memory map - sidbox machine - 4.3.5667

Zero Page	$0000
255 bytes	store $ff into $0000 to terminate the program
	$00FF

CHARACTER MAP	$0400
300 bytes	300 bytes allows up to 20 x 15 letters on screen, from top left, to bottom right
	$052B

SCREEN MEMORY	$8000
2400 bytes	2400 bytes - 160 x 120 pixel areas
	$895F

COLOUR PALETTE		$A000
Change colour of pallete and colour cycle speed	00 1FF 200	colour palette index $00-$1FF, each colour is 16bit wide, RGB565. Change the speed of the cycle of colours. The colours in palette $80-$87 are cycled
520 bytes		$A207

SPRITES		$A300
A bank of sprite pointers and registers, each sprite can be upto 32x32 pixels of 2 or 4 colours, set bit 3 to set colour modes, show and hide sprites using bit 0. Colour 0 is transparent in both modes. single colour mode, the colour pointer is used only to store the colour id and should be stored in high byte, no colour lookup is done here. multicolour mode, the colour data is stored else where so you need to point to the colour data by storing the pointer here instead.		W=width, H=Height, MC=multicolour, cd=collision_en or=order, en=enable
	00	#0 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	01	#1 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	02	#2 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	03	#3 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	04	#4 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	05	#5 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	06	#6 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	07	#7 = W[7..6] H[5..4] mc[3] cd[2] or[1] en[0]
	08	16bit sprite #0 colour pointer / colour id
	0A	16bit sprite #1 colour pointer / colour id
	0C	16bit sprite #2 colour pointer / colour id
	0E	16bit sprite #3 colour pointer / colour id
	10	16bit sprite #4 colour pointer / colour id
	12	16bit sprite #5 colour pointer / colour id
	14	16bit sprite #6 colour pointer / colour id
	16	16bit sprite #7 colour pointer / colour id
POINTER TO SPRITES: Setting these memory locations help point to the sprite data.		NOTE: HiByte, LoByte
	20	16bit - pointer to sprite #0
	22	16bit - pointer to sprite #1
	24	16bit - pointer to sprite #2
	26	16bit - pointer to sprite #3
	28	16bit - pointer to sprite #4
	2A	16bit - pointer to sprite #5
	2C	16bit - pointer to sprite #6
	2E	16bit - pointer to sprite #7
SPRITE POSITIONING: Each sprite has its own positioning 16bit Registers. NOTE: sprites are rendered #0 first, #7 last	30	Sprite #0 - X: [7..0], Y: [7..0]
	32	Sprite #1 - X: [7..0], Y: [7..0]
	34	Sprite #2 - X: [7..0], Y: [7..0]
	36	Sprite #3 - X: [7..0], Y: [7..0]
	38	Sprite #4 - X: [7..0], Y: [7..0]
	3A	Sprite #5 - X: [7..0], Y: [7..0]
	3C	Sprite #6 - X: [7..0], Y: [7..0]
	3E	Sprite #7 - X: [7..0], Y: [7..0]
SYSCON: Persist will only work if not in TEXT mode	40	Sprite Collition Bits [7..0]
	41	Background Collition Bits [7..0]
	42	Sprite SYS CON: persist[7]
67 bytes		$A342

Screen Control	bit	$D011
	0	
	1	
	2	
	3	
	4	0 - screen auto refresh off, 1 - auto on
	5	0 - Text mode, 1 - Bitmap mode
	6	1 = Extended Background mode (multicolour)
	7	r/w - 8th bit of the raster line interrupt
1 bytes		$D011

Raster line	$D012
	Read / Write Raster line interrupt
1 bytes	$D012

Interrupt Status Register	bit	$D019
	0	w/r: 1 = Ack raster line Interrupt
	1	w/r: 1 = Ack Vblank (sidbox)
	2	w/r: 1 = Ack Hardware button
	3	w/r: 1 = Ack Touch Screen
	4	w/r: 1 = Ack Timer 1 overflow
	5	w/r: 1 = Ack Screen update Interrupt
	6	w/r: 1 = Ack Sprite Collision
D019	7	w/r: 1 = Ack Timer 2 Overflow
1 bytes		$D019

Interrupt Control Register	bit	$D01A
	0	w/r: 1 = Raster interrupt Enable
	1	w/r: 1 = Vblank (Sidbox)
	2	w/r: 1 = Hardware buttons En
	3	w/r: 1 = Touch Screen En
	4	w/r: 1 = Timer 1 overflow En
	5	w/r: 1 = Screen update En (25fps)
	6	w/r: 1 = Sprite Collision En
D01A	7	w/r: 1 = Timer 2 overflow En
1 bytes		$D01A

NMI Status Register	bit	$D01B
	0	w/r: 1 = Ack raster line Interrupt
	1	w/r: 1 = Ack Vblank (sidbox)
	2	
	3	
	4	w/r: 1 = Ack Timer 1 overflow
	5	w/r: 1 = Ack Screen update Interrupt
	6	
D01B	7	
1 bytes		$D01B

NMI Control Register	bit	$D01C
	0	w/r: 1 = Raster interrupt Enable
	1	w/r: 1 = Vblank (Sidbox)
	2	
	3	
	4	w/r: 1 = Timer 1 overflow En
	5	w/r: 1 = Screen update En (25fps)
	6	
D01C	7	
1 bytes		$D01C

SID CHIP		$D400
We all know what this is, no description needed ;)		**Voice 1**
	00	Freq Lo
	01	Freq Hi
	02	PWM - 7..0
	03	PWM - 11..8
	04	NSE|SQ|SW|TR|TST|RNG|SYN|GTE
	05	ATK [3..0] | DCY [3..0]
	06	SUST[3..0] | REL [3..0]
		Voice 2
	07	Freq Lo
	08	Freq Hi
	09	PWM - 7..0
	0A	PWM - 11..8
	0B	NSE|SQ|SW|TR|TST|RNG|SYN|GTE
	0C	ATK [3..0] | DCY [3..0]
	0D	SUST[3..0] | REL [3..0]
		Voice 3
	0E	Freq Lo
	0F	Freq Hi
	10	PWM - 7..0
	11	PWM - 11..8
	12	NSE|SQ|SW|TR|TST|RNG|SYN|GTE
	13	ATK [3..0] | DCY [3..0]
	14	SUST[3..0] | REL [3..0]
		Filter/Volumes
	15	LC LOW - Filter [2..0]
	16	FC HIGH - Filter [10..3]
	17	RES [7..4] FILT EX [3] FILT 3,2,1 [2..0]
	18	3 OFF [7] HP[6] BP[5] LP[4] VOL[3..0]
		Misc
	19	not used
	1A	not used
	1B	not used
	1C	not used
29 bytes		**$D41C**

CELL BACK COLOUR	$D800
300 bytes	300 bytes a value in this sets the background colour of the text, access to 256 colours $D92B

FOREGROUND CELL COLOUR #0	$DA00
Primary colour for single colour mode and multicolour mode 300 bytes	300 bytes a value in this sets the background colour of the text, access to 256 colours $DB2B

FOREGROUND CELL COLOUR #1	$DB30
Colour data used only in multicolour mode 300 bytes	300 bytes a value in this sets the background colour of the text, access to 256 colours $DC5B

TIMERS		$DD04
a value of $ffff is 66ms	+00	low byte Timer 1
	+01	high byte Timer 1
	+02	low byte Timer 2
	+03	high byte Timer 2
4 bytes		$DD07

Timer STATUS	BIT	$DD0D
		Read:
	0	Timer A over flow status set
	1	Timer B over flow status set
	2	<not implemented yet>
	3	<not implemented yet>
	4	<not implemented yet>
	7	<not implemented yet>
		Write:
	0	Timer A overflow status clear
	1	Timer B overflow status clear
	2	<not implemented yet>
	3	<not implemented yet>
	4	<not implemented yet>
	7	<not implemented yet>
1 bytes		$DD0D

SERIAL PORT RS232		$DE00
A basic preset RS232 port, set to 115200baud - The Read and Write pointers refer to a string of data terminated with a $00. If you want to send data of any value, better to use the send single byte. Simply store the byte into the address and that's it. Would recommend a pause between bytes depending on the receiver	00	High Byte Send Pointer
	01	Low Byte Pointer
	02	reserved
	03	reserved
	04	Single Byte to send
	05	Control Bits 0x01 - Send string from pointer, terminated with 0x00
NOTE: The RS232 processor has a 64byte buffer	10	64 bytes of receive buffer
70 bytes		$DE45

CHAR FONTS	$DF00
00 2048 bytes	256 chars each char has 8 bytes which can be anything you like. $E6FF

FOREGROUND CELL COLOUR #2	$E710
Colour data used only in multicolour mode 300 bytes	300 bytes a value in this sets the background colour of the text, access to 256 colours $E83B

Hardware Control	$F000
	Putting values in this memory causes things to happen on the computer **FLAGS:** 0x01 - clear screen 0x02 - flip screen now 0x04 - update sprites 0x08 - redraw charmap 0x10 - redraw sprites (this doesn't cls)

CPU CONTROL	$F002
+00	delay in ms

SDCARD IO		$F100
DISK IO, Open, Save, Close, databuffers, file locations.	00	OPEN function
	01	SAVE function
	02	CLOSE function
	03	filename$ [8.3]
	10	directory name[16]
	20	hibyte output ptr
	21	lobyte output ptr
	22	hibyte input ptr
	23	lobyte input ptr
	24	send bytes - setting this will cause the function to save the bytes stored in the output buffer.
	25	get bytes - setting this will cause the function to load the bytes size into the set input pointer, bytes up to 255 bytes perload
	26	**status bits** 01 - File open OK 02 - Save file OK 04 - Close OK 08 - Send OK - Must be cleared after sending

Random Generator		$FE00
	00	8 bit random number
	02	rnd seed - low byte
	03	rnd seed - hi byte #0 - from timer 1 #1 - #65535

IO		$FE10
SIDbox has 4 physical buttons and 1 hidden Moddable button Input + touch irq and + touch(x,y)	+$0	Hardware buttons [4..0]
	+$1	touch x : 0-160 (screen buffer width)
	+$2	touch y : 0-120 (screen buffer height)

INTERRUPT VECTOR ADDRSSES

Interrupts allow your program to jump to another subroutine using the address set in the IRQ/NMI vectors. In order to stop your program interrupting again while still processing the previous interupt the FLAGS must be cleared. Another Interrupt Request wont be trigged until you clear the IRQ/NMI flags

The interrupt bits

These are used so you can get the bits of what caused the interrupt. Get these values from the NMI or IRQ interrupt bits. Don't forget to clear the Interupt bits (Flags) otherwise each new IRQ/NMI you wont know what was actually causing the Interrupt

NMI (INTERRUPT ENABLE BITS) - $FFF6

IRQ (INTERRUPT ENABLE BITS) - $FFF7

NMI FLAGS - $FFF8, IRQ FLAGS - $FFF9

Bits set to 1 by the IRQ and NMI requests. Recommend clearing the bits after use.

NMI VECTOR - $FFFA - $FFFB

Two bytes, $fffa - low byte, $fffb - high byte - where your subroutine is located

COLD BOOT VECTOR - $FFFC - $FFFD

this CPU will look at the address here and move to the location pointed here and begin executeing… - HOWEVER THIS IS NOT REQUIRED FOR THE SIDBOX

IRQ VECTOR - $FFFE - $FFFF

Two bytes, $fffe - low byte, $ffff - high byte, set the vector with the address of your subroutine.

$FFFF