehciDataOut ; Data out buffer
000044A6 [8003000000000000]
000044AE E8EEF8FFFF                    call .ehciWriteQHeadTD

000044B3 B103                          mov cl, 11b      ;Lock out internal buffer
000044B5 E81AF2FFFF                    call .ehciProcessCommand          ;Run controller
000044BA 59                            pop rcx      ;Wait ecx ms for "motors to spin up"
000044BB 0F827F000000                  jc .emboexit      ;If catastrophic Host system error, exit!

000044C1 50                            push rax
000044C2 B486                          mov ah, 86h
000044C4 CD35                          int 35h
000044C6 58                            pop rax
                                   ;Write the IN Queue Head
000044C7 49BB-                         mov r11, ehciTDSpace + ehciSizeOfTD ;Move to position 2 to
                                                                    preserve OUT TD
000044C9 [4001000000000000]
000044D1 E85E010000                    call .ehciMsdWriteInQHead

000044D6 4C89DF                        mov rdi, r11      ;Move pointer to TD buffer head
000044D9 4C8D4740                      lea r8, qword [rdi + ehciSizeOfTD]  ;Point to next TD
000044DD 4D89C1                        mov r9, r8
000044E0 4D89E2                        mov r10, r12       ;Get back number of bytes to transfer from the
                                                                    stack
000044E3 49C1E210                      shl r10, 8*2      ;Shift into 3rd byte
000044E7 4181CA800D0000                or r10d, 00000D80h ;Add control bits: Active TD, IN EP, Error
                                                                    ctr = 3
000044EE 0FB64E0E                      movzx ecx, byte [rsi + 14]   ;Get IN EP dt bit in r9d
000044F2 80760E01                      xor byte [rsi + 14], 1   ;Toggle bit
000044F6 D1C9                          ror ecx, 1 ;Roll dt bit to upper bit of dword
000044F8 4109CA                        or r10d, ecx      ;Add dt bit to r10d
000044FB 4989DB                        mov r11, rbx ; Data out buffer, default ehciDataIn
000044FE E89EF8FFFF                    call .ehciWriteQHeadTD

00004503 4881C740000000                add rdi, ehciSizeOfTD       ;Go to next TD space
                                   .emboiicommonep:
0000450A 49B801000000000000-           mov r8, 1
00004513 00
00004514 4D89C1                        mov r9, r8
00004517 41BA808D0D00                  mov r10d, 000D8D80h       ;Active TD, IN EP, Error ctr = 3, 0Dh =
                                                                    13 byte CSW
0000451D 0FB64E0E                      movzx ecx, byte [rsi + 14]   ;Get IN EP dt bit in r9d
00004521 80760E01                      xor byte [rsi + 14], 1   ;Toggle bit
00004525 D1C9                          ror ecx, 1 ;Roll dt bit to upper bit of dword
00004527 4109CA                        or r10d, ecx      ;Add dt bit to r10d
0000452A 49BB-                         mov r11, msdCSW
0000452C [C005000000000000]

00004534 E868F8FFFF                    call .ehciWriteQHeadTD

00004539 B103                          mov cl, 11b      ;Lock out internal buffer
0000453B E894F1FFFF                    call .ehciProcessCommand          ;Run controller
                                   .emboexit:
00004540 59                            pop rcx
00004541 415C                          pop r12
00004543 415B                          pop r11
00004545 415A                          pop r10
00004547 4159                          pop r9
```

```
00004549 4158                          pop r8
0000454B 5F                            pop rdi
0000454C C3                            ret
                                   .ehciMsdBOTOI: ;Out In transfer
                                   ;Input - rsi = MSD device parameter block
                                   ;        rbx = Input buffer for Data In
                                   ;        ecx = Number of milliseconds to wait between Out and In
                                                  packets
                                   ;        r8  = Number of bytes to be transferred (for the DATA
                                                  phase)
                                   ;        r10 = LUN Value
                                   ;        r11 = Length of CBW command block

0000454D 57                            push rdi
0000454E 4150                          push r8
00004550 4151                          push r9
00004552 4152                          push r10
00004554 4153                          push r11
00004556 51                            push rcx
00004557 FC                            cld


                                   ;Write the OUT Queue Head
00004558 49BB-                         mov r11, ehciTDSpace ;First TD is the head of the Out buffer
0000455A [0001000000000000]
00004562 E88E000000                    call .ehciMsdWriteOutQHead

00004567 4C89DF                        mov rdi, r11     ;Move pointer to TD buffer head
0000456A 49B801000000000000-           mov r8, 1
00004573 00
00004574 4D89C1                        mov r9, r8
00004577 440FB6560F                    movzx r10d, byte [rsi + 15]    ;Get Out EP dt bit
0000457C 80760F01                      xor byte [rsi + 15], 1   ;Toggle bit
00004580 41D1CA                        ror r10d, 1 ;Roll dt bit to upper bit of dword
00004583 4181CA808C1F00                or r10d, 001F8C80h
                                   ; Active TD, OUT EP, Error ctr = 3, 01Fh = 31 byte transfer
0000458A 49BB-                         mov r11, ehciDataOut ; Data out buffer
0000458C [8003000000000000]
00004594 E808F8FFFF                    call .ehciWriteQHeadTD

00004599 B103                          mov cl, 11b     ;Lock out internal buffer
0000459B E834F1FFFF                    call .ehciProcessCommand           ;Run controller
000045A0 7248                          jc .emboiexit      ;If catastrophic Host system error, exit!

                                   ;Write the IN Queue Head
000045A2 49BB-                         mov r11, ehciTDSpace + ehciSizeOfTD ;Move to position 2 to
                                                  preserve OUT TD
000045A4 [4001000000000000]
000045AC E883000000                    call .ehciMsdWriteInQHead

000045B1 4C89DF                        mov rdi, r11     ;Move pointer to TD buffer head
000045B4 49B801000000000000-           mov r8, 1
000045BD 00
000045BE 4D89C1                        mov r9, r8
000045C1 41BA808D0D00                  mov r10d, 000D8D80h      ;Active TD, IN EP, Error ctr = 3, 0Dh =
                                                  13 byte CSW
000045C7 0FB64E0E                      movzx ecx, byte [rsi + 14]  ;Get IN EP dt bit in r9d
000045CB 80760E01                      xor byte [rsi + 14], 1   ;Toggle bit
000045CF D1C9                          ror ecx, 1 ;Roll dt bit to upper bit of dword
000045D1 4109CA                        or r10d, ecx     ;Add dt bit to r10d
000045D4 49BB-                         mov r11, msdCSW
000045D6 [C005000000000000]

000045DE E8BEF7FFFF                    call .ehciWriteQHeadTD

000045E3 B103                          mov cl, 11b     ;Lock out internal buffer
000045E5 E8EAF0FFFF                    call .ehciProcessCommand           ;Run controller
                                   .emboiexit:
000045EA 59                            pop rcx
000045EB 415B                          pop r11
000045ED 415A                          pop r10
000045EF 4159                          pop r9
000045F1 4158                          pop r8
000045F3 5F                            pop rdi
000045F4 C3                            ret
                                   .ehciMsdWriteOutQHead:
                                   ;Input: rsi = Valid MSD device
                                   ;       r11 = Ptr to First QHTD
000045F5 E84CEEFFFF                    call .ehciGetNewQHeadAddr
000045FA 4181C802000000                or r8d, 2     ;Process QHs
00004601 41B900600000                  mov r9d, 00006000h  ;Default mask, no nak counter
00004607 0FB74E0C                      movzx ecx, word [rsi + 12]  ;wMaxPacketSizeOut
0000460B C1E110                        shl ecx, 8*2
```

```
0000460E 4109C9                       or r9d, ecx
00004611 0FB64E0B                     movzx ecx, byte [rsi + 11]  ;EP address
00004615 81E10F000000                 and ecx, 0Fh
0000461B C1E108                       shl ecx, 8  ;Shift to second byte
0000461E 4109C9                       or r9d, ecx ;Add bits
00004621 8A06                         mov al, byte [rsi]  ;Get device address
00004623 247F                         and al, 7Fh    ;Force clear upper bit of al
00004625 4108C1                       or r9b, al     ;Set lower 8 bits of r9 correctly
00004628 41BA00000040                 mov r10d, 40000000h    ;1 transaction/ms
0000462E E849F7FFFF                   call .ehciWriteQHead
00004633 C3                           ret
                                  .ehciMsdWriteInQHead:
                                  ;Input: rsi = Valid MSD device
                                  ;        r11 = Ptr to First QHTD
00004634 E80DEEFFFF                   call .ehciGetNewQHeadAddr
00004639 4981C802000000               or r8, 2
00004640 41B900600000                 mov r9d, 00006000h  ;Default mask
00004646 0FB74E09                     movzx ecx, word [rsi + 9]  ;wMaxPacketSizeIn
0000464A C1E110                       shl ecx, 8*2
0000464D 4109C9                       or r9d, ecx
00004650 0FB64E08                     movzx ecx, byte [rsi + 8]  ;EP address
00004654 81E10F000000                 and ecx, 0Fh
0000465A C1E108                       shl ecx, 8  ;Shift to second byte
0000465D 4109C9                       or r9d, ecx ;Add bits
00004660 8A06                         mov al, byte [rsi]  ;Get device address
00004662 247F                         and al, 7Fh    ;Force clear upper bit of al
00004664 4108C1                       or r9b, al     ;Set lower 8 bits of r9 correctly
00004667 41BA00000040                 mov r10d, 40000000h    ;1 transaction/ms
0000466D E80AF7FFFF                   call .ehciWriteQHead
00004672 C3                           ret
                                  .ehciMsdBOTRequest:
                                  ;Input: ecx = Number of miliseconds to wait between Out and In
                                                 requests
                                  ;        rbx = Data in Buffer
                                  ;        r8  = Number of bytes to be returned by command
                                  ;        r11 = Length of SCSI command block
                                  ;        r14 = Pointer to EHCI(USB) transaction function
                                  ;        r15 = Pointer to SCSI command function
                                  ;Output:
                                  ;    CF=CY: Host error, Reset host system
                                  ;    CF=NC: Transaction succeeded, check data transferred
                                                 successfully
00004673 50                           push rax
00004674 51                           push rcx
00004675 57                           push rdi
00004676 4151                         push r9
00004678 4152                         push r10
                                  ;Clear the previous CSW
0000467A 48BF-                        mov rdi, msdCSW
0000467C [C005000000000000]

00004684 30C0                         xor al, al
00004686 B90D000000                   mov ecx, 13
0000468B F3AA                         rep stosb
                                  ;Write the CBW
0000468D 48BF-                        mov rdi, ehciDataOut     ;Write the CBW at the data out point
0000468F [8003000000000000]


00004697 41B180                       mov r9b, 80h             ;Recieve an IN packet
0000469A 4D31D2                       xor r10, r10             ;LUN 0
0000469D E865030000                   call .msdWriteCBW        ;Write the 15 byte CBW
                                  ;Append the Command Block to the CBW
000046A2 30C0                         xor al, al               ;LUN 0 device
000046A4 41FFD7                       call r15                 ;Write the valid CBW Command block
                                  ;Enact transaction
000046A7 41FFD6                       call r14

000046AA 415A                         pop r10
000046AC 4159                         pop r9
000046AE 5F                           pop rdi
000046AF 59                           pop rcx
000046B0 58                           pop rax
000046B1 C3                           ret

                                  .ehciMsdBOTInquiry:
                                  ;Input:
                                  ; rsi = Pointer to MSD table data structure that we want to Inqure
                                  ;Output:
                                  ;    CF=CY: Host error, Reset host system
                                  ;    CF=NC: Transaction succeeded, check data transferred
                                                 successfully
000046B2 53                           push rbx
000046B3 51                           push rcx
000046B4 4150                         push r8
```

```
9718 000046B6 4153                          push r11
9719 000046B8 4156                          push r14
9720 000046BA 4157                          push r15
9721 000046BC 48BB-                         mov rbx, ehciDataIn
9721 000046BE [C003000000000000]
9722 000046C6 B900000000                    mov ecx, 0
9723 000046CB 41B824000000                  mov r8d, 024h            ;36 bytes to be returned
9724 000046D1 49BB0C000000000000-           mov r11, 0Ch             ;The command block is 12 bytes (As per
                                                                          Bootability)
9724 000046DA 00
9725 000046DB 49BF-                         mov r15, .scsiInquiry
9725 000046DD [384A000000000000]
9726 000046E5 49BE-                         mov r14, .ehciMsdBOTOII
9726 000046E7 [6144000000000000]
9727 000046EF E87FFFFFFF                    call .ehciMsdBOTRequest
9728 000046F4 415F                          pop r15
9729 000046F6 415E                          pop r14
9730 000046F8 415B                          pop r11
9731 000046FA 4158                          pop r8
9732 000046FC 59                            pop rcx
9733 000046FD 5B                            pop rbx
9734 000046FE C3                            ret
9735
9736                                    .ehciMsdBOTReadFormatCapacities:
9737                                    ;Input:
9738                                    ; rsi = Pointer to MSD table data structure
9739                                    ;Output:
9740                                    ;    CF=CY: Host error, Reset host system
9741                                    ;    CF=NC: Transaction succeeded, check data transferred
                                                                          successfully
9742 000046FF 53                            push rbx
9743 00004700 51                            push rcx
9744 00004701 4150                          push r8
9745 00004703 4153                          push r11
9746 00004705 4156                          push r14
9747 00004707 4157                          push r15
9748 00004709 48BB-                         mov rbx, ehciDataIn
9748 0000470B [C003000000000000]
9749 00004713 B900000000                    mov ecx, 0
9750 00004718 49B8FC000000000000-           mov r8, 0FCh             ;Return 252 bytes
9750 00004721 00
9751 00004722 49BB0A000000000000-           mov r11, 0Ah             ;The command block is 10 bytes
9751 0000472B 00
9752 0000472C 49BF-                         mov r15, .scsiReadFormatCapacities
9752 0000472E [974A000000000000]
9753 00004736 49BE-                         mov r14, .ehciMsdBOTOII
9753 00004738 [6144000000000000]
9754 00004740 E82EFFFFFF                    call .ehciMsdBOTRequest
9755 00004745 415F                          pop r15
9756 00004747 415E                          pop r14
9757 00004749 415B                          pop r11
9758 0000474B 4158                          pop r8
9759 0000474D 59                            pop rcx
9760 0000474E 5B                            pop rbx
9761 0000474F C3                            ret
9762
9763                                    .ehciMsdBOTReadCapacity10:
9764                                    ;Input:
9765                                    ; rsi = Pointer to MSD table data structure that we want to Read
                                                                          Capcities
9766                                    ;Output:
9767                                    ;    CF=CY: Host error, Reset host system
9768                                    ;    CF=NC: Transaction succeeded, check data transferred
                                                                          successfully
9769 00004750 53                            push rbx
9770 00004751 51                            push rcx
9771 00004752 4150                          push r8
9772 00004754 4153                          push r11
9773 00004756 4156                          push r14
9774 00004758 4157                          push r15
9775 0000475A 48BB-                         mov rbx, ehciDataIn
9775 0000475C [C003000000000000]
9776 00004764 B900000000                    mov ecx, 0
9777 00004769 49B808000000000000-           mov r8, 8
9777 00004772 00
9778 00004773 49BB0A000000000000-           mov r11, 0Ah
9778 0000477C 00
9779 0000477D 49BF-                         mov r15, .scsiReadCap10
9779 0000477F [A74A000000000000]
9780 00004787 49BE-                         mov r14, .ehciMsdBOTOII
9780 00004789 [6144000000000000]
9781 00004791 E8DDFEFFFF                    call .ehciMsdBOTRequest
9782 00004796 415F                          pop r15
```

```
00004798 415E                              pop r14
0000479A 415B                              pop r11
0000479C 4158                              pop r8
0000479E 59                                pop rcx
0000479F 5B                                pop rbx
000047A0 C3                                ret
                                       .ehciMsdBOTFormatUnit:
                                       ;Input:
                                       ; rsi = Pointer to MSD table data structure that we want to Format
                                       ;Output:
                                       ;    CF=CY: Host error, Reset host system
                                       ;    CF=NC: Transaction succeeded, check data transferred
                                                                  successfully
000047A1 50                                push rax
000047A2 4150                              push r8
000047A4 4153                              push r11
000047A6 4156                              push r14
000047A8 4157                              push r15
000047AA 4D31C0                            xor r8, r8   ;Request no data
000047AD 49BB06000000000000-               mov r11, 06h   ;Command length is 6 bytes
000047B6 00
000047B7 49BE-                             mov r14, .ehciMsdBOTOI
000047B9 [4D45000000000000]
000047C1 49BF-                             mov r15, .scsiFormatUnit
000047C3 [B14A000000000000]
000047CB E8A3FEFFFF                        call .ehciMsdBOTRequest
000047D0 7236                              jc .embfuerror
000047D2 E897FBFFFF                        call .ehciMsdBOTCheckTransaction
000047D7 6685C0                            test ax, ax
000047DA 752C                              jnz .embfuerror
                                       .embfu0:
000047DC E8CA000000                        call .ehciMsdBOTTestReady
000047E1 7225                              jc .embfuerror
000047E3 E886FBFFFF                        call .ehciMsdBOTCheckTransaction
000047E8 6685C0                            test ax, ax
000047EB 7411                              jz .embfuexit
000047ED E868000000                        call .ehciMsdBOTRequestSense
000047F2 7214                              jc .embfuerror
000047F4 E875FBFFFF                        call .ehciMsdBOTCheckTransaction
000047F9 6685C0                            test ax, ax
000047FC 75DE                              jnz .embfu0
                                       .embfuexit:
000047FE 415F                              pop r15
00004800 415E                              pop r14
00004802 415B                              pop r11
00004804 4158                              pop r8
00004806 58                                pop rax
00004807 C3                                ret
                                       .embfuerror:
00004808 F9                                stc
00004809 EBF3                              jmp short .embfuexit
                                       .ehciMsdBOTVerify:
                                       ;Input:
                                       ; rsi = Pointer to MSD table data structure that we want to Verify
                                                                  Sectors
                                       ; edx = Starting LBA to verify
                                       ;Output:
                                       ;    CF=CY: Host error, Reset host system
                                       ;    CF=NC: Transaction succeeded, check data transferred
                                                                  successfully
0000480B 50                                push rax
0000480C 4150                              push r8
0000480E 4153                              push r11
00004810 4154                              push r12
00004812 4156                              push r14
00004814 4157                              push r15
00004816 4D31C0                            xor r8, r8   ;Request no data
00004819 49BB0A000000000000-               mov r11, 0Ah   ;Command length is 10 bytes
00004822 00
00004823 4189D4                            mov r12d, edx
00004826 49BE-                             mov r14, .ehciMsdBOTOI
00004828 [4D45000000000000]
00004830 49BF-                             mov r15, .scsiVerify
00004832 [C64A000000000000]
0000483A E834FEFFFF                        call .ehciMsdBOTRequest
0000483F 7216                              jc .embvbad
00004841 E828FBFFFF                        call .ehciMsdBOTCheckTransaction
00004846 6685C0                            test ax, ax
00004849 750C                              jnz .embvbad
                                       .embvexit:
0000484B 415F                              pop r15
0000484D 415E                              pop r14
0000484F 415C                              pop r12
```

```
00004851 415B                              pop r11
00004853 4158                              pop r8
00004855 59                                pop rcx
00004856 C3                                ret
                                       .embvbad:
00004857 F9                                stc
00004858 EBF1                              jmp short .embvexit
                                       .ehciMsdBOTRequestSense:
                                       ;Input:
                                       ; rsi = Pointer to device MSD table data structure
                                       ;Output:
                                       ;    CF=CY: Host error, Reset host system
                                       ;    CF=NC: Transaction succeeded, check data transferred
                                                       successfully
0000485A 53                                push rbx
0000485B 51                                push rcx
0000485C 4150                              push r8
0000485E 4153                              push r11
00004860 4156                              push r14
00004862 4157                              push r15
00004864 48BB-                             mov rbx, ehciDataIn
00004866 [C003000000000000]
0000486E B900000000                        mov ecx, 0
00004873 49B812000000000000-               mov r8, 12h          ;Request 18 bytes
0000487C 00
0000487D 49BB0600000000000-                mov r11, 6           ;Command length is 6
00004886 00
00004887 49BF-                             mov r15, .scsiRequestSense
00004889 [7A4A000000000000]
00004891 49BE-                             mov r14, .ehciMsdBOTOII
00004893 [6144000000000000]
0000489B E8D3FDFFFF                        call .ehciMsdBOTRequest
000048A0 415F                              pop r15
000048A2 415E                              pop r14
000048A4 415B                              pop r11
000048A6 4158                              pop r8
000048A8 59                                pop rcx
000048A9 5B                                pop rbx
000048AA C3                                ret

                                       .ehciMsdBOTTestReady:
                                       ;Input:
                                       ; rsi = Pointer to MSD table data structure that we want to Test
                                                       Ready
                                       ;Output:
                                       ;    CF=CY: Host error, Reset host system
                                       ;    CF=NC: Transaction succeeded, check data transferred
                                                       successfully
000048AB 4150                              push r8
000048AD 4153                              push r11
000048AF 4156                              push r14
000048B1 4157                              push r15
000048B3 4D31C0                            xor r8, r8   ;Request no data
000048B6 49BB0600000000000-                mov r11, 6   ;Command length is 6
000048BF 00
000048C0 49BE-                             mov r14, .ehciMsdBOTOI
000048C2 [4D45000000000000]
000048CA 49BF-                             mov r15, .scsiTestUnitReady
000048CC [8D4A000000000000]
000048D4 E89AFDFFFF                        call .ehciMsdBOTRequest
000048D9 415F                              pop r15
000048DB 415E                              pop r14
000048DD 415B                              pop r11
000048DF 4158                              pop r8
000048E1 C3                                ret
                                       .ehciMsdBOTModeSense6:
                                       ;Input:
                                       ; rsi = Pointer to MSD table data structure that we want to Test
                                                       Ready
                                       ;Output:
                                       ;    CF=CY: Host error, Reset host system
                                       ;    CF=NC: Transaction succeeded, check data transferred
                                                       successfully
000048E2 53                                push rbx
000048E3 51                                push rcx
000048E4 4150                              push r8
000048E6 4153                              push r11
000048E8 4156                              push r14
000048EA 4157                              push r15
000048EC 48BB-                             mov rbx, ehciDataIn
000048EE [C003000000000000]
000048F6 B900000000                        mov ecx, 0
000048FB 49B8C000000000000-                mov r8, 0C0h         ;Request 192 bytes
```

```
00004904 00
00004905 49BB0600000000000000-     mov r11, 6            ;Command length is 6
0000490E 00
0000490F 49BF-                     mov r15, .scsiModeSense6
00004911 [E74A000000000000]
00004919 49BE-                     mov r14, .ehciMsdBOTOII
0000491B [6144000000000000]
00004923 E84BFDFFFF                call .ehciMsdBOTRequest
00004928 415F                      pop r15
0000492A 415E                      pop r14
0000492C 415B                      pop r11
0000492E 4158                      pop r8
00004930 59                        pop rcx
00004931 5B                        pop rbx
00004932 C3                        ret

                               ;.ehciMsdBOTOutSector64:
                               .ehciMsdBOTOutSector512:
                               ;Input:
                               ; rsi = Pointer to MSD table data structure that we want to read
                               ; rbx = Address of the buffer to read the segment from
                               ; edx = Starting LBA to read to
                               ;Output:
                               ;    CF=CY: Host error, Reset host system
                               ;    CF=NC: Transaction succeeded, check data transferred
                                                                    successfully
00004933 4151                      push r9
00004935 4156                      push r14
00004937 4157                      push r15
00004939 50                        push rax
0000493A 4D31C9                    xor r9, r9  ;Send an OUT packet
0000493D 49BE-                     mov r14, .ehciMsdBOTOOI
0000493F [9543000000000000]
00004947 49BF-                     mov r15, .scsiWrite10
00004949 [534A000000000000]
00004951 E860000000                call .ehciMsdBOTSector512
00004956 7223                      jc .emboseerror
00004958 E811FAFFFF                call .ehciMsdBOTCheckTransaction
0000495D 6685C0                    test ax, ax
00004960 7519                      jnz .emboseerror
00004962 E844FFFFFF                call .ehciMsdBOTTestReady    ;Seems to flush data onto disk
00004967 7212                      jc .emboseerror
00004969 E800FAFFFF                call .ehciMsdBOTCheckTransaction
0000496E 6685C0                    test ax, ax
00004971 7508                      jnz .emboseerror
                               .embosexit:
00004973 58                        pop rax
00004974 415F                      pop r15
00004976 415E                      pop r14
00004978 4159                      pop r9
0000497A C3                        ret
                               .emboseerror:
0000497B F9                        stc
0000497C EBF5                      jmp short .embosexit
                               ;.ehciMsdBOTInSector64:
                               .ehciMsdBOTInSector512:
                               ;Input:
                               ; rsi = Pointer to MSD table data structure that we want to read
                               ; rbx = Address of the buffer to read the segment into
                               ; edx = Starting LBA to read from
                               ;Output:
                               ;    CF=CY: Host error, Reset host system
                               ;    CF=NC: Transaction succeeded, check data transferred
                                                                    successfully
0000497E 4151                      push r9
00004980 4156                      push r14
00004982 4157                      push r15
00004984 50                        push rax
00004985 49B98000000000000000-     mov r9, 80h ;Recieve an IN packet
0000498E 00
0000498F 49BE-                     mov r14, .ehciMsdBOTOII
00004991 [6144000000000000]
00004999 49BF-                     mov r15, .scsiRead10
0000499B [574A000000000000]
000049A3 E80E000000                call .ehciMsdBOTSector512
000049A8 72D1                      jc .emboseerror
000049AA E8BFF9FFFF                call .ehciMsdBOTCheckTransaction
000049AF 6685C0                    test ax, ax
000049B2 75C7                      jnz .emboseerror
000049B4 EBBD                      jmp short .embosexit
                               .ehciMsdBOTSector512:
                               ;Input:
                               ; rsi = Pointer to MSD table data structure that we want to read
```

```
                                                ; rbx = Address of the buffer to read the segment into
                                                ; edx = Starting LBA to read to/from
                                                ; r9  = CBW flag (IN or OUT transaction)
                                                ; r15 = SCSI function
                                                ;Output:
                                                ;    CF=CY: Host error, Reset host system
                                                ;    CF=NC: Transaction succeeded, check data transferred
                                                                            successfully
000049B6 57                                         push rdi
000049B7 4150                                       push r8
000049B9 4152                                       push r10
000049BB 4153                                       push r11

000049BD 48BF-                                      mov rdi, ehciDataOut     ;Write the CBW at the data out point
000049BF [8003000000000000]
000049C7 41B800020000                               mov r8d, 200h            ;512 bytes to be transferred
000049CD 4D31D2                                     xor r10, r10             ;LUN 0
000049D0 49BB0C000000000000-                        mov r11, 0Ch             ;The command block is 10 bytes long
000049D9 00
000049DA E828000000                                 call .msdWriteCBW       ;Write the CBW

000049DF 50                                         push rax                 ;Temp push ax
000049E0 4150                                       push r8                  ;Temp save # of bytes for transfer
000049E2 30C0                                       xor al, al               ;LUN 0 device
000049E4 4189D0                                     mov r8d, edx             ;Starting LBA to read from
000049E7 49B901000000000000-                        mov r9, 1                ;Number of LBAs to read
000049F0 00
000049F1 41FFD7                                     call r15                 ;Write the valid CBW Command block
000049F4 4158                                       pop r8
000049F6 58                                         pop rax

000049F7 B90A000000                                 mov ecx, 10               ;Wait for data preparation, 10ms
000049FC 41FFD6                                     call r14

000049FF 415B                                       pop r11
00004A01 415A                                       pop r10
00004A03 4158                                       pop r8
00004A05 5F                                         pop rdi
00004A06 C3                                         ret
                                                .msdWriteCBW:
                                                ;Writes a Command Block Wrapper at the location pointed to by rdi
                                                ; without a functional command block. Must be appended by user.
                                                ; Input:  rdi=Pointer to CBW buffer
                                                ;         r8d=Command Block Wrapper Data Transfer Length
                                                ;         r9b=Command Block Wrapper Flags
                                                ;         r10b=Command Block Wrapper LUN nybble
                                                ;         r11b=Command Block Wrapper Command Block Length
                                                ; Output: rdi = Pointer to CBW's (SCSI) Command Descriptor Block
                                                                            buffer
00004A07 50                                         push rax
00004A08 B855534243                                 mov eax, CBWSig
00004A0D AB                                         stosd
00004A0E 0FB60425[4A020000]                         movzx eax, byte [cbwTag]
00004A16 FE0425[4A020000]                           inc byte [cbwTag]
00004A1D AB                                         stosd
00004A1E 4489C0                                     mov eax, r8d
00004A21 AB                                         stosd
00004A22 4488C8                                     mov al, r9b
00004A25 AA                                         stosb
00004A26 4488D0                                     mov al, r10b
00004A29 AA                                         stosb
00004A2A 4488D8                                     mov al, r11b
00004A2D AA                                         stosb
00004A2E 31C0                                       xor eax, eax
00004A30 57                                         push rdi
00004A31 48AB                                       stosq   ;16 bytes in csw command block
00004A33 48AB                                       stosq   ;Clear memory
00004A35 5F                                         pop rdi
00004A36 58                                         pop rax
00004A37 C3                                         ret

                                                ;                    ---------SCSI functions---------

                                                .scsiInquiry:
                                                ;Writes an inquiry scsi command block to the location pointed to by
                                                                            rdi
                                                ;al contains the LUN of the device we are accessing. (lower 3 bits
                                                                            considered)
                                                ;al not preserved
00004A38 B412                                       mov ah, 12h         ;Move inquiry command value high
00004A3A C0E005                                     shl al, 5         ;Shift left by five to align LUN properly
00004A3D 86E0                                       xchg ah, al         ;swap ah and al
00004A3F 66AB                                       stosw             ;Store command and shifted LUN together
```

```
00004A41 4831C0                        xor rax, rax
00004A44 66AB                          stosw              ;Store two zeros (reserved fields)
00004A46 48B824000000000000-           mov rax, 24h     ;Allocation length (36 bytes)
00004A4F 00
00004A50 48AB                          stosq
00004A52 C3                            ret
                                   ;NOTE! Using read/write 10 means can't read beyond the first 4 Gb of Medium.
                                   .scsiWrite10:
                                   ;Writes a scsi write 10 transfer command to the location pointed at by rdi
                                   ;al contains the LUN of the device we are accessing
                                   ;r8d contains the LBA start address
                                   ;r9w contains the Verification Length
00004A53 B42A                          mov ah, 2Ah          ;Operation code for command
00004A55 EB02                          jmp short .scsirw
                                   .scsiRead10:
                                   ;Writes a scsi Read 10 command to the location pointed to by rdi
                                   ;al contains the LUN of the device we are accessing.
                                   ;r8d contains the LBA to read from
                                   ;r9w contains the number of contiguous blocks to read (should be 1 for us)
00004A57 B428                          mov ah, 28h          ;Move read(10) command value high
                                   .scsirw:
00004A59 C0E005                        shl al, 5          ;Shift left by five to align LUN properly
00004A5C 86E0                          xchg ah, al          ;swap ah and al
00004A5E 66AB                          stosw              ;Store command and shifted LUN together
00004A60 410FC8                        bswap r8d          ;swap endianness of r8d
00004A63 4489C0                        mov eax, r8d
00004A66 AB                            stosd
00004A67 4831C0                        xor rax, rax     ;Clear for a Reserved byte
00004A6A AA                            stosb
00004A6B 664489C8                      mov ax, r9w          ;move into ax to use xchg on upper and lower bytes
00004A6F 86C4                          xchg al, ah          ;MSB first, yuck yuck yuck
00004A71 66AB                          stosw
00004A73 C1E810                        shr eax, 16          ;Bring zeros down onto lower word
00004A76 66AB                          stosw              ;Store one reserved byte and two padding bytes
00004A78 AA                            stosb
00004A79 C3                            ret
                                   .scsiRequestSense:
                                   ;Writes a scsi Request Sense command to the location pointer to by rdi
                                   ;al contains the LUN of the device we are accessing.
00004A7A B403                          mov ah, 03h          ;Move reqsense command value high
00004A7C C0E005                        shl al, 5          ;Shift left by five to align LUN properly
00004A7F 86E0                          xchg ah, al          ;swap ah and al
00004A81 66AB                          stosw              ;Store command and shifted LUN together
00004A83 4831C0                        xor rax, rax
00004A86 66AB                          stosw              ;Reserved word
00004A88 B012                          mov al, 12h     ;Move alloc length byte into al
00004A8A 48AB                          stosq
00004A8C C3                            ret
                                   .scsiTestUnitReady:
                                   ;Writes a scsi test unit ready command to the location pointed to by rdi
                                   ;al contains the LUN of the device we are accessing.
00004A8D 30E4                          xor ah, ah          ;Operation code zero
00004A8F C0E005                        shl al, 5
00004A92 86E0                          xchg ah, al
00004A94 66AB                          stosw              ;Store shifted LUN and command code
00004A96 C3                            ret
                                   .scsiReadFormatCapacities:
                                   ;al contains the LUN of the device
00004A97 88C4                          mov ah, al
00004A99 B023                          mov al, 23h          ;Operation code for command
00004A9B 66AB                          stosw              ;Store shifted LUN and command code
00004A9D 4831C0                        xor rax, rax
00004AA0 AB                            stosd             ;Reserved dword
00004AA1 66AB                          stosw             ;Reserved word
00004AA3 B0FC                          mov al, 0FCh     ;Move alloc length byte into al
00004AA5 AA                            stosb
00004AA6 C3                            ret
                                   .scsiReadCap10:
                                   ;Writes a scsi read capacity command to the location pointed to by rdi
                                   ;al contains the LUN of the device we are accessing
00004AA7 B425                          mov ah, 25h          ;Operation code for command
00004AA9 C0E005                        shl al, 5
00004AAC 86E0                          xchg ah, al
00004AAE 66AB                          stosw              ;Store shifted LUN and command code
00004AB0 C3                            ret
                                   .scsiFormatUnit:
                                    ;Writes a scsi format unit command to the location pointed to by rdi
                                    ;al contains the LUN of the device we are accessing
00004AB1 B404                           mov ah, 04h         ;Operation code for format command
00004AB3 C0E005                         shl al, 5
00004AB6 0C17                           or al, 17h          ;Set bits [3:0] and 5, keep bit 4 clear
00004AB8 86E0                           xchg ah, al
00004ABA 66AB                           stosw
00004ABC 30C0                           xor al, al
00004ABE 66AB                           stosw               ;Vender specific, set to 0!!
00004AC0 4831C0                         xor rax, rax
00004AC3 48AB                           stosq               ;Store LSB byte and all the 0 padding
00004AC5 C3                             ret
                                    .scsiVerify:
                                    ;Writes a scsi verify transfer command to the location pointed at by rdi
                                    ;al contains the LUN of the device we are accessing
                                    ;r12d contains the LBA for the sector address
                                    ;Verifies one sector
00004AC6 B42F                           mov ah, 2Fh         ;Operation code for command
00004AC8 C0E005                         shl al, 5           ;Hardcode bytecheck (byte [1]) to 0
00004ACB 86E0                           xchg ah, al
00004ACD 66AB                           stosw               ;Store shifted LUN and command code
00004ACF 410FCC                         bswap r12d          ;swap endianness of r12d
00004AD2 4489E0                         mov eax, r12d
00004AD5 AB                             stosd
00004AD6 4831C0                         xor rax, rax        ;Clear for a Reserved byte
00004AD9 AA                             stosb
00004ADA 66B80001                       mov ax, 0100h       ;Write the number 1 in Big endian
00004ADE 66AB                           stosw
00004AE0 C1E810                         shr eax, 16         ;Bring zeros down onto lower word
00004AE3 66AB                           stosw               ;Store one reserved byte and two padding bytes
00004AE5 AA                             stosb
00004AE6 C3                             ret
                                    .scsiModeSense6:
                                    ;al contains the LUN of the device we are accessing
00004AE7 B41A                           mov ah, 1Ah         ;Operation code for Mode Sense 6
00004AE9 C0E005                         shl al, 5           ;Move LUN
00004AEC 86E0                           xchg ah, al
00004AEE 66AB                           stosw
00004AF0 B83F00C000                     mov eax, 0C0003Fh
                                        ;Request all pages, reserve byte, 192 bytes and 0 end byte
00004AF5 AB                             stosd
00004AF6 C3                             ret
                                    ;                   ________________________________
                                    .ehciGetOpBase:
                                    ;Gets opbase from mmio base (aka adds caplength) into eax
                                    ;Input:
                                    ; al = offset into ehci table
                                    ;Return:
                                    ; eax = opbase (low 4Gb)
00004AF7 53                             push rbx
00004AF8 4831DB                         xor rbx, rbx
00004AFB 480FB6C0                       movzx rax, al
00004AFF 8B04C5[19020000]               mov eax, dword [eControllerList + 4 + 8*rax]    ;get mmiobase into eax
00004B06 85C0                           test eax, eax               ;addrress of 0 means no controller
00004B08 7406                           jz .egob1
00004B0A 670FB618                       movzx ebx, byte [eax]       ;get the offset to opbase into ebx
00004B0E 01D8                           add eax, ebx                ;add this offset to mmiobase to get opbase
                                    .egob1:
00004B10 5B                             pop rbx
00004B11 C3                             ret

                                    ;========================CPU Interrupts========================
                                    i0:
00004B12 4831C0                         xor rax, rax
00004B15 E936010000                     jmp cpu_2args
                                    i1:
00004B1A 48B801000000000000-            mov rax, 1
00004B23 00
00004B24 E927010000                     jmp cpu_2args
                                    i2:
00004B29 48B802000000000000-            mov rax, 2
00004B32 00
00004B33 E918010000                     jmp cpu_2args
                                    i3:
00004B38 48B803000000000000-            mov rax, 3
00004B41 00
00004B42 E909010000                     jmp cpu_2args
                                    i4:
00004B47 48B804000000000000-            mov rax, 4
00004B50 00
```

```
00004B51 E9FA000000                  jmp cpu_2args
                                 i5:
00004B56 48B805000000000000-         mov rax, 5
00004B5F 00
00004B60 E9EB000000                  jmp cpu_2args
                                 i6:
00004B65 48B806000000000000-         mov rax, 6
00004B6E 00
00004B6F E9DC000000                  jmp cpu_2args
                                 i7:
00004B74 48B807000000000000-         mov rax, 7
00004B7D 00
00004B7E E9CD000000                  jmp cpu_2args
                                 i8:
00004B83 48B808000000000000-         mov rax, 8
00004B8C 00
00004B8D E9B2000000                  jmp cpu_3args
                                 i9:
00004B92 48B809000000000000-         mov rax, 9
00004B9B 00
00004B9C E9AF000000                  jmp cpu_2args
                                 i10:
00004BA1 48B80A000000000000-         mov rax, 0Ah
00004BAA 00
00004BAB E994000000                  jmp cpu_3args
                                 i11:
00004BB0 48B80B000000000000-         mov rax, 0Bh
00004BB9 00
00004BBA E985000000                  jmp cpu_3args
                                 i12:
00004BBF 48B80C000000000000-         mov rax, 0Ch
00004BC8 00
00004BC9 E976000000                  jmp cpu_3args
                                 i13:
00004BCE 48B80D000000000000-         mov rax, 0Dh
00004BD7 00
00004BD8 EB6A                        jmp short cpu_3args
                                 i14:
00004BDA 48B80E000000000000-         mov rax, 0Eh
00004BE3 00
00004BE4 EB52                        jmp short cpu_4args
                                 i15:
00004BE6 48B80F000000000000-         mov rax, 0Fh
00004BEF 00
00004BF0 EB5E                        jmp short cpu_2args
                                 i16:
00004BF2 48B810000000000000-         mov rax, 10h
00004BFB 00
00004BFC EB52                        jmp short cpu_2args
                                 i17:
00004BFE 48B811000000000000-         mov rax, 11h
00004C07 00
00004C08 EB3A                        jmp short cpu_3args
                                 i18:
00004C0A 48B812000000000000-         mov rax, 12h
00004C13 00
00004C14 EB3A                        jmp short cpu_2args
                                 i19:
00004C16 48B813000000000000-         mov rax, 13h
00004C1F 00
00004C20 EB2E                        jmp short cpu_2args
                                 i20:
00004C22 48B814000000000000-         mov rax, 14h
00004C2B 00
00004C2C EB22                        jmp short cpu_2args
                                 i21:
00004C2E 48B815000000000000-         mov rax, 15h
00004C37 00
                                 cpu_4args:
00004C38 48B903000000000000-         mov rcx, 3
00004C41 00
00004C42 EB16                        jmp short cpu_exception
                                 cpu_3args:
00004C44 48B902000000000000-         mov rcx, 2
00004C4D 00
00004C4E EB0A                        jmp short cpu_exception
                                 cpu_2args:
00004C50 48B901000000000000-         mov rcx, 1
00004C59 00
                                 cpu_exception:
00004C5A 50                          push rax
00004C5B 51                          push rcx
00004C5C 66BB1F00                    mov bx, 001Fh       ;cls attribs
```

```
10288 00004C60 E892B4FFFF                 call cls
10289
10290 00004C65 48B800020000000000-        mov rax, 0200h
10290 00004C6E 00
10291 00004C6F 4831DB                     xor rbx, rbx
10292 00004C72 48BA2207000000000-         mov rdx, 0722h    ;7 Rows down, 24 columns across
10292 00004C7B 00
10293 00004C7C 48BD-                      mov rbp, .fatalt0
10293 00004C7E [C74D000000000000]
10294 00004C86 66BB7100                   mov bx, 0071h     ;blue grey attribs, page 0
10295 00004C8A 66B80113                   mov ax, 1301h     ;print zero 8 chars, with bh attrib
10296 00004C8E 48B908000000000000-        mov rcx, 8
10296 00004C97 00
10297 00004C98 CD30                       int 30h
10298
10299 00004C9A 48B800020000000000-        mov rax, 0200h
10299 00004CA3 00
10300 00004CA4 30FF                       xor bh, bh
10301 00004CA6 48BA040A0000000000-        mov rdx, 0A04h    ;11 Rows down, 24 columns across
10301 00004CAF 00
10302 00004CB0 CD30                       int 30h
10303 00004CB2 48BD-                      mov rbp, .fatal1
10303 00004CB4 [CF4D000000000000]
10304 00004CBC 30FF                       xor bh, bh        ;blue grey attribs, page 0
10305 00004CBE 66B80413                   mov ax, 1304h           ;print zero terminated string
10306 00004CC2 CD30                       int 30h
10307
10308 00004CC4 59                         pop rcx
10309 00004CC5 58                         pop rax                 ;pop the exception number back into rax
10310 00004CC6 E8DF000000                 call .printbyte
10311
10312 00004CCB 48B804130000000000-        mov rax, 1304h
10312 00004CD4 00
10313 00004CD5 30FF                       xor bh, bh
10314 00004CD7 48BD-                      mov rbp, .fatal2
10314 00004CD9 [104F000000000000]
10315 00004CE1 CD30                       int 30h
10316
10317 00004CE3 80F901                     cmp cl, 1
10318 00004CE6 773A                       ja .cpuextendederror    ;rax contains error code, or extra cr2
                                                                    value
10319                                   .cpurollprint:
10320 00004CE8 488B1424                   mov rdx, qword [rsp]    ;Get address
10321                                   ;Takes whats in rdx, rols left by one byte, prints al
10322 00004CEC B108                       mov cl, 8    ;8 bytes
10323                                   .cpurollprint1:
10324 00004CEE 48C1C208                   rol rdx, 8
10325 00004CF2 88D0                       mov al, dl
10326 00004CF4 52                         push rdx
10327 00004CF5 E8B0000000                 call .printbyte
10328 00004CFA 5A                         pop rdx
10329 00004CFB FEC9                       dec cl
10330 00004CFD 75EF                       jnz .cpurollprint1
10331
10332                                   .cpuexendloop:
10333 00004CFF 6631C0                     xor ax, ax
10334 00004D02 CD36                       int 36h
10335 00004D04 3C1B                       cmp al, 1Bh    ;Check for escape pressed (unlikely?)
10336 00004D06 740F                       je .cpu_exception_appret
10337 00004D08 3C0D                       cmp al, 0Dh ;Check for enter pressed
10338 00004D0A 75F3                       jne .cpuexendloop
10339
10340 00004D0C 66BB0700                   mov bx, 0007h     ;cls attribs
10341 00004D10 E8E2B3FFFF                 call cls
10342 00004D15 CD38                       int 38h    ;Jump to debugger
10343                                   .cpu_exception_appret:
10344 00004D17 66BB0700                   mov bx, 0007h     ;cls attribs
10345 00004D1B E8D7B3FFFF                 call cls
10346 00004D20 48CF                       iretq ;Return to address on stack
10347
10348                                   .cpuextendederror:
10349 00004D22 5A                         pop rdx
10350 00004D23 48FFC9                     dec rcx
10351 00004D26 51                         push rcx
10352 00004D27 B102                       mov cl, 2      ;CAN CHANGE TO 4 BYTES IN THE FUTURE
10353                                   .pr1:
10354 00004D29 C1C208                     rol edx, 8     ;Print just edx
10355 00004D2C 88D0                       mov al, dl
10356 00004D2E 52                         push rdx
10357 00004D2F E876000000                 call .printbyte
10358 00004D34 5A                         pop rdx
10359 00004D35 FEC9                       dec cl
10360 00004D37 75F0                       jnz .pr1
```

```

00004D39 48B804130000000000-        mov rax, 1304h
00004D42 00
00004D43 48BB17000000000000-        mov rbx, 17h
00004D4C 00
00004D4D 48BD-                      mov rbp, .fatal2
00004D4F [104F000000000000]
00004D57 CD30                       int 30h
00004D59 59                         pop rcx     ;Bring the comparison value back into rcx

00004D5A 48FFC9                     dec rcx
00004D5D 7489                       jz .cpurollprint

00004D5F B108                       mov cl, 8
00004D61 0F20D2                     mov rdx, cr2     ;Get page fault address
                                .pr2:
00004D64 48C1C208                   rol rdx, 8     ;Print rdx
00004D68 88D0                       mov al, dl
00004D6A 52                         push rdx
00004D6B E83A000000                 call .printbyte
00004D70 5A                         pop rdx
00004D71 FEC9                       dec cl
00004D73 75EF                       jnz .pr2

00004D75 48B804130000000000-        mov rax, 1304h
00004D7E 00
00004D7F 48BB17000000000000-        mov rbx, 17h
00004D88 00
00004D89 48BD-                      mov rbp, .fatal2
00004D8B [104F000000000000]
00004D93 CD30                       int 30h

00004D95 E94EFFFFFF                 jmp .cpurollprint



                                .char:     ;Print a single character
00004D9A 48BB-                      mov rbx, .ascii
00004D9C [144F000000000000]
00004DA4 D7                         xlatb     ;point al to entry in ascii table, using al as offset
                                                     into table

                                    ;xor bh, bh
00004DA5 B40E                       mov ah, 0Eh
00004DA7 CD30                       int 30h     ;print char
00004DA9 C3                         ret
                                .printbyte:
00004DAA 88C2                       mov dl, al          ;save byte in dl
00004DAC 6625F000                   and ax, 00F0h       ;Hi nybble
00004DB0 6681E20F00                 and dx, 000Fh       ;Lo nybble
00004DB5 66C1E804                   shr ax, 4           ;shift one hex place value pos right
00004DB9 E8DCFFFFFF                 call .char
00004DBE 6689D0                     mov ax, dx          ;mov lo nybble, to print
00004DC1 E8D4FFFFFF                 call .char
00004DC6 C3                         ret
00004DC7 5343502F42494F53       .fatalt0:  db "SCP/BIOS"
00004DCF 4120706F74656E7469-    .fatal1:   db "A potentially fatal error has occured. To continue:
                                                     ",0Ah,0Ah,0Dh

00004DD8 616C6C792066617461-
00004DE1 6C206572726F722068-
00004DEA 6173206F6363757265-
00004DF3 642E20546F20636F6E-
00004DFC 74696E75653A200A0A-
00004E05 0D
00004E06 202020205072657373-    db "     Press Enter to launch SYSDEBUG, or",0Ah,0Ah,0Dh
00004E0F 20456E74657220746F-
00004E18 206C61756E63682053-
00004E21 595344454255472C20-
00004E2A 6F720A0A0D
00004E2F 202020205072657373-    db "     Press ESC to try and return to the application which caused
                                                     the error,"

00004E38 2045534320746F2074-
00004E41 727920616E64207265-
00004E4A 7475726E20746F2074-
00004E53 6865206170706C6963-
00004E5C 6174696F6E20776869-
00004E65 636820636175736564-
00004E6E 20746865206572726F-
00004E77 722C
00004E79 6F720A0A0D             db "or", 0Ah, 0Ah,0Dh,
00004E7E 202020205072657373-    db "     Press CTRL+ALT+DEL to restart your system. If you do
                                                     this,",0Ah,0Dh

00004E87 204354524C2B414C54-
00004E90 2B44454C20746F2072-
00004E99 65737461727420796F-
```

```
10411 00004EA2 75722073797374656D-
10411 00004EAB 2E20496620796F7520-
10411 00004EB4 646F20746869732C0A-
10411 00004EBD 0D
10412 00004EBE 20202020796F752077-   db "     you will lose any unsaved information in all open
                                                              applications.",0Ah,
10412 00004EC7 696C6C206C6F736520-
10412 00004ED0 616E7920756E736176-
10412 00004ED9 656420696E666F726D-
10412 00004EE2 6174696F6E20696E20-
10412 00004EEB 616C6C206F70656E20-
10412 00004EF4 6170706C6963617469-
10412 00004EFD 6F6E732E0A
10413 00004F02 0A0D                 db 0Ah, 0Dh
10414 00004F04 202020204572726F72-   db "     Error: ",0
10414 00004F0D 3A2000
10415 00004F10 203A2000              .fatal2:    db " : ",0
10416 00004F14 303132333435363738-   .ascii:     db '0123456789ABCDEF'
10416 00004F1D 39414243444546
10417                                ;==========================Dummy Interrupts======================
10418                                dummy_interrupt:
10419                                .pic2:
10420 00004F24 50                        push rax
10421 00004F25 B020                      mov al, EOI
10422 00004F27 E6A0                      out pic2command, al      ;EOI to pic2
10423 00004F29 EB01                      jmp short .p1
10424                                .pic1:
10425 00004F2B 50                        push rax
10426                                .p1:
10427 00004F2C B020                      mov al, EOI
10428 00004F2E E620                      out pic1command, al      ;EOI to pic2
10429 00004F30 58                        pop rax
10430                                dummy_return_64:
10431 00004F31 48CF                      iretq
10432                                ;------------------------------------------------------------------
10433 00004F33 76302E392053435042-   signature:     db "v0.9 SCPBIOS"     ;12 byte signature
10433 00004F3C 494F53
10434 00004F3F 436F70797269676874-   signature2:    db "Copyright (C) Yll Buzoku"
10434 00004F48 2028432920596C6C20-
10434 00004F51 42757A6F6B75
10435 00004F57 30332F31322F323032-                  db "03/12/2021"
10435 00004F60 31
10436
10437                                codeResidentEndPtr:
10438                                residentLength   equ $-$$
```

Example SCP/BIOS compatible bootloader

```
LINE   LOC    OBJ                                   SOURCE

  1                                    BITS 16
  2                                    ORG     600h
  3
  4                                    relocBase   equ 600h ;Relocate to 600h
  5                                    loadAddress equ 800h
  6                                    startSector equ 33
  7 00000000 EB3C                          jmp short start
  8 00000002 90                            nop
  9                                    ;——————————————————————Tables——————————————————————
 10 00000003 534350444F535631              osname: db 'SCPDOSV1'
 11
 12                                        ;Start of BIOS Parameter Block
 13
 14 0000000B 0002                          bypsec: dw 0200h      ;bytes per sector (200h=512)
 15 0000000D 01                            secpcl: db 01h        ;sectors per cluster
 16 0000000E 0100                          ressec: dw 0001h      ;reserved sectors
 17 00000010 02                            numFAT: db 02h        ;number of FATs
 18 00000011 E000                          nortdr: dw 00E0h      ;number of root directory entries
 19 00000013 400B                          nosect: dw 0B40h      ;number of sectors (1440 sectors per side)
 20 00000015 F0                            medesc: db 0F0h       ;media descriptor (f0=FDD)
 21 00000016 0900                          FATsec: dw 0009h      ;number of sectors per FAT
 22 00000018 1200                          sectrc: dw 0012h      ;number of sectors/tracks
 23 0000001A 0200                          numhed: dw 0002h      ;number of read/write heads
 24 0000001C 00000000                      numhid: dd 00000000h ;number of hidden sectors
 25 00000020 00000000                      nsecfs: dd 00000000h ;number of "huge" sectors in the FS (FAT)
 26
 27                                        ;End of BPB
 28
 29 00000024 00                            ldrvnu: db 00h        ;logical drive number, 80h=first HDD,
                                                                        00h=1st FDD
 30 00000025 00                            res1:   db 00h        ;reserved sector 1, BS reserved, used in
                                                                        boot
 31 00000026 29                            extsig: db 29h        ;Extended boot signature (29h = EBPB
                                                                        signature)
 32
 33                                        ;Start of Extended BPB
 34 00000027 0F0D2A1C                      sernum: dd 1C2A0D0Fh        ;serial number of drive
 35 0000002B 4E4F204E414D452020-           vollbl: db 'NO NAME    '   ;default volume label name
 35 00000034 2020
 36 00000036 4641543132202020              fstype: db 'FAT12   '      ;file system type
 37
 38                                    ;———————————————————————————————————————————————————
 39                                    start:
 40 0000003E 31C0                          xor ax, ax
 41 00000040 8ED8                          mov ds, ax
 42 00000042 8EC0                          mov es, ax
 43 00000044 8ED0                          mov ss, ax
 44 00000046 BC0080                        mov sp, 8000h
 45 00000049 BE007C                        mov si, 7C00h
 46 0000004C BF0006                        mov di, relocBase
 47 0000004F B90001                        mov cx, 100h
 48 00000052 FC                            cld               ;Ensure writes are in the write direction
 49 00000053 F3A5                          rep movsw
 50 00000055 EA[5A00]0000                  jmp 0:s1        ;Far jump to the next instruction
 51
 52                                    s1:
 53 0000005A 89D6                          mov si, dx   ;Save drive number in si
 54 0000005C B801E8                        mov ax, 0e801h
 55 0000005F CD15                          int 15h
 56 00000061 3D0008                        cmp ax, 800h      ;Get number of Kb
 57 00000064 730A                          jae .s2           ;Above or equal, OK!
 58 00000066 30C0                          xor al, al        ;Error code
 59 00000068 81F90008                      cmp cx, 800h
 60 0000006C 0F828E00                      jb fail
 61                                    .s2:
 62 00000070 B80300                        mov ax, 03h
 63 00000073 CD10                          int 10h ;set video mode
 64                                    ;sectrc used and numhed used for sectors per track and number of
                                                                        heads
 65 00000075 89F2                          mov dx, si
 66 00000077 8816[0000]                    mov byte [drvnum], dl    ;Save the drive byte from dl
 67 0000007B F6C280                        test dl, 80h
 68 0000007E 742E                          jz readFloppy
 69                                    ;If the boot device is emulated as a hard drive,
 70                                    ;   use BIOS extensions as CHS is buggy.
 71 00000080 BE[0400]                      mov si, pktptr
 72 00000083 89F7                          mov di, si
 73 00000085 31C0                          xor ax, ax
```

```
00000087 B90800                      mov cx, 8
0000008A F3AB                        rep stosw    ;Store 8 zero words
0000008C C7041000                    mov word [si], 0010h      ;Packet size and reserved zero
00000090 C744023A00                  mov word [si + 2], 58    ;Number of sectors to transfer
00000095 C744040008                  mov word [si + 4], loadAddress    ;Offset of buffer
0000009A C744060000                  mov word [si + 6], 0          ;Segment of buffer
0000009F C744082100                  mov word [si + 8], startSector       ;Starting sector
000000A4 B442                        mov ah, 42h
000000A6 CD13                        int 13h
000000A8 B406                        mov ah, 6
000000AA 7252                        jc fail
000000AC EB41                        jmp short launchSCP
                                 readFloppy:
000000AE BE1000                      mov si, 10h       ;Up to 16 error retries
000000B1 BF3A00                      mov di, 58        ;Copy MAXIMUM 58 sectors!!!!
000000B4 BD2100                      mov bp, startSector         ;Start at LBA 33
000000B7 BB0008                      mov bx, loadAddress     ;Start copy buffer at 800h
                                 readDisk:
                                 ;Convert bp into CHS for int 13h
000000BA 55                          push bp           ;Save the current LBA on the stack temporarily
                                 ;Sector
000000BB 89E8                        mov ax, bp             ;mov LBA into ax to get head and sec num
000000BD F636[1800]                      div byte [sectrc]         ;divide ax by the low byte of
                                                                    sectrc
000000C1 FEC4                            inc ah                    ;increment the remainder to get
                                                                    sectors
000000C3 88E1                            mov cl, ah                ;save the remainder in its ret
                                                                    register
                                 ;———————————————————————————————————————
                                 ;Head
000000C5 30E4                            xor ah, ah                ;nullify the remainder for the next
                                                                   part
000000C7 F636[1A00]                      div byte [numhed]         ;divide ax by the low byte of numhed
000000CB 88E5                            mov ch, ah                ;Save the head in ch
                                 ;———————————————————————————————————————
                                 ;Cylinder
000000CD A1[1A00]                        mov ax, word [numhed]     ;mov numhead into ax
000000D0 F726[1800]                      mul word [sectrc]              ;multiply ax by sec/trc
000000D4 95                              xchg bp, ax               ;switch bp and ax so that we can
                                                                   divide them
000000D5 F7F5                            div bp                    ;Divide them here!
000000D7 88C6                            mov dh, al                ;Save the result in dh
                                 ;———————————————————————————————————————
000000D9 86EE                        xchg ch, dh       ;Swap ch and dh for return value
000000DB 5D                          pop bp            ;Return the current LBA
000000DC 8A16[0000]                  mov dl, byte [drvnum]    ;we saved the drive in medesc
000000E0 B80102                      mov ax, 0201h             ;Disk read, one sector at a time
000000E3 CD13                        int 13h
000000E5 7210                        jc diskError              ; Error detected, restart file copy
000000E7 81C30002                    add bx, 200h          ; Goto next sector position
000000EB 45                          inc bp
000000EC 4F                          dec di
000000ED 75CB                        jnz readDisk
                                 launchSCP:
                                 ;Construct SCPBIOS SysInit Parameter Table
000000EF BB[3601]                    mov bx, SysInitTable     ;es points to segment, get table to bx
000000F2 EA00080000                  jmp 0:loadAddress ; go to the next file
                                 diskError:
000000F7 31C0                        xor ax, ax            ; Disk reset
000000F9 CD13                        int 13h
000000FB 4E                          dec si
000000FC 75BC                        jnz readDisk          ; Reset disk and read sector again
                                 ;——————————————————————————————Errors——————————————————————————————
                                 fail:
000000FE BE[1501]                    mov si, .msg
                                 .write: ;destroys registers ax and bx
00000101 AC                          lodsb
00000102 3C00                        cmp al, 0 ;check for zero
00000104 7409                        je .cont
00000106 B40E                        mov ah, 0Eh ;TTY output
00000108 BB0700                      mov bx, 0007h ;colour
0000010B CD10                        int 10h
0000010D EBF2                        jmp short .write
                                 .cont:
0000010F 31C0                        xor ax, ax
00000111 CD16                        int 16h       ;await keystroke
00000113 CD18                        int 18h       ;Reset
00000115 4E6F6E205379737465-     .msg: db "Non System Disk or Disk Error.",0Ah,0Dh,0
0000011E 6D204469736B206F72-
00000127 204469736B20457272-
00000130 6F722E0A0D00

                                 SysInitTable:
00000136 0C                      .lengthb db 0Ch
```

```
149 00000137 01                        .numSecb db 1
150 00000138 0000                      .resWord dw 00h
151 0000013A 5800000000000000          .FileLBA dq 88   ;Start at Sector 88
152
153 00000142 E8<rep BCh>               times 510-($-$$) db 0E8h
154 000001FE 55                            db 55h
155 000001FF AA                            db 0AAh
156
157                                    Segment .bss nobits start=502h
158 00000000 ??                        drvnum  resb  1 ;Drive number
159 00000001 ??????                        alignb 4
160 00000004 <res 10h>                 pktptr  resq  2 ;Packet Pointer, 16 bytes in size
```

Appendix B: Character Set and Scan Codes

This section will include the details pertaining to the character set and the scan codes sent by the keyboard to the computer. For reference, SCP/BIOS uses the standard IBM scancode set 1, which is the original IBM PC keyboard scancode set. Full details on this scancode set can be found in the IBM Technical Reference Manuals for the IBM PC family and on the web.

Appendix C: Using SYSDEBUG

SYSDEBUG is a simple debugger that a programmer can use to see the state of the system whilst writing application programs or to debug issues in the system. It is contained within SCP/BIOS to be ever present and usable by any application program without the programmer needing to write complex debugging subroutines. SYSDEBUG also includes the ability to return to a calling application via the "quit" command. An applications programmer can also use the relevant SCP/BIOS functions to connect and disconnect the debugger whenever convenient. At this stage the debugger is limited to being only properly usable on screen page 0 and not via the serial port. The debugger is also keyboard driven and non-reentrant.
When SYSDEBUG is entered, the user will be presented with the SYSDEBUG prompt, which is a hyphen "-".

The following eighteen programs are included with SYSDEBUG:

Command Letter	Command Name	Command Description
d	DUMP	Dumps system memory
e	EDIT	Edit a byte of system memory
s	SINGLE STEP	Single Step a procedure
g	GO TO	Transfers control to an address
p	PROCEED	Returns control to a subroutine

l	LOAD	Loads logical block(s) from a block storage device
w	WRITE	Writes logical block(s) to a block storage device
q	QUIT	Exit SYSDEBUG
c	CLEAR SCREEN	Clears the display
r	REGISTERS	Display and edit the system general purpose registers
b	BREAKPOINTS	Display and edit the systems debug registers
h	HEXADECIMAL CALCULATOR	Compute the sum and difference of two 64-bit values
i	IN PORT	Read a byte from an I/O port
o	OUT PORT	Write a byte to an I/O port
v	VERSION	Display the SCP/BIOS and SYSDEBUG version number
m	PRINT MEMORY	Print the system memory map
k	CONNECT DEBUGGER	Connect SYSDEBUG to the system
x	DISCONNECT DEBUGGER	Disconnect SYSDEBUG from the system

To use these commands, you must type the command letter in lower case, for that particular command at the prompt. A space will be entered for you if a valid letter is pressed.

If an invalid letter is pressed the error message "∧ Error" will be displayed underneath the invalidly typed letter and a new prompt will be displayed.

Once a valid command letter has been typed, depending on the command, you may need to enter a number of other arguments as will be outlined in the section below. If the command needs additional arguments, you may exit from entering other arguments by pressing the “q” key, and return to the prompt.

Guide to internal programs

Below is a reference guide on how to use each of SYSDEBUG’s commands. Each command argument is separated by a space and is indicated as follows: [Usage,x] with Usage indicating what the meaning of the value that should go there is, and x indicating *up to* how many digits this value can be. If an argument is optional, it will be written in curly braces as so {Usage, x}. All values are given in hexadecimal with alphabetical digits given in lowercase.
Warning! SYSDEBUG does not always check that input is correct and incorrect input may cause a system error. Please be mindful when inputting commands.

The DUMP command

This command allows the programmer to produce a memory dump. This dump will display the contents of the memory region selected in both hexadecimal and ASCII characters. To use this command, the programmer must type the following at the prompt:

- d {Address, 16} {Number of bytes to print, 16}

The default behaviour is to print 128 bytes starting at the current RIP value. If only an address is specified then the default behaviour is to print 128 bytes starting at the specified address.

The EDIT command

This command allows the programmer to edit memory one byte at a time. To use this command, the programmer must type the following at the prompt:

- e [Address of the byte in memory, 16]

Once the user has typed in the command letter and the address of the byte they wish to edit, they will be presented with the current byte at that location and a “ . ” symbol. The user can now type in the new byte they wish to store at this location or “q” to exit. Once the user has typed in the correct byte they wish to enter, they must strike enter for the edit to be made.
This command can be used to insert software breakpoints “CCH” anywhere in code, which if the debugger is connected, then will break into the debugger.

The SINGLE STEP command

This command allows a programmer to single step through a procedure. When the debugger has been connected, using the relevant SCP/BIOS command, when a hardware or software breakpoint is hit, then control will be given to SYSDEBUG. To then proceed through that particular subroutine or procedure one instruction at a time, the programmer can proceed using the single step command. Upon entering the debugger via a breakpoint, the programmer will be presented with a screen which shows the start of the general purpose system registers and the hardware breakpoint registers. To use this command, the programmer must type "s" at the prompt. Once this command is pressed, one instruction is executed and control is returned to SYSDEBUG via INT 01H. Therefore, correct usage of this command is dependent on the debugger being connected to both Interrupts 01H and 03H, as can be set using the relevant SCP/BIOS commands.

The GO TO command

This command allows a programmer to transfer control to a particular address in memory. To use this command, the programmer must type the following at the prompt:

- g {Address, 16}

The default address to jump to is the value stored in RIP.

The PROCEED command

This command allows a programmer to transfer control to a particular subroutine in memory. Its purpose is to return control to a program that is being debugged, after having broken into the debugger via a breakpoint. This command is the same as the default behaviour of the GO TO command in that returns to the address pointed to by RIP.

The LOAD command

This command allows the programmer to load logical blocks from a valid block storage device into system memory. To use this command, the programmer must type the following at the prompt:

- l [Address to store data at, 16] [SCP/BIOS device number, 2] [Start LBA number, 16] [Number of sectors to read, 16]

The WRITE command

This command allows the programmer to write logical blocks of data from system memory to a valid block storage device. To use this command, the programmer must type the following at the prompt:

- w [Address to read data from, 16] [SCP/BIOS device number, 2] [Start LBA number, 16] [Number of sectors to write, 16]

The QUIT command

This command allows the programmer to quit SYSDEBUG and return to the calling application. This command is run by pressing “q” only when at the prompt. Pressing “q” at any other time will only cancel the command. The details of how to return to a calling application are provided in the section below. After system initialisation, this function simply returns control back to SYSDEBUG and as such care must be taken, as pressing “q” whilst debugging reenters SYSDEBUG and destroys the state of the program being debugged. The provision of this command allows a programmer to launch SYSDEBUG from an operating system or application program, connect the debugger, launch an application to debug, enter the debugger from a breakpoint or CTRL+BREAK event, disconnect the debugger and return to the operating system or application program.

The CLEAR SCREEN command

This command will clear the screen.

The REGISTERS command

This command will display and allow the programmer to edit the main system registers before transferring control to a subroutine. When a programmer uses this command, they will be presented with a dump of the system registers. The programmer will then be prompted with the “ . ” prompt, where the programmer can then type in the hexadecimal number of a register to print, or press “q” to return back to the prompt. If a valid register number is entered, the name of the register will be printed followed by an equals sign, and the programmer can then type in the new 64-bit value they wish to store in that register, or “q” or Carriage Return to return back to the prompt. If an invalid register number is entered, the programmer will be presented with an “∧ Error” message and will be returned to the prompt. The general purpose registers are enumerated as follows:

RAX	0
RBX	1
RCX	2
RDX	3
RSI	4
RDI	5
R8	6
R9	7
R10	8
R11	9
R12	A
R13	B
R14	C
R15	D
RBP	E
RSP	F
RIP	10
RFLAGS	11

The BREAKPOINTS command

This command will display and allow the programmer to edit the system debugging registers before entering a subroutine or after being broken into by a breakpoint. When a programmer uses this command, they will be presented with a dump of the debug registers. The programmer will then be prompted with the " . " prompt, where the programmer can then type in the hexadecimal number of a register to print, or press "q" to return back to the prompt. If a valid register number is entered, the name of the register will be printed followed by an equals sign, and the programmer can then type in the new 64-bit value they wish to store in that register, or "q" or Carriage Return to return back to the prompt. If an invalid register number is entered, the programmer will be presented with an "∧ Error" message and will be returned to the prompt. Note that DR6 is a read only register and thus cannot be modified. For full details on how to use the debug registers, please refer to the Intel processor architecture manuals. The debug registers are enumerated as follows:

DR0	0
DR1	1
DR2	2
DR3	3
DR7	6

The HEXADECIMAL CALCULATOR command

This command allows the programmer to quickly work out the sum and difference of two hexadecimal values. The programmer uses this command in the following way:

- h [First number, 16] [Second Number, 16]

The IN PORT command

This command allows the programmer to read a byte from a byte sized I/O port. This will be expanded in future versions to allow for word and dword reads. This command is used as follows:

- i [I/O port to read, 4]

The program then prints the read byte value.

The OUT PORT command

This command allows the programmer to write a byte to an I/O port. This will be expanded in future versions to allow for word and dword writes. This command is used as follows:

- o [I/O port to write, 4]

The programmer is then prompted with a “ . ” after which they must type in a byte to send to the I/O port. The byte is sent when the programmer presses the Carriage Return key. The programmer may also return to the prompt by pressing “q”.

The VERSION command

This command allows the programmer to view the version of the program.

The PRINT MEMORY command

This program can be used by a programmer to see the System Memory Map. This is the SCP/BIOS Memory Map, which is built from the “so-called” E820H memory map, but has an entry for SCP/BIOS too. This program is entered by striking the “m” key. The programmer will then be prompted with the first entry in the memory table. The programmer can then strike any key to get the next entry and continue so on until they return to the prompt. This command cannot be quit until all entries have been displayed. Each entry is split into three fields as follows:

Start of Memory Region	Size of Memory Region	Memory Region Status

Each field is a qword. If the Size of Memory Region is 0, then the entry should be ignored. There are five types of Memory Region Statuses:

- 1 - Free
- 2 - Reserved
- 3 - ACPI Reclaimable
- 4 - ACPI Reserved
- 5 - Bad memory region

Any other values in the low dword of this entry should be considered as reserved and unusable. Bit 0 of the high dword of each Memory Region Status qword may also be set to 1. Not all systems support this though for those that do, this bit indicates that the entry is valid. If some entries have this bit set and some don't, those entries without this bit set, should be considered as unusable. Bit 1 may also be set, as this indicates that this memory is non-volatile. The details of these bits are outlined in the ACPI 3.0 standard.

Note further that at this time, SCP/BIOS does not order the entries of you and so, SCP/BIOS may appear as its own entry at the end of the memory map or correctly inserted in its correct location. This will be modified in future versions so that all entries are ordered in order of start addresses.

The CONNECT DEBUGGER command

This command "connects" the debugger to the system, in effect, hooking INT 1H, INT 3H and INT 3BH to allow the user to break into the debugger using the CTRL+BREAK key combination. This allows the programmer to non-programmatically connect the debugger without having to add special code in the program they are testing to allow them to break into the debugger upon hitting a breakpoint or a CTRL+BREAK. Please note that at this time, the original addresses of these handlers are not saved and are in effect overridden.

The DISCONNECT DEBUGGER command

This command “disconnects” the debugger from the system, in effect, returning the default interrupt handlers to INT 1H, INT 3H and INT 3BH. This means that INT 1H and INT 3H will trigger default exception screens henceforth (such as if a breakpoint is hit or a single step is attempted), and CTRL+BREAK will do nothing.

Linking SYSDEBUG with an application program

SYSDBEUG has uses a single interrupt, INT 40H, through which control can be returned to a calling application. The Quit command calls INT 40H to return back control back to the target address. After system initialisation, INT 40H contains the address to SYSDEBUG itself. It is recommended that a user application hooks INT 40H before beginning to use SYSDEBUG.

Appendix D: Confirmed Supported Hardware Configurations

All the following systems meet the minimum system requirements.

- Desktop:
  - Motherboard: Asus P8Z77-V LX motherboard
  - CPU: Intel Core i5-2500K CPU at 3.30GHz
  - Physical System Memory: 12GB
  - System BIOS Version/Date: American Megatrends Inc. 2501, 21/07/2014
  - SMBIOS Version: 2.7
  - Boot mode: Legacy
  - Keyboard: IBM Model M with PS/2 connector
- Laptop: Dell Studio XPS 1647
- Emulator: Bochs
- Virtual Machine: Oracle VirtualBox with Extensions (for EHCI support).

Appendix E: CPU Exception Reference

For a full reference list of the CPU Exceptions and how to handle them appropriately, please reference Volume 3A: System Programming Guide, Part 1 of the Intel 64 and IA-32 Architectures Software Developer's Manual. A brief table based on Table 6-1 in the aforementioned reference is presented below:

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Device by Zero	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/Trap	No	Instruction, data and I/O breakpoints; sing-step; and others.
2	-	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3H instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	Bound instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (Zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9	-	Coprocessor Segment Overrurn (Reserved)	Fault	No	Floating-point instruction. Not generated on SCP/BIOS supported processors.
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.
15	-	Intel reserved. Do not use.	-	No	-
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions.
20	#VE	Virtualisation Exception	Fault	No	EPT violations.
21	#CP	Control Protection Exception	Fault	Yes	RET, IRET, RSTORSSP and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect jump.
22-31	-	Intel reserved. Do not use.	-	-	-