memory map - sidbox machine

Zero Page	$0000 $00ff

SCREEN MEMORY	**$8000**
2400 bytes	300 bytes allows up to 20 x 15 letters on screen, from top left, to bottom right
	$895F

CHARACTER MAP	**$8A00**
300 bytes	300 bytes allows up to 20 x 15 letters on screen, from top left, to bottom right
	$8B2B

CELL BACK COLOUR	**$8C00**
300 bytes	300 bytes a value in this sets the background colour of the text, access to 256 colours
	$8D2B

FOREGROUND CELL COLOUR	**$8E00**
300 bytes	300 bytes a value in this sets the background colour of the text, access to 256 colours
	$8F2B

COLOUR PALETTE		**$A000**
Change colour of pallete and colour cycle speed	**00** **1FF**	colour palette index $00-$1FF, each colour is 16bit wide, RGB565.
	200	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.		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]
Colours are handled by offsets, EG: a sprite with a colour 66, in multimode will use the 3 colours. The other 3 colours are picked from the pallete 66 + the colour selected in the multimode. Transparent colour is 0 and will be 0. In single colour mode it will either be 0-transparent or 1-colour 66. There is a palette of 256 colours available.	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	sprite #0 colour
	09	sprite #1 colour
	0A	sprite #2 colour
	0B	sprite #3 colour
	0C	sprite #4 colour
	0D	sprite #5 colour
	0E	sprite #6 colour
	0F	sprite #7 colour
POINTER TO SPRITES:		NOTE: HiByte, LoByte
Setting these memory locations help point to the sprite data.	10	16bit - pointer to spite #0
	12	16bit - pointer to spite #1
	14	16bit - pointer to spite #2
	16	16bit - pointer to spite #3
	18	16bit - pointer to spite #4
	1A	16bit - pointer to spite #5
	1C	16bit - pointer to spite #6
	1E	16bit - pointer to spite #7
SPRITE POSITIONING:	20	Sprite #0 - X: [7..0], Y: [7..0]
Each sprite has its own positioning 16bit Registers. NOTE: sprites are rendered #0 first, #7 last	22	Sprite #1 - X: [7..0], Y: [7..0]
	24	Sprite #2 - X: [7..0], Y: [7..0]
	26	Sprite #3 - X: [7..0], Y: [7..0]
	28	Sprite #4 - X: [7..0], Y: [7..0]
	2A	Sprite #5 - X: [7..0], Y: [7..0]
	2C	Sprite #6 - X: [7..0], Y: [7..0]
	2E	Sprite #7 - X: [7..0], Y: [7..0]
	30	Sprite Collition Bits [7..0]
	31	Background Collition Bits [7..0]
64 bytes		$A33F

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

CHAR FONTS		$D500
	00	256 chars each char has 8 bytes which can be anything you like.
2048 bytes		$DCFF

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 Send 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

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)

Hardware Settings	$F001
	Bitwise settings controls how things work **FLAGS:** 0x01 - screen mode 0x02 - set multicolour map 0x04 - super frame rate 24fps to 50fps

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

Timers		**$FD00**
The timers available are 16bits. Read or write timer values here. Using the interrupts with timers can be set and the Interrupts will be triggered on timer overflows. For simplicity and due to the lack of power for everything, this timer is set to 1 kHz, should be enough to run timed interrupts The 1 kHz is a true timer it's independent of the CPU (6502)	**00**	Timer 1 hibyte
	01	Timer 1 lobyte
	02	Timer 2 hibyte
	03	Timer 2 lobyte
	04	Timer 1 Period overflow Hibyte
	05	Timer 1 Period overflow Lobyte
	06	Timer 2 Period overflow Hibyte
	07	Timer 2 Period overflow Lobyte

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 + touch irq and + touch(x,y)	**0**	Hardware buttons [3..0]
	1	touch x : 0 - 160 (screen buffer width)
	2	touch y : 0 - 120 (screen buffer height)
	F	put anything in here to exit emulation

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

NMI - INTERUPT ENABLE BITS

0x01 - Enable timer 1

0x02 - Enable timer 2

0x04 - Enable RS232 RX Buffer used

0x08 - Enable Sprite Collision

IRQ VECTOR - $FFFE - $FFFF

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

IRQ - INTERUPT ENABLE BITS

0x01 - Enable screen blank time 24hz / 50hz

0x02 - Enable hardware button down

0x04 - Enable Touch screen

$FFFF