Lion CPU/Computer Technical Documentation

# INTRODUCTION

This is a project started at 2005 as fpga/vhdl learning experiment and became a 32 bit computer with graphics, sound and software like a c compiler and games.

At the beginning I made a half working 8bit cpu with composite b&w video but I soon redesigned this to 16bit cpu with vga output and 16 colors and there I added the most parts that the 32bit version also shares.

I built a video controller, a UART, wrote an assembler and ported PA Tiny Basic adding fixed point and more. I added SPI to use a SD Card and wrote simple FAT system support. For sound I made 3 audio channels and a noise channel.

Then I added hardware sprites.

I made a generator for Lion CPU code for the Java Grinder open source project and so I have a simple Java Compiler for Lion. Then I wrote "Astro", a simple demo game in Java. Of course I have added joysticks.

Added Vector graphics!

A new 32-bit Lion is now completed with 256 colors, sampled audio playback, more sprites and all software is upgraded to 32bits, plus a couple of C compilers.

To this project I enjoy keep returning between my other projects.

Theodoulos Liontakis

Contents

[INTRODUCTION 1](#_Toc123773389)

[Lion CPU 4](#_Toc123773390)

[Overview 4](#_Toc123773391)

[Hardware Signals of Lion cpu. 4](#_Toc123773392)

[SR register bits: 5](#_Toc123773393)

[Addressing modes 5](#_Toc123773394)

[Instructions 6](#_Toc123773395)

[Data movement instructions 7](#_Toc123773396)

[Arithmetic instructions 8](#_Toc123773397)

[Logic instructions 9](#_Toc123773398)

[Test and branch instructions 9](#_Toc123773399)

[Loop instructions 10](#_Toc123773400)

[Stack related instructions 11](#_Toc123773401)

[Interrupt related instructions 11](#_Toc123773402)

[I/O instructions 12](#_Toc123773403)

[Miscellaneous instructions 12](#_Toc123773404)

[Block move & block fill instructions 12](#_Toc123773405)

[Assembler directives 13](#_Toc123773406)

[Lion Computer 14](#_Toc123773407)

[Memory 14](#_Toc123773408)

[Graphics 14](#_Toc123773409)

[Video Mode 0 15](#_Toc123773410)

[Video Mode 1 15](#_Toc123773411)

[Sprites 16](#_Toc123773412)

[Vector Graphics 17](#_Toc123773413)

[Sound 18](#_Toc123773414)

[Tone channels and noise 18](#_Toc123773415)

[Sampled Audio 19](#_Toc123773416)

[Keyboard 20](#_Toc123773417)

[Joystick 20](#_Toc123773418)

[Serial ports 20](#_Toc123773419)

[Storage - SD Card 21](#_Toc123773420)

[Miscellaneous 22](#_Toc123773421)

[System function Calls - Software Interrupts 22](#_Toc123773422)

[Tiny Basic 27](#_Toc123773423)

[TB as Lion’s command interpreter 30](#_Toc123773424)

[Appendices 31](#_Toc123773425)

[System I/O ports 31](#_Toc123773426)

[System variables 32](#_Toc123773427)

[Music notes to clock divider table 33](#_Toc123773428)

# Lion CPU

## Overview

I didn’t study modern cpu design so it’s not an advanced cpu, it reflects my experiences with the 80’s cpus in the computers I used when I was young. This cpu was not designed as a 32bit from the start and I kept it’s command size at 16bit for code compactness but everything else is 32bit, the registers, data transfers, the logic/math operations, data and address bus are all 32bit.

So at the beginning of each cycle it fetches a word from memory, decodes it and executes it. In many cases it can in parallel begin the fetching of the new word at the end of the execution of the previous command.

It has 8 general registers A0 to A7. Also it has a loop index register IDX, a status register SR, a user stack pointer SP, a system stack pointer ISP and the program counter PC. All registers are 32bit except SR which is 16bit.

It has an IO signal and IO commands for moving data on an IO system bus.

It supports 4 hardware interrupts that can be disabled.

It can do addition, subtraction, multiplication and all the logic operations and bit set/clear operations and comparisons.

Execution starts at address 0x40, addresses 0-64 contain a 16 entry trap addresses table for the 4 hardware and 12 software interrupts.

User stack (SP) is initially set at 0x1FFFC and system stack (ISP) for interrupt routines and system software is set at 0x041FC (because I like it there ☺ ).

## Hardware Signals of Lion cpu.

Line RESET resets the CPU when LOW.

Line Clock is the clock 50MHz .

Lines A0-A31 are the address bus

Lines D0-31 are the data bus

Line RD can be used to insert wait states when driven low.

Line HOLD active low releases the buses after finishing the current instruction and CPU sets HOLDA to acknowledge.

Line AS is address strobe and DS is the data strobe (active low) they indicate valid data on the address and data lines.

Line RW when high indicate a read cycle and when low a write.

Line IO indicates an input output operation.

Line INT when low sets the interrupt, input lines I1 and I0 define the interrupt vector and lines IACK, IA1, IA0 acknowledge the service of the interrupt.

Line BACS when high indicates a byte access.

Line WACS when high indicates a word access.

## SR register bits:

(as with all registers, SR(0) is the rightmost less significant bit )

SR(0) = Carry flag

SR(1) = Overflow flag

SR(2) = Zero flag

SR(3) = Negative flag

SR(4) = JXA loop idx increase or decrease register

SR(5) = TRAP Interrupt 15 on/off (off=0 the default value)

SR(6) = Interrupt disable (default disabled)

## Addressing modes

It has the basic addressing modes plus relative addressing

General form: Operation Destination, Source

**Register direct**

MOV.D A0,A2

**Immediate value**

MOV.D A0,100 ; 100 is the value that will be moved in A0

CMP.B A1,13

**Immediate small value** 0..15 only 32bit operations

MOVI A0,0

ADDI A1,1

CMPI A1,0

**Register, address indirect**

MOV.D (A0),(A2)

MOV.W (1024),A0 ; Moves word from A0 to address 1024

ADD.D A0,(100) ; Moves double word from address 100 to A0

**Register indirect with displacement**

MOV.D A0,100(A1)

MOV 10(A1),0

**Direct with stack pointer**

MOV SP,A1

**Relative addressing**

MOVR.D A1,(100) ; moves the contents of the address that is 100 ; bytes offset from the current address

JR LABEL1

**Get effective address**

GADR A1, label ; calculates at runtime the address of label in A1

## Instructions

Instructions must be aligned to even addresses (the assembler takes care of this).

Many Instructions have 3 sizes

8bit Byte .B extension

16bit Word no extension (because the cpu started as 16 bit)

32bit Double Word .D extension

**NOP** does nothing, after lot of debugging it works fine.

### Data movement instructions

Move instructions **don’t** affect the flags (SR)

**MOV or MOV.L** moves a 16bit/32bit word and has the form

MOV destination, source

MOV Ax|(Ax)|(address) , Ax|address|(Ax)|(address)

Source can be a 16bit/32bit number or a register or content of address constant or contents of address stored in a register (indirect).

Destination can be a register, or address/register indirect .

**MOV.B** moves the rightmost byte of source to rightmost byte of destination when it is a register or to the address specified.

examples: MOV A0,$100 MOV.B (A0),A1 MOV (A1),(A0)

MOV A0,($FF00) MOV (100),A2 MOV (100),13

**MOVI** is used to store a small number 0-15 to a 32bit register is smaller and faster.

ex. MOVI A0,9

**XCHG** Ax,Ax exchanges values of two registers

**MOVHL** **MOVHL.D** moves the low byte/word of source to the high byte/word of destinatio.

**MOVLH** **MOVLH.D** moves the high byte/word of source to the low byte/word of destination.

**MOVHH MOVHH.D** moves the high byte/word of source to the high byte/word of destination.

**MOVR.D, MOVR, MOVR.B** relative address move and **GADR** get absolute address instructions work together with relative jumps to produce relocatable code.

**MOVR.D** A1,(Label1) Where the second operand is the calculated, by the assembler, offset to the label Label1.

**MOVR.D** (Label),A0

**GADR** A1,Label gets the runtime calculated absolute address of Label.

### Arithmetic instructions

All arithmetic and logic instructions affect the flags (SR)

**ADD ADD.D ADD.B ADC ADC.D**  add without and with carry

**SUB SUB.D SUB.B** subtraction

**MULU.D, MULU.B** unsigned multiplication

**ADDI SUBBI** quick add/subtract a small number 0-15 (32bit operation)

Destination a register or an address.

**INC INC.B DEC DEC.B** Increase decrease value

result is stored in destination (first register)

examples: ADD A1,(100) MUL.B A0,10 SUB A3,(A2) INC A0

DEC (COUNTER)

Division implemented only by software (signed INT 4 with A0=9, unsigned INT5,A0=6) with assist of the specially for this purpose SRLL, SLLL and ALGN added commands.

### Logic instructions

Logic instructions affect the flags (SR)

Destination always a register, source a constant.

**AND OR NOT XOR AND.B OR.B XOR.B NOT.B**

**SRL SLL SRL.B SLL.B** logical shift, SLL Ax,n shifts Ax n bits

**SRA SLA** Arithmetic Shift

**ROL** Rotate

ex. SLL A0,3 Shifts Left Logically A0 register 3 bit positions

**SLLL SRLL SLLL.D SRLL.D** Ax,Ax logical shift by 1 place a register pair

### Test and branch instructions

Compare instructions affect the flags (SR)

**CMP.D CMP CMP.B** compare register or constant or value of address in a register or constant and set the flags

**CMPI CMPI.B** Reg,0-15 compares register with small number, smaller and faster than CMP.

examples CMP A4,A1 CMP A0,10 CMP A1,(100) CMP A0,(A3) CMP (100),A2 CMPI A0,2

**BTST** Ax,(n,An) Tests nth bit of register Ax and sets the zero flag

**JMP JR** Jump to an absolute or relative address

ex. JMP (A1) JR A0 JMP $FF7E JR 8

**JSR JRSR** Jump to subroutine pushes next address to the user stack

**RET** Return from subroutine restores PC from user stack

**IJSR** Jump to subroutine pushes next address to the system stack

**IRET** Return from subroutine restores PC from system stack

**JZ JE JNZ, JRZ, JRNZ** Jump depending on zero flag (absolute and relative)

**JC JNC JRC JRNC** Jump depending on carry flag

**JO JNO JRO JRNO** Jump depending on overflow flag

**JN JP JRN** **JRP** Jump depending if negative or positive

**JBE JA JAE JRBE JRA** Jump below or equal, above, above or equal for unsigned values

**JLE JRLE JG JGE JRG JRLE JRG JRGE** as above for signed values

### Loop instructions

**SETX** Sets IDX index register to a Value, register, indirect value/reg.

**JMPX JRX** Jump if not zero and decrease IDX

example:

SETX 3

loop: SRL A0,1

JMPX loop

This will loop 4 times dividing A0 by 16

**JXAB JXAW JRXAB JRXAW** same as JMPX JRX but if IDX is non zero increases a specified register to act as pointer to a byte or word

*example:*

SETX 99

MOV A2,address1

loop: MOV.B (A2),0

JXAB A2,loop

This fills 100 bytes starting at address1 with zeros.

**MOVX** Ax moves the value of IDX to register Ax

### Stack related instructions

For user stack:

**PUSH** pushes Ax or SR or (Ax) to the stack decreasing SP by 2

**POP** pops AX or SR from the stack increasing SP by 2

**PUSHX POPX** pushes/pops IDX

**SETSP** Ax|address|(Ax)|(address) Sets the user stack pointer to a new address, MOV SP,AX is also valid

**GETSP** Ax Stores sp value in Ax,  MOV Ax,SP is also valid

**ADD** SP,Ax

**SUB** SP,Ax

For system stack:

**PUSHI, POPI**, **PUSHXI, POPXI** are the instructions for system stack.

**SETISP** Ax|address|(Ax)|(address) Sets the system stack pointer to a new address

**GETISP** Ax Stores isp value in Ax

### Interrupt related instructions

**INT n** Jumps to the nth interrupt vector pushing PC and SR

**RETI** Returns from interrupt restoring SR and PC

**STI** Enables interrupts

**CLI** Disable interrupts

When a hardware interrupt occurs interrupts are disabled by default until the end of service or code enables it.

### I/O instructions

**IN** Ax, address|Ax|(Ax) inputs a word

**OUT**, **OUT.B** address|Ax , value|Ax|(Ax) outputs a word, byte

### Miscellaneous instructions

**SWAP** Ax swaps low and high bytes in a register

**SWAP.D** swaps low and high words in a register

**BSET** Ax,(n|Ax) **BCLR** Ax,(n|Ax) Sets/clears the nth bit of Ax

**SRSET** n, **SRCLR** n sets, clears the nth bit of status register

**SEX** Ax, **SEX.B** Ax sign extend word to double, byte to double

**ALNG** Ax1,Ax2 rotate right until the 31th bit of Ax1 is a 1,returns number of shifts required in Ax2.

### Block move & block fill instructions

The IDX register that is used as a loop counter and index, is used by the transfer instructions to hold the number of words to be moved.

Instructions **MTOM, MTOI, ITOM, ITOI** with two arguments An1,An2 move fast blocks of words starting from memory or IO address An2 to memory or IO address An1. Instructions **NTOM, NTOI** with arguments An1,N or An1,An2 fill the block pointed by An1 of size IDX+1 with the 16 bit value N (or An2).

Also there are the byte block move instructions MTOM.B MTOI.B ITOM.B ITOI.B

MOV A1,buffer1

MOV A2,buffer2

SETX 99

MTOM A1,A2 ;copy the 100 words block starting at buffer2 to buffer1

SETX 99

NTOM A2,0 ;fill buffer2 with zeros

## Assembler directives

**ORG** address Sets the address that code that follows it will start from

[name] **DW** w1,w2,...wn (auto align in even address)

[name] **DB** b1.b2...bn

[name] **DS** count in bytes

[name] **TEXT** "some text"

[name] **DA** (address constant, also auto aligned)

# Lion Computer

## Memory

Currently only 20 of the 32 address bits are used for system ram so 1MB address space is used but it’s easy to adjust the system to use as many bits up to 32 as needed in a newer version of pcb.

This is the memory map:

0x00000 – 0x00040 Interrupt addresses table

0x00040 – 0x03FFF 16K System rom

0x04000 – 0x04FFF System variables, buffers, file tables, system stack

0x05000 - 0xFFFFF Free system ram

As it is implemented the first 128K of memory reside in the fpga and have 1 cycle 32bit access time. The rest ram resides in an external IC and because of lack of free IO pins at the fpga board a 16bit IC is used and data has to be multiplexed so 32bit access takes 3 to 4 cycles.

There is also memory mapped in the IO bus

0x00000 - 0x03FFF is used for various devices interface

0x04000 - 0x07FFF 4X4K blocks of sprite data ram

0x08000 - 0x17FFF this is the 64K video ram

0x18000 - 0x1FFFF 16K of vector/digital audio buffer ram

## Graphics

Lion has 64K of 16bit dual bus video memory so it won’t slow the cpu when video is generated and has 2 video modes, one more text oriented and a graphics oriented. Using the MTOI, NTOI block transfer cpu instructions fast data transfers can be implemented to video ram.

Video Mode 0 : 640x240 pixels, 30x80 chars, 16 colors (2 per char)

To choose video mode 0 you can use the command *mode 0* from basic or do an *out 24,0*

It uses less than half of the 64K memory that is attached to the IO bus at address 0x08000 and it consists of 2 areas the pixel data and color data.

Pixel data is arranged in a bit per pixel way so it occupies a byte for every 8 pixels. For the mode 0 resolution of 640x240 pixels this is 19200 bytes starting at 0x08000.

Every byte defines 8 pixels vertically and the next byte defines the 8 vertical pixels to the right of it. The first 640 bytes define the first horizontal stripe of 8 pixels height. The following 640 bytes define the second horizontal stripe and it goes on for all 30 stripes of the screen.

Color in this mode is not defined per pixel but per 8x8 pixel tile.

There are 80x30 pixel tiles on the screen and a byte is needed for each one so 2400 bytes are used for color data starting at address 0x0EEE0

Each byte is divided in 2 nibbles, the higher 4 bits of the byte define the foreground color and the 4 least significant bytes the background color.

Video Mode 1 : 320x200 pixels, 25x53 chars, 256 colors

To choose video mode 1 you can use the command *mode 1* from basic or do an *out 24,1*

In this mode every pixel occupies 1 byte that contains the 8bit color of that pixel so it needs 320x200 = 64000 bytes starting at address 0x08000.

Pixel data is arranged in the same way of a crt display raster scan, it begins at upper left and moves in a horizontal a line to the right and then to the left of the next line for all 200 lines.

Color in the byte has the form:

RRRGGGBB It has 3 bits for green, 3 bits red and 2 bits for blue

## Sprites

Four sprite engines each with 4K dedicated ram that provide 14 sprites each give Lion 56 16x16 pixel double buffered hardware sprites.

Sprites have 15 colors + transparency and for each engine a palette can be chosen out of 8 variations.

Ram is mapped at the IO address space as follows

0x05000 - 0x05FFF 1st 4K block of sprite data ram

0x06000 - 0x06FFF 2nd 4K block of sprite data ram

0x07000 - 0x07FFF 3rd 4K block of sprite data ram

0x08000 - 0x08FFF 4th 4K block of sprite data ram

The first 256 bytes in every 4K block are parameter table for the first buffer of the sprites

The second 256 bytes in every 4K block are parameter table for the second buffer of the sprites

I reserved more space than needed but anyway.

The parameter table is divided in a 8 byte record per sprite as follows:

2 bytes X position 0..319

2 bytes Y position 0..239

3 bytes reserved

1 bytes disabled if 0 enabled if 1 (visible or not)

Sprite graphic data begins at offset 512 for the first buffer and at offset 2304 for the second buffer. Every sprite graphic data is 128 bytes long and they are sequentially stored.

Each byte in sprite graphic data contains color for 2 pixels 4 bits for each. So each sprite is made of 8 byte rows by 16 = 128 bytes

B0 B1 B2 B3 B4 B5 B6 B7

B8 B9 …

Color 0xF is the transparent color.

The switch from buffer 0 to buffer 1 for the sprites is controlled by writing a byte to IO address 0x14.

The 4 most significant bits control which parameter buffer is enabled, one bit for every sprite engine 0 = use buffer 0, 1= use buffer 1.

The 4 least significant bits control which data buffer is enabled, one bit for every sprite engine.

Palette for each engine is chosen by writing a word at IO address 40, in that word 4x4 bits define the 4 engines palettes.

## Vector Graphics

I like vector graphics and I added support to Lion for it. I added 3 double 16bit Digital to Analog Converter IC (DAC). The interface to the chips is a fast triple SPI so I can write in parallel to the 3 DACs at 20Mhz clock.

One of the outputs of each dac is connected to X, Y, Z outputs to a vector display or an oscilloscope. The other output is used for sampled audio playback.

There is a 16K buffer that is also shared between the vector graphics and sampled audio so these don’t work simultaneously.

The buffer at the IO Bus at address 0x18000 (98304) has serially arranged couples of words that define points on the vector display and brightness of the beam.

W1a W1b W2a W2b W3a W3b….

The 8 leftmost bits of the first word define the brightness of the beam Z=0..255.

The right 8 bits of the second word have the 8 most significant bits of X coordinate and the left 8 bits of the second word have the 8 most significant bits of Y coordinate. For more fine resolution one more bit, the least significant can be added for X at bit12 of the first word and for Y to bit8 of first word. So X and Y can be 9 bit numbers 0..512.

The vector controller reads 2 words every time and moves the beam to the point they define with the beam intensity defined, when buffer ends it starts from the beginning again. This repeats continually as long as vector graphics are enabled.

To obtain correct visual results the controller uses a line drawing algorithm in hardware so it illuminates every point between two sequentially defined points by following the line between them.

The buffer length can be defined shorter by writing a word at IO address 44.

The byte at IO address 30 controls the DAC (and sampled audio).

Setting bit1 to 0 selects vector mode of the DACs and output is enabled at X, Y, Z connectors.

Bit0 of the same byte when in the vector mode disables the line draw algorithm when set to 1 and then it depends on the behavior of the connected vector display how lines will appear.

## Sound

Lion has two ways to produce sound, one way is with the 3 tone generators and the noise generator and the other is by using dacs to play digitally sampled audio.

### Tone channels and noise

It has 3 tone generator channels that are controlled by writing a word at IO addresses 8, 10, 12

The 3 rightmost bits of the word define the duration:

111 = 4 sec, 110=2 sec, 101=1 sec, 100=0.5 sec …

The rest 13 bits define a divider of a 200Khz clock to produce the output frequency

There is a digital pwm in the high frequency that enables to control volume by writing a byte at IO addresses 25, 26, 27 for each channel.

Also for every channel there is the option to enable a harmonic to the basic frequency by writing a byte at IO addresses 31, 32, 33. The 4 least significant bits of this byte choose one of 16 different harmonics.

By reading IO address 9 one can determine if a channel still plays a tone or it has finished. The 3 least significant bits show if set which of the channels play a tone. The 4th ls bit shows if the noise channel is producing noise.

The noise channel uses as top limit frequency the frequency of a channel and produces noise using the LFSR method with a 25 bit shift register.

Enable noise with a channel by writing a byte at IO address 11, the 3 least significant bits when set enable noise when the corresponding channel plays a tone.

### Sampled Audio

Lion has a 16K buffer dedicated to the 3 DAC ICs that are used for either reproduction of 8bit sampled audio data or XY-Display. This buffer is placed at the IO Bus at address 0x18000 (98304).

Audio data in the buffer can be mono or stereo and many sample rates are supported. When stereo mode is selected data bytes from the 2 channels must be written alternately in the buffer.

The byte at IO address 30 controls the playback

Setting bit 1 to 1 selects audio sampled data mode of the DACs

Bit 2 selects stereo mode when set

When bit 0 is set to 1 playback is started, to restart clear bit 0 and set it to 1 again.

The starting and ending point in the buffer can be set by writing a word at IO addresses 43 and 44.

Playback rate is set by writing a word at the IO address 34, default value is 3600 that sets the rate to 8192 samples/sec, 1800 is 16384 s/sec.

Finally to determine if the playback has finished and more data needs to be written to the buffer the 4th bit of IO address 9 when set means that audio is still playing.

## Keyboard

A PS/2 keyboard interface is implemented for input, it has a hardware buffer of 16 bytes and it uses few IO addresses.

Unread data exists in the keyboard buffer when after reading a byte from IO address 6, bit2 is set.

Data when ready is read from IO address 14. To remove the entry from the buffer, write a 2 at IO address 15 and then a 0.

There are ready routines for the keyboard that we will discuss in the system calls section.

## Joystick

Lion has two Atari compatible joystick ports.

Their state can be read as a word from I/O port 22.

the lower byte has the state of the first joystick.

the higher byte has the state of the second joystick.

the byte has the form: 0 0 0 Right Left Up Down Button

## Serial ports

Lion has 2 serial ports.

The first serial port is also used by Tiny basic as input in parallel with the keyboard and for data transfer for easy testing by some development tools.

There are system calls to service the serial connections.

Serial ports have a 12 byte input buffer and an 8 byte output buffer.

For serial port 0 data is read from the buffer from IO address 4, to remove the entry from the buffer, write a 2 at IO address 2 and then a 0.

Unread data exists in the serial buffer when after reading a byte from IO address 6, bit1 is set.

To send a byte first one has to check bit 0 of IO ad. 6 to confirm the bit is 0 so the output buffer is not full. Then write the byte to send at IO ad. 0 and write a 1 at address 2 and then a 0 at IO address 2.

For serial port 2 data is read from the buffer from IO address 5, to remove the entry from the buffer, write a 2 at IO address 3 and then a 0.

Unread data exists in the serial buffer when after reading a byte from IO address 6, bit4 is set.

To send a byte first one has to check bit 3 of IO ad. 6 to confirm the bit is 0 so the output buffer is not full. Then write the byte to send at IO address 1 and write a 1 at address 3 and then a 0 at IO address 3.

Baud rate is set by writing a word at IO addresses 41 (port0), 42(port1). The word contains a divider of a 25Mhz clock and for 38400bps it is 1301.

## Storage - SD Card

For storage Lion has a SD Card connected through a SPI interface.

The file system for the card is FAT16 with up to 31MB size.

I won’t try to explain the hardware details here but there are system function calls to initialize the card and read and write blocks from it and for file handling, see the system function calls section.

## Miscellaneous

Hardware interrupt 2 can be enabled to occur in the start of every horizontal synchronization pulse. It is enabled by writing 1 at the IO address 13 and disabled by writing a 0.

There is a 32bit counter that counts milliseconds and it can be read from IO address 20 the low word and 21 the high word.

## System function Calls - Software Interrupts

Over forty system software interrupt based function calls have been implemented to support the various system functions.

To choose a specific function the function number must be loaded to register A0, various parameter values are passed through other registers and then start the interrupt with command INT 4 or INT 5.

Interrupts retain the values of registers except register A0 and any register that is used for result values.

Let’s start from the display related functions. These functions use the screen colors and other values stored in system variables:

BCOL EQU $485C ; Current background color for mode 1

FCOL EQU $485D ; Current foreground color for mode 1

VMODE EQU $4846 ; Current Video Mode

SCOL EQU $4847 ; Current screen color for mode 0

**INT4, A0=3 Clear screen**

**------------------------------------**

MOVI A0, 3

INT 4

The above commands clear the screen

**INT4 A0=4 Put Char**

This will display an ascii character from byte in A1 to x,y position on screen where x is the high byte and y is the low byte of the word stored in A2

**INT4 A0=5 Print zero or CR terminated string**

Prints a string that A1 points to, at position defined in A2 as above

**INT4 A0=6 Scroll**

Scrolls the screen 1 char up (8 points)

**INT4 A0=2 Plot**

Plot a point using the current colors at X=A1, Y=A2 mode in A4=1 set A4=0 clear

**INT5 A0=14 Line**

Plot a line using the current colors from x1=a1, y1=a2 to x2=a3, y2=a4

**INT 5 A0=17 Circle**

Print a circle from x=A1, y=A2 with radius = A3

**INT 5 A0=15 Horizontal Scroll Cyclical**

**INT 5 A0=19 Horizontal Scroll with graphics data filled from area pointed by A7**

Have to fill the structure below with the correct values before this call

; General Horizontal Scroll INT5 A0=15 for MODE 1

; Block at IO address 64780

; +0 line

; +2 length lines

; +4 pixel

; +6 length pixels

; +8 pixels to scroll

; Buffer at 96790 to 98303(1514 b) up to 7 pixels scroll full screen

**INT 4 A0=15 Vertical Scroll Cyclical**

**INT 5 A0=18 Vertical Scroll with graphics data filled from area pointed by A7**

Have to fill the structure below with the correct values before this call

; General Vertical Scroll INT4 A0=15 MODE 1

; Block starts at IO address 64770

; +0 line

; +2 length lines

; +4 point

; +6 length pixels

; +8 lines to scroll

; Buffer at 96790 to 98303(1514 b) up to 4 lines scroll full screen

Input Output related functions

**INT4 A0=7 Read Keyboard**

If available then bit2 of A0=1 and result keycode stored at A1

**INT 5 A0=20 Read Last Key**

Empty buffer return last key pressed in A1 or 0 if none pressed

**INT 4 A0=10 Convert Keycode to ASCII**

Convert keycode in A1 to ASCII code result in A1

**INT4 A0=7 Read from Serial Port 0**

If available then bit0 of A0=1 and result byte stored at A1

**INT4 A0=1 Write byte to Serial Port 0**

Write byte stored at A1 to serial port 0

Arithmetic operation related functions

**INT4 A0=8 32bit Integer Multiplication**

**(Replaced by CPU instruction MULU A1, A2)**

Multiply A1 by A2 result in A2A1

**INT4 A0=9 32bit Integer Division**

Divide A2 by A1 result in A1 remainder in A0

**INT5 A0=6 Unsigned 32bit Integer Division**

Divide A2 by A1 result in A1 remainder in A0

**INT5 A0=9 64bit Floating Point Multiplication**

Multiply A1A2\*A3A4 result in A1A2

**INT5 A0=10** **64bit Floating Point Division**

Divide A1A2 by A3A4 result in A1A2

**INT5 A0=11**  **64bit Floating Point Addition**

Add A1A2 + A3A4 result A1A2

**INT5 A0=11**  **64bit Floating Point Comparison**

Compare A1A2 to A3A4 result in A0 (1,0,-1)

File system related function calls

**INT5 A0=11 Initialize SDCard through SPI**

A0=1 success, 0 failure

**INT5 A0=12 Send/Receive byte to SPI**

A1 has byte to send, A0 received byte, A2=mode (1=CS low 3=CS high)

**INT5 A0=13 Read Block from SD Card**

Block number in A1, pointer to buffer (512bytes + CRC) to read to in A2

**INT5 A0=14 Write Block to SD Card**

Block number in A1, pointer to buffer to write from in A2

**INT5 A0=3 Mount Volume**

Mount first volume from SD Card, return A0=1st cluster of FAT root

**INT5 A0=2 Load File in Memory**

Pointer to filename in A4, pointer to memory location in A3

Filename has to be 8+3 length without the “.”

For example “HELLO BAS”

**INT5 A0=5 Save memory to File**

Pointer to filename in A4, pointer to memory location in A6, length in A7

**INT5 A0=4 Delete File**

Pointer to filename pointed to by A4

**INT4 A0=16 Open File**

A1 – type, A4 pointer to File name, A0=ID if success or 0 on failure

A1 = ASCII code of characters ‘r’,’ w’, ’a’

**INT4 A0=17 Close File**

A1 = file ID

**INT4 A0=18 File Read**

Read from current position, A0 returns bytes read

A1 file ID, A2 number of bytes, A3 pointer to buffer

**INT4 A0=19 File Seek**

A1=File ID. Move to the position number that is contained in A2

**INT4 A0=20 File Get Position**

A1=File ID. Return the current file position in A0

**INT4 A0=21 Get File Size**

A1=File ID. Return the file size in bytes in A0

**INT4 A0=22 Write byte in File**

A1=File ID. Write the byte in A2 to file and advance the file position

Memory related functions

FMEMORG EQU $4864 ; Free memory origin above TB

MEMTOP EQU $4858 ; TOP OF Total memory

**INT5 A0=7 Allocate Memory**

Allocate memory of A1 bytes and return pointer to memory block in A0

**INT4 A0=16**

Free A1 bytes of memory from memory block pointed by A2

## Tiny Basic

Tiny Basic for Lion is compatible with Palo Alto Tiny Basic but has a lot of new commands and fixed point arithmetic operation support added.

[Original Palo Alto Language pdf](http://www.jk-quantized.com/experiments/8080Emulator/TinyBASIC-2.0.pdf)

Fixed point arithmetic allows decimals, because in Lion TB it uses only 64bits it has limited precision.

LTB also has the following math functions: COS, SIN, INT, ROUND, SQRT, PI ex. Y=COS(X) x from -pi to pi

Lion TB has doubled the possible variables because it distinguishes between small and capital letters so a..z and A..Z variables are available. Commands are recognized in both low case and capitals.

**Loading and saving**

Lion TB supports FAT16 file system. Loads a basic file using LOAD command ex. LOAD "TEST2" the extension .BAS can be omitted

Saves a basic file using SAVE, ex. SAVE "TEST2"

With DIR command it displays the current directory

CD command changes directory examples CD VECTOR CD ..

Delete a file using DELETE command, example DELETE "TEST2.BAS" extension must be included

Load a binary file using the LCODE filename, address command, example: LCODE "MANDEL",20000 loads file MANDEL.BIN starting at address 20000 Then you can execute the code you loaded at the address 20000 using command RCODE 20000

A special variable named BTOP contains the first available address after the space currently used by basic, so a usual way to execute relocatable binary code (as produced by Lion compilers) and then return to basic would be LCODE "PROG1",BTOP and then RCODE BTOP

A binary file can be loaded from the serial port using the command GCODE address, length

**Screen and graphics**

 CLS command clears the screen

CLSP command disables all sprites and hides them from the screen

POS Y,X sets the printing position, X=0..79 Y=0..29 for mode 0 and X=0..52 Y=0..24 for mode 1

MODE command selects 0 or 1 graphics modes

Mode 0, text optimized, resolution 640x240, 80x30 chars, 16 colors restricted to 2 per char

Mode 1, graphics optimized, resolution 320x200, 53x25 chars, 16 colors no restriction

 In mode 0 command SCREEN N sets both foreground and background color with the formula N=16\*foreground + background, background and foreground are numbers 0..15 that correspond to colors 0 to 15

Also in mode 0 the command COLOR X,Y,N sets the color of individual character at X,Y to N

In mode 1 color is changed with 2 commands BCOLOR and FCOLOR

BCOLOR N where N=0 to 15, sets background color and FCOLOR N where N=0 to 15, sets foreground color

In both modes graphics are drawn using the commands PMODE, PLOT, LINE, and CIRCLE

PMODE n where n=0 or 1 sets the way the other graphic commands draw, when 0 they use foreground color, when 1 they use the background

PLOT X,Y plots a point at X,Y

LINE X1,Y1,X2,Y2 draws a line from x1,y1 to x2,y2

CIRCLE X1,Y1,R draws a circle with center x1,y1 and radius R

**Sound**

Lion has 3 sound channels + noise

BEEP F,T,C makes a sound of frequency F, duration T= 0 to 7, Channel C=0..2

To add noise OUT 11,1 to remove noise OUT 11,0

To change volume OUT 25+C,V where C=channel 0..3 and V=0..255

**Sprites**

Sprite TB commands are also to be implemented and currently sprites can be controlled with the OUT OUTB and INP INPB commands

**Input**

Function WAITK waits for a keypress and returns the ascii code of the key pressed

Function KEY returns the ascii code of a pressed key but doesn't wait for a key press

Functions JOY1 and JOY2 return the joystick status (binary: 0 0 0 Right Left Up Down Button)

10 J=JOY1

20 IF J=16 PRINT "RIGHT"

30 IF J=1 PRINT "FIRE BUTTON PRESSED"

40 K=WAITK

 50 PRINT K

**Misc.**

Function TIMER returns a value in milliseconds that resets every 65536ms

 Command SLIST lists basic program to serial port

### TB as Lion’s command interpreter

Tiny basic currently plays the role of Lion’s command interpreter.

By using the character ! as a prefix, binary files are loaded and the parameter line passes to them and are executed as in the following examples:

!play song.raw

The above line executes the program play.xct that reads the file song.raw and plays the sampled music it contains.

!view read.me displays the contents of file read.me

# Appendices

## System I/O ports

#0, OUT Serial port 0 byte to send

#1, OUT Serial port 1 byte to send

#2, IN Serial send/receive command port 0

#3, IN Serial send/receive command port 1

#4, IN Serial byte input port 0

#5, IN Serial byte input port 1

#6, IN Serial status and Serial Keyboard status

#8, OUT Sound channel 1 divider/duration/start word

#9, IN audio channel status

#10, OUT Sound channel 2 divider/duration/start word

#11, OUT Audio noise channel enable for each channel bits 0-2, 1=enable 0=disable

#12, OUT Sound channel 3 divider/duration/start word

#13, OUT Enable interrupt #2 at horizontal sync signal

#14, IN PS/2 Keyboard data in

#15, OUT PS/2 Keyboard read command

#16, IN SPI data read

#17, IN SPI status

#18, OUT SPI data to send

#19, OUT SPI control

#20, OUT sprite buffer change

#20, IN read 16-bit timer lower word 1 KHz

#21, IN read 16-bit timer higher word 1 KHz

#22, IN Joystick status

#23, IN Vsync in bit1, Hsync in bit0

#24, OUT Video mode

#25, OUT channel 1 volume 8bit

#26, OUT channel 2 volume 8bit

#27, OUT channel 3 volume 8bit

#28, OUT noise volume 8bit

#30, OUT vector display mode, Audio/XY, Stereo

#31-#33, OUT Second frequency for channel 1 to 3, bits 0 to 3

#34 OUT Set sample playback rate, default =3600, 8192bps

#35, IN Current horizontal line number

#41, #42 OUT Set baud rate divider of 25MMhz clock, 1301 = 38400 for Port 0, 1

#53 OUT Set spi speed

## System variables

ISTACK EQU $4000

SDCBUF1 EQU $4200 ;DS 514 Buffer 1

SDCBUF2 EQU $4402 ;DS 514 Buffer 2

SDCBUF3 EQU $4604

FATBOOT EQU $4806 ; Fat boot #sector

CURDIR EQU $4808 ; Root directory #sector

FSTCLST EQU $480A ; First data #sector

FSTFAT EQU $480C ; First Fat first #sector

SDFLAG EQU $480E ; SD card initialized by rom=256

COUNTER EQU $4810 ; General use counter increased by int 3

CURSOR EQU $4814 ; CURSOR XY

RHINT0 EQU $4816 ; Hardware interrupt 0

RHINT1 EQU $481C ; Hardware interrupt 1

RHINT2 EQU $4822 ; Hardware interrupt 1

RINT6 EQU $4828

RINT7 EQU $482E

RINT8 EQU $4834

RINT9 EQU $483A

RINT15 EQU $4840

VMODE EQU $4846 ; Current video mode

SCOL EQU $4847 ; Screen color mode 0

SHIFT EQU $4848

CAPSL EQU $4849 ; Caps lock flag

CIRCX EQU $484A

CIRCY EQU $484C

PLOTM EQU $484E ; Plot mode

SECNUM EQU $4850

SECPFAT EQU $4852

FATROOT EQU $4854

SDHC EQU $4856 ; is the sd hc ?

SDERROR EQU $4857 ; sd card #error

MEMTOP EQU $4858 ; TOP OF Total memory

BCOL EQU $485C ; Background color mode 1

FCOL EQU $485D ; Foreground color mode 1

HINT EQU $485E

FMEMORG EQU $4864 ; Free memory origin above TB

RESERVED EQU $486A

XX EQU $4868

YY EQU $4869

COLTABLE EQU 61152

## Music notes to clock divider table

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Freq** | **Note** |
| 3635 | 55,01 | A |
| 3431 | 58,28 | A# |
| 3239 | 61,73 | B |
| 3056 | 65,42 | C |
| 2884 | 69,32 | C# |
| 2724 | 73,39 | D |
| 2570 | 77,79 | D# |
| 2426 | 82,41 | E |
| 2290 | 87,30 | F |
| 2161 | 92,51 | F# |
| 2040 | 97,99 | G |
| 1925 | 103,84 | G# |
| 1817 | 110,01 | A |
| 1715 | 116,55 | A# |
| 1619 | 123,46 | B |
| 1528 | 130,80 | C |
| 1442 | 138,60 | C# |
| 1362 | 146,74 | D |
| 1285 | 155,52 | D# |
| 1213 | 164,74 | E |
| 1145 | 174,52 | F |
| 1080 | 185,01 | F# |
| 1020 | 195,89 | G |
| 962 | 207,68 | G# |
| 908 | 220,02 | A |
| 857 | 233,10 | A# |
| 809 | 246,91 | B |
| 764 | 261,44 | C |
| 721 | 277,01 | C# |
| 681 | 293,26 | D |
| 642 | 311,04 | D# |
| 606 | 329,49 | E |
| 572 | 349,04 | F |
| 540 | 369,69 | F# |
| 510 | 391,39 | G |
| 481 | 414,94 | G# |
| 454 | 439,56 | A |
| Parameter | Freq | Note |
| 428 | 466,20 | A# |
| 404 | 493,83 | B |
| 382 | 522,19 | C |
| 360 | 554,02 | C# |
| 340 | 586,51 | D |
| 321 | 621,12 | D# |
| 303 | 657,89 | E |
| 286 | 696,86 | F |
| 270 | 738,01 | F# |
| 255 | 781,25 | G |
| 240 | 829,88 | G# |
| 227 | 877,19 | A |
| 214 | 930,23 | A# |
| 202 | 985,22 | B |
| 191 | 1041,67 | C |
| 180 | 1104,97 | C# |
| 170 | 1169,59 | D |
| 160 | 1242,24 | D# |
| 151 | 1315,79 | E |
| 143 | 1388,89 | F |
| 135 | 1470,59 | F# |
| 127 | 1562,50 | G |
| 120 | 1652,89 | G# |
| 113 | 1754,39 | A |
| 107 | 1851,85 | A# |
| 101 | 1960,78 | B |
| 95 | 2083,33 | C |
| 90 | 2197,80 | C# |
| 85 | 2325,58 | D |
| 80 | 2469,14 | D# |
| 75 | 2631,58 | E |
| 71 | 2777,78 | F |
| 67 | 2941,18 | F# |
| 63 | 3125,00 | G |
| 60 | 3278,69 | G# |
| 56 | 3508,77 | A |