JBit QS

Getting Started

If you are new to 6502 programming, read the online tutorial and then run some programs step by step. From the JBit menu, select Demos and then 6502. Study all the programs (they are very short) in the order they are presented. First select the option Info to get some hints about the program, then select the option Load&Debug to start the Monitor.

Monitor Status Bar: At the right there is the indication of the current view (CPU or MEM). In CPU view, at the left there is the PC of the next instruction. In MEM view, at the left there is the current cell address.

Monitor Keys: `2`, `4`, `6` and `8` are the cursor keys. `1` performs 1 step. `3` performs n steps (you can select n by using the option StepSize; default is 10). `7` goes to the next line. `9` steps out of the current subroutine. `0` switches between CPU and MEM view. `*` continues the program (you can then pause it by pressing a soft key). `#` shows the video (you can then go back to the monitor by pressing any key). In MEM view, `5` changes the value of a cell. The option Edit can be used to change the registers of the CPU.

Editor Status Bar: At the right there is the indication of the current major (NAV and EDT) and minor modes. At the left there is the letter C if you are on a code page or D if you are on a data page followed by the current cell address.

Editor NAV Mode Keys: `2`, `4`, `6` and `8` are the cursor keys. `5` switches to EDT mode. `1` goes to the previous snap point, `3` to the next one. `7` sets the mark and `0` swaps the cursor and the mark. `9` goes to the address of the operand. `*` runs the program. `#` switches between ASM and MEM minor modes.

Editor EDT Mode Keys: `*` moves to the next cell. `#` switches between ASM and MEM minor modes. In MEM minor mode, `0-9` are used to enter the decimal value of the cell. In ASM minor mode, `2-9` are used to enter the letters of the mnemonic.

Editor Example: To enter `LDA #12` press `5` to enter EDT mode, then press `#` to enter ASM minor mode, then press the sequence `5` JKL, `3` DEF and `2` ABC for L-D-A, then select the instruction `LDA #n` from the list, then press `1` and `2` to enter the operand and finally select OK to confirm the instruction.

On some phones, use `@` for `#`.

System Overview

Neither interrupts nor BCD mode are supported. SED, RTI or any invalid opcode cause the program to abort. To end the program use the BRK (0) opcode.

By convention a program is a series of code pages containing a single stream of valid instructions followed by a series of data pages. Programs structured in this way are said to be *well-formed*. JBit neither needs nor enforces this convention, but some editing operations (e.g. Resize) are only available on well-formed programs.

The hexadecimal notation is cumbersome on mobile phones; it is therefore common in JBit to use the decimal notation. Absolute addresses are presented in the non-standard *page:offset* notation.

The program is loaded and starts at the beginning of page 3. Page 255 is reserved for future use. The IO chip is mapped to page 2.

IO Basic Operations

IO registers are stated as decimal offsets relative to page 2.

The console video memory is a 10x4 matrix of extended Latin1 characters starting from `CONVIDEO` and disposed in row-major order.

```
CONVIDEO 40  (40 bytes)
```

The frame refresh is controlled by two registers:

```
FRMFPS   17      FRMDRAW  18
```

`FRMFPS` is the number of frames per second multiplied by 4 (e.g. 40, the initial value, is 10 fps). Writing into `FRMDRAW` causes the CPU to be suspended until the current frame has been drawn.

Random numbers $\leq n$ are read from `RANDOM`. Writing 0 to `RANDOM` swaps the current sequence generator: time-based (default) or deterministic. Any other value sets n (default is 255).

```
RANDOM   23
```

The standard KeyPresses (the ones that can be represented by ASCII codes; usually only `0-9`, `#` and `*`) are enqueued starting from `KEYBUF`; the rest of the buffer is filled with 0s. Write into `KEYBUF` to consume a KeyPress. If the buffer is full when a new key is pressed, that KeyPress is lost.

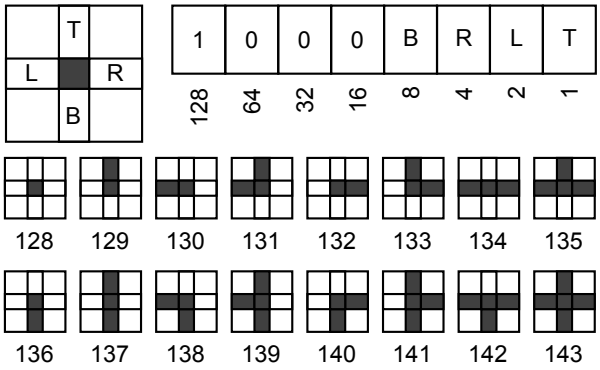
```
KEYBUF   24  (8 bytes)
```


Latin1

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			

160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Line Art



Conversions

0	0000	0	0	8	1000	8	128
1	0001	1	16	9	1001	9	144
2	0010	2	32	10	1010	A	160
3	0011	3	48	11	1011	B	176
4	0100	4	64	12	1100	C	192
5	0101	5	80	13	1101	D	208
6	0110	6	96	14	1110	E	224
7	0111	7	112	15	1111	F	240

6502

Operations

BRK: BReaK; in JBit, used to halt the VM.
NOP: No OPeration.
LDA, *LDX*, *LDY*: LoaD Accumulator/X/Y.
STA, *STX*, *STY*: STore Accumulator/X/Y.
INX, *INY*, *INC*: INCrement X/Y/memory.
DEX, *DEY*, *DEC*: DECrement X/Y/memory.
TAX, *TAY*, *TXA*, *TYA*, *TSX*, *TXS*: Transfer register.
CMP, *CPX*, *CPY*: CoMPare with accumulator/X/Y.
JMP: JuMP.
JSR: Jump to SubRoutine.
RTS: ReTurn from Subroutine.
CLC, *CLV*: CLear Carry/oVerflow.
SEC: SEt Carry.
BEQ, *BNE*: Branch if EQual/Not Equal (z flag).
BMI, *BPL*: Branch if MInus/PLus (n flag).
BCC, *BCS*: Branch if Carry Clear/Set.
BVC, *BVS*: Branch if oVerflow Clear/Set.
ADC: ADd (to/into accumulator) using Carry.
SBC: SuBtract (from/into accumulator) using Carry.
AND: bitwise AND (with accumulator).
ORA: bitwise inclusive OR (with Accumulator).
EOR: bitwise Exclusive OR (with accumulator).
BIT: test BITs: #6 to v flag and #7 to n flag.
ASL, *LSR*: Arithmetic/Logical Shift Left/Right.
ROL, *ROR*: ROtate Left/Right.
PHA, *PLA*: PusH/PuLl Accumulator.
PHP, *PLP*: PusH/PuLl Processor status.

Operands

#n: Constant n.
n: Cell 0:n.
n:n: Cell n:n.
n:n,X: Cell X cells after n:n.
n:n,Y: Cell Y cells after n:n.
r: Next cell + relative offset r, shown as n:n.
n,X: Cell 0:(n+X modulo 256).
n,Y: Cell 0:(n+Y modulo 256).
(n:n): Cell pointed by n:n.
(n,X): Cell pointed by 0:(n+X modulo 256).
(n),Y: Cell Y cells after cell pointed by 0:n.

Examples

TAX: Transfer (i.e. copy) Accumulator into X.
LDA #2: LoaD Accumulator with constant 2.
LDX 2: LoaD X with content of cell 0:2.
STX 2:18: STore (i.e copy) X into cell 2:18.
BCC 6: If C=0, skip (i.e. jump forward) 6 bytes.
BCC 240: If C=0, jump back 16 (i.e. 256-240) bytes.

