BIOS Documentation

Version 1.0

General Properties and Memory Setup

Boot Sequence

The bios starts with a boot sequence located at 0c8000 0000 0000 0000. It sets up the Trap handler registers rTT (dynamic traps) and rT (forced traps). It calls a subroutine (memory) to initialize rV and, if needed, the page tables.
If the CPU simulator or a debugger has loaded a user program, it should set the following registers:

- Set rXX to #FB0000FF = UNSAVE $255. The user program is then started with a RESUME 1 operation after preparing $255 to contain the initial value (-1) for rK.
- Set rBB to the address in the stack segment where UNSAVE will find its data. The RESUME 1 operation will copy rBB to $255. Hence, rBB must be initialized to contain the appropriate value.
- Set rWW to the entry point in the main program, that's where the program continues after the UNSAVE.

If no program is loaded, rXX will be 0, that is TRAP 0,Halt,0 and it ends the program, before it has started, in the Halt trap handler.

Hardware

This Bios assumes the following physical device addresses and interrupt numbers:

Device	Start Address	Interrupt
ROM	0x0000 0000 0000 0000	
RAM	0x0000 0001 0000 0000	
FLASH	0x0000 0002 0000 0000	
Keyboard	0x0001 0000 0000 0000	40
Screen	0x0001 0000 0000 0008	41
Mouse	0x0001 0000 0000 0010	42
GPU	0x0001 0000 0000 0020	43
Video Ram	0x0002 0000 0000 0000	
Timer	0x0001 0000 0000 0060	44
Serial	0x0001 0000 0000 0080	45 (In) 46 (Out)
Sevensegement	0x0001 0000 0000 0090	
Unused	0x0001 0000 0000 0098	
LED 0	0x0001 0000 0000 00B0	
LED 1	0x0001 0000 0000 00B8	
LED 2	0x0001 0000 0000 00C0	
LED 3	0x0001 0000 0000 00C8	
Disk	0x0001 0000 0000 00D0	47

Button 0		48
Button 1		49
Button 2		50
Button 3		51

Page Tables

The page tables are kept in ROM, so there is no way of altering the tables at run time. Consequently, page faults can not be serviced properly.
The memory is organized in pages of 8 kByte.
The first page in RAM is not mapped into user space, it is reserved for the bios.

The user TEXT segment has 12 pages (96kB) mapping virtual addresses from 0x0 to 0x17FFF to physical addresses from #0000 0001 0000 2000 to #0000 0001 0001 9FFF

The user DATA segment has 8 pages (64 kByte) of RAM, mapping virtual addresses from 0x2000 0000 0000 0000 to 0x2000 0000 0000 FFFF to physical addresses from #0000 0001 0001 a000 to #0000 0001 0002 9FFF. Another 8 pages (64 kByte) are mapped to FLASH, a kind of persistent RAM, mapping virtual addresses from 0x2000 0000 0001 0000 to 0x2000 0000 0001 FFFF to physical addresses from #0000 0002 0000 0000 to #0000 0002 0001 FFFF.

The user POOL segment has 2 pages (16 kByte) mapping virtual addresses from 0x4000 0000 0000 0000 to 0x4000 0000 0000 3FFF to physical addresses from #0000 0001 0002 a000 to #0000 0001 0002 DFFF

The user STACK segment has 12 pages, 10 (80 kByte) at the upper and 2 at the lower end. The lower end mapping virtual addresses from 0x6000 0000 0000 0000 to 0x6000 0000 0001 3FFF to physical addresses from #0000 0001 0002 e000 to #0000 0001 0004 1FFF is used for the MMIX register stack (growing upward). The upper end mapping virtual addresses from 0x6000 0000 007F C000 to 0x6000 0000 007F FFFF to physical addresses from #0000 0001 0004 2000 to #0000 0001 0004 5FFF is used by the GCC compiler for its memory stack (growing downward).

Utility Functions

There are two utility functions: KeyboardC and ScreenC.

KeyboardC

Reads one character from the keyboard and returns the character. In case of errors (negative status) or if no data is available (character count is zero) it will power down the processor with a SYNC 4 instruction, and waits until the keyboard interrupt occurs.

ScreenC

There are two versions of this function, one for output to the screen device and one for output to the winvram device using the GPU.
The procedure takes one parameter: in $0 is an ASCII character code that is to be displayed on the screen. Both implementations are very simple using no buffering. They just rely on the fact that the devices are fast.

Version one sends it to the screen device by storing it into the output byte location. Version two stores it to the Screen using the GPU. The GPU command to put a character on the screen, advances the current location automatically. All that is needed is adding a 0x01 in the high byte of a TETRA containing the character and store it to the command TETRA of the GPU. In case the character was a carriage return (0x0D) a following line feed character (0x0A) is added. This accounts for windows style input, where typing the return key, just produces a carriage return character.

Dynamic Traps

During the boot sequence the register rTT is set to the dynamic trap handler DTrap. It saves all local registers jumping to DTrap:Handler. After return from the DHandler, it restores rJ, sets up $255 for rK and returns to the user program.

The DHandler inspects rQ, to find the lowest bit set, resets it to zero and jumps through a jumptable to the specific trap handler. Most entries in the jump table point to d default handler for unhandled traps. The specific trap handlers, must not alter any global registers. They can alter local registers and must return with a POP 0,0 instruction.

Unhandled Traps

Informs the debugger with a SYMW 5 instruction and returns.

Keyboard Interrupt

Reads the status and data OCTA from the keyboard. In case of errors (negative status) returns immediately. If no data available (character count is zero) return immediately. Otherwise, the available character is echoed to the screen. There is no buffering for a subsequent read operation.

Forced Traps

Forced traps are mapped to the FTrap routine using the rT register. The FTrap routine saves the local registers by calling the FTrap:Handler and will restore rJ and rK (through $255) before resuming. .Forced traps come in different flavors; the most common is caused by a TRAP instruction, indicated by a negative ROP-Code and the instruction byte in rXX being zero (the TRAP opcode is zero). The others are either ignored or just don't happen (Emulate Instruction, Page table translation in software). For TRAP XYZ instructions, the Y byte contains the function code. The handler extracts it and uses it to access a jump table. Most values are handled by the routine FTrap:Unhandled.
Individual trap handlers can be inserted in the jump table. The trap handlers can inspect rXX, for example, to find out about the Z Byte, the value of rZZ to find the value of $Z, or the value of rBB, which contains the value of $255 at the time when the TRAP instructions was executed. It is the common parameter register for TRAP instructions. Results should be stored back into rBB, such that the final RESUME 1 instruction will place them in $255 for the user program. The routines must end with a POP 0,0 to get back to the FTrap routine. The MMIXware book has a sequence of predefined TRAP instructions (Halt, Fopen, …, Fseek, Ftell). Not all of them are implemented in this bios and the remaining ones are only partly implemented, since there is no file I/O. There are, however, some other TRAP routines available that allow access to the connected hardware.

Halt (0x00)

The Halt Trap informs the debugger, enables interrupts, and goes into an endless idle loop. Executing a Trap 0,Halt,0 instruction will never return to the user program. The CPU will however service dynamic interrupts. This can be appropriate, if there is no user program and the only purpose of the bios is to serve interrupts.

Fopen (0x01)

Not implemented. The debugger is informed on this before returning.

Fclose (0x02)

Not implemented. The debugger is informed on this before returning.

Fread (0x03)

Not implemented. The debugger is informed on this before returning.

Fgets (0x04)

See MMIXware. The Trap is implemented only for StdIn., reading characters from the keyboard with echoing to the screen.

Fgetws (0x05)

Not implemented. The debugger is informed on this before returning.

Fwrite (0x06)

See MMIXware. The Trap is implemented only for StdOut and StdErr, writing characters to the screen.

Fputs (0x07)

See MMIXware. The Trap is implemented only for StdOut and StdErr, writing characters to the screen.

Fputws (0x08)

Not implemented. The debugger is informed on this before returning.

Fseek (0x09)

Not implemented. The debugger is informed on this before returning.

Ftell (0x0a)

Not implemented. The debugger is informed on this before returning.

TWait (0x10)

Register $255 should contain a time in milliseconds. The Trap will wait until the given time has elapsed and then will return.

TDate (0x11)

Returns in $255 the current date in the following 8 Byte Format: YYYY MM DW,
Where YYYY is the current year, M is the current month (0 to 11), D is the day in the month (1 to 31) and W is the day in the week (0-6).

TTimeOfDay (0x12)

Returns in $255 the number of milliseconds since midnight.

VPut (0x20)

Expects in $255 the following data: 0xII 0xRR 0xGG 0xBB 0xXXXX XXXX
(II byte ignored, RR red color value byte, GG green color value byte, BB blue color value byte, XXXXXXXX 32-bit offset into the video RAM) It will store the high TETRA into the Video RAM using the low TETRA as offset. The Video RAM is a two dimensional array of TETRAs, each TETRA representing the RGB value of one pixel. The Pixels are stored line by line. The dimensions of the array are configuration dependent.

VGet (0x21)

Expects in $255 a 32-bit offset into the video RAM. It will read one TETRA from the video RAM at the given offset and return this value. The value is the RGB value of the corresponding Pixel. (see VPut above)

GSize (0x22)

Return the size of Screen and Framebuffer as four WYDEs in $255 using the format: FRAMEWIDTH, FRAMEHEIGHT, SCREENWIDTH, SCREENHEIGHT. The screen dimensions, given in the medium-low and low WYDE, describe the size of the visible part of the frame buffer; the frame buffer dimensions in the high and medium-high WYDES give the full dimension including the off-screen memory. These dimensions can be used to convert (x,y) pixel coordinates to video ram offsets using the formula offset=4*(x + y*FRAMEWIDTH).

GSetWH (0x23)

The trap expects in the low TETRA of $255 two WYDEs, the width and the height. These values are used for drawing Rectangles and Bitmaps (see below).

GSetPos (0x24)

The trap expects in the low TETRA of $255 two WYDEs, the x and the y coordinate. These values are the new current position used for the next drawing operations.

GSetTextColor (0x25)

The trap expects in the high TETRA of $255 the new text background color and in the low TETRA of $255 the new text foreground color. In each TETRA the highest byte is ignored and the three lower order byte contain the red green and blue value. The new colors are used for all following text output operations (including Fputs).

GSetFillColor (0x26)

The trap expects in the low TETRA of $255 the new fill color. In the TETRA the highest byte is ignored and the three lower order byte contain the red green and blue value. The new color is used for all following fill operations (e.g. rectangles).

GSetLineColor (0x27)

The trap expects in the low TETRA of $255 the new line color. In the TETRA the highest byte is ignored and the three lower order byte contain the red green and blue value. The new color is used for all following line drawing operations.

GPutPixel (0x28)

Expects in $255 the following data: 0xII 0xRR 0xGG 0xBB 0xXXXX 0xYYYY
(II byte ignored, RR red color value byte, GG green color value byte, BB blue color value byte, XXXX x coordinate as two byte value (range 0 to 639), YYYY y coordinate as two byte value (range 0 to 479)). Coordinates start in the top left corner with (0,0) and extend to bottom right. The Pixel with the given coordinates receives the given color.

GPutChar (0x29)

The trap expects in $255 the following data: 0x000000 0xCC 0xXXXX 0xYYYY
The high TETRA contains the ASCII value of a character, padded to the left with three zero byte. The low TETRA consists of two WYDEs, one for the x and the other for the y coordinate. The given character is put at the given position using the current text and text-background colors. The current position is updated to the position exactly after the character written.

GPutStr (0x2A)

The trap expects in $255 the address of a zero terminated string. It is written to the screen starting at the current position using the current text and text-background colors. The current position is updated to the position exactly after the string.

GLine (0x2B)

The trap expects in the low TETRA of $255 two WYDEs, the x and the y coordinate. In the low WYDE of the high TETRA of $255, the trap expects the line width. If the width is zero, a default width of 1 pixel is used.
A line is drawn from the current position to the position given by x and y. The current position is then updated to the end of the line, such that successive calls will produce joined lines.

GRectangle (0x2C)

The trap expects in the high TETRA of $255 two WYDES, the width and height, and in the low TETRA of $255 two WYDEs, the x and the y coordinate of the top left corner of the rectangle. The Trap draws a rectangle using the current fill color. No update of the current position is performed.

GBitBlt (0x2D)

This trap facilitates a bit block transfer from video ram to video ram. It is entirely handled within the GPU and therefore very fast. The trap expects in $255 the address of two OCTAs. The first OCTA (must be OCTA aligned) consists of four WYDE values: 0xWWWW 0xHHHH 0xXXXX 0xYYYY. These values indicate width, height and destination coordinates x and y of the destination rectangle. The current position will be changed to match the new x, y position.
From the second OCTA (at address $255+8) only the first (high) TETRA is used. It contains two WYDEs: the x and the y coordinate of the source bitmap. A rectangular region of the screen is copied from the source rectangle to the destination rectangle.

GBitBltIn (0x2E)

This trap facilitates a bit block transfer from memory into video ram. The trap expects in $255 the address of two OCTAs. The first OCTA (must be OCTA aligned) consists of four WYDE values: 0xWWWW 0xHHHH 0xXXXX 0xYYYY. These values indicate the with and height of the bitmap in main memory and the destination coordinates x and y on the screen the upper left corner of the bitmap will appear at this position. The current position will be changed to match this new position. The second OCTA (at address $255+8) contains the memory address of the bitmap data. It is a sequence of width*height TETRAS, where each TETRA contains one color value in the format 0x00RRGGBB. The whole bitmap is displayed on the screen.

GBitBltOut (0x2F)

This trap facilitates a bit block transfer from video ram to memory. The trap expects in $255 the address of two OCTAs. The first OCTA (must be OCTA aligned) consists of four WYDE values: 0xWWWW 0xHHHH 0xXXXX 0xYYYY. These values indicate the with and height of the bitmap and its coordinates x and y in video ram. The current position will be changed to match this new position. The second OCTA (at address $255+8) contains the destination address of the bitmap data in ram. It will be filled with a sequence of width*height TETRAS, where each TETRA contains one color value in the format 0x00RRGGBB from one source pixel on the screen.

MWait (0x30)

The routine waits until a mouse interrupt occurs. It then returns in $255 the following four WYDE values: 0xBBBB 0xEEEE 0xXXXX 0xYYYY

0xBBBB is the Button status indicating which buttons were down when the interrupt was triggered. It is a combination of these values:

Name	Value	Remark
MK_LBUTTON	0x01	The left mouse button is down.
MK_MBUTTON	0x10	The middle mouse button is down.
MK_RBUTTON	0x02	The right mouse button is down.
MK_SHIFT	0x04	The SHIFT key is down.
MK_CONTROL	0x08	The CTRL key is down.

0xEEEE is the event status indicating what triggered the interrupt. It is one of the following values:

Name	Value	Remark
MOUSEMOVE	0x80	
LBUTTONDOWN	MK_LBUTTON | MK_SHIFT	
LBUTTONUP	MK_LBUTTON	
LBUTTONDBLCLK	MK_LBUTTON | MK_SHIFT | MK_CONTROL	
RBUTTONDOWN	MK_RBUTTON | MK_SHIFT	
RBUTTONUP	MK_RBUTTON	
RBUTTONDBLCLK	MK_RBUTTON | MK_SHIFT | MK_CONTROL	
MBUTTONDOWN	MK_MBUTTON | MK_SHIFT	
MBUTTONUP	MK_MBUTTON	
MBUTTONDBLCLK	MK_MBUTTON | MK_SHIFT | MK_CONTROL	

Mouse movement will cause interrupts only if enabled for the mouse device. For the three buttons, a down event is typically first, followed by an up event. If the next down event comes fast enough, in addition to the bit indicating the button and the shift-bit indicating the down event also the control-bit is set to indicate a double click. Application usually start an action with the first down or up event and modify the continuation when receiving the double click event.

0xXXXX and 0xYYYY are two 16 bit values indicating the mouse coordinates where the event happened.

BWait (0x40)

Wait for the Button to be pressed. Return immediately if the button was already pressed.

SSet (0x50)

Set the LED's of a 7-Segment Display using Register $255.

Each of the eight digits (including the dot to the left of the digit) corresponds to one of the eight byte of register $255. The leftmost digit corresponds to the most significant byte and the right most digit to the least significant byte. Within a byte, every bit corresponds to one LED of the corresponding digit according to this Table:

Bit	Hex-Value	LED
0	0x01	Top-Horizontal
1	0x02	Middle-Horizontal
2	0x04	Bottom-Horizontal
3	0x08	Top-Left-Vertical
4	0x10	Bottom-Left-Vertical
5	0x20	Top-Right-Vertical
6	0x40	Bottom-Right-Vertical
7	0x80	Bottom-Right-Dot

Setting a bit to 1 will illuminate the LED; setting a bit to 0 will turn the LED off. Multiple LEDs can be set by setting multiple bits.

SDecimal (0x51)

$255 specifies the number to display in decimal. Z is the position (from the right) where the dot is displayed. If Z is zero, no dot will be shown.