

## JBit E0

### /6502/ciao

Show the word "CIAO".

CPU Instructions: LDA (LoaD Accumulator), STA (STore Accumulator) and BRK (BReaK).

CPU Addressing Modes: Immediate [#n], absolute [n:n] and implied [].

IO: Video memory is at 40-79 in page 2; 1st row starts at 2:40, 2nd row starts at 2:50 and so on for a 10x4 matrix of Latin1 (extended ASCII) characters.

Size: 1 code page, 0 data pages.

C 3:0

```
000: 169 067 141 053 002 169 073 141
008: 054 002 169 065 141 055 002 169
016: 079 141 056 002 000 000 000 000
```

### /6502/loop1

Keep updating a character on the screen (infinite loop). To stop the program press a soft key.

CPU Instructions: INC (INCrement memory), NOP (No OPeration) and JMP (JuMP).

IO: Writing 0 into 2:18 suspends the CPU until the screen has been redrawn (refresh rate is 10 frames per second).

Style: Even if in JBit the registers are always 0 at the beginning, it is suggested to clear them anyway.

Size: 1 code page, 0 data pages.

C 3:0

```
000: 169 000 238 040 002 141 018 002
008: 234 076 002 003 000 000 000 000
```

### /6502/fill1

Fill the display with Xs.

CPU Instructions: LDX (LoaD X register), INX (INcrement X register), CPX (ComPare X register) and BNE (Branch on Not Equal)

CPU Addressing Modes: Absolute indexed [n:n,X] and relative [r]; relative mode is used in branches and looks like absolute mode in the monitor.

Size: 1 code page, 0 data pages.

C 3:0

```
000: 162 000 169 088 157 040 002 232
008: 224 040 208 248 000 000 000 000
```

### /6502/fill2

Same as fill1, but faster.

Note that X ranges from 40 to 1 inside the loop (in fill1 the range was 0 to 39).

CPU Instructions: DEX (DEcrement X register).

Puzzle: Not Equal in BNE really means "Not Zero" and CPX really means "subtract discarding the result".

Style: This is the usual way to write loops for the 6502.

Size: 1 code page, 0 data pages.

C 3:0

```
000: 162 040 169 088 157 039 002 202
008: 208 250 000 000 000 000 000 000
```

### /6502/loop2

Changing letters (infinite loop).

CPU Instructions: DEC (DECrement memory), LDY, STY and TAX (Transfer/copy Accumulator to X register).

IO: Write the desired FPS \* 4 into 2:17 (e.g. 4 = 1 FPS).

Size: 1 code page, 0 data pages.

C 3:0

```
000: 169 004 141 017 002 169 065 141
008: 040 002 160 000 140 018 002 169
016: 025 170 238 040 002 140 018 002
024: 202 208 247 170 206 040 002 140
032: 018 002 202 208 247 076 017 003
```

### /6502/loop3

Bouncing letters (finite loop A-F). More of the same.

Size: 1 code page, 0 data pages.

C 3:0

```
000: 160 065 162 000 169 032 157 040
008: 002 232 152 157 040 002 169 000
016: 141 018 002 224 009 208 237 169
024: 032 157 040 002 202 152 157 040
032: 002 169 000 141 018 002 224 000
040: 208 237 200 192 071 208 211 169
048: 032 157 040 002 000 000 000 000
```

**/6502/keypad**

Firing characters (press any key except \* to fire a character, use \* to stop firing).

CPU: BEQ (Branch if EQual) and CMP (CoMPare accumulator).

IO: Read from 2:24; if the value is 0 no key has been pressed; a value different than 0 is the ASCII code of the key that has been pressed (e.g. '0' is 48). Write 1 into 2:24 to acknowledge the key and remove it from 2:24. At most 8 unacknowledged keys are kept, the others are lost.

Size: 1 code page, 0 data pages.

C 3:0

```
000: 169 000 141 018 002 173 024 002
008: 240 248 201 042 240 030 162 000
016: 157 040 002 168 169 000 141 018
024: 002 169 032 157 040 002 152 232
032: 224 010 208 236 169 001 141 024
040: 002 076 000 003 000 000 000 000
```

**/6502/charset**

Browse the charset (2=Up, 8=Down and 0=Exit).

CPU: CLC (CLear Carry), SEC (SEt Carry), ADC (ADd with Carry), SBC (SuBtract with Carry), BCS (Branch on Carry Set), BCC (Branch on Carry Clear).

Style: This is "spaghetti" code; note how difficult it is to understand how it works. A few tricks have also been used to keep it in 64 bytes (BCC/BCS instead of JMP and branching to 3:1).

Size: 1 code page, 0 data pages.

C 3:0

```
000: 169 000 170 160 000 153 040 002
008: 024 105 001 200 192 040 208 245
016: 138 160 000 140 018 002 172 024
024: 002 240 248 192 050 240 015 192
032: 056 240 020 192 048 240 218 160
040: 001 140 024 002 208 212 201 000
048: 240 245 056 233 010 176 240 201
056: 220 240 236 024 105 010 144 231
```