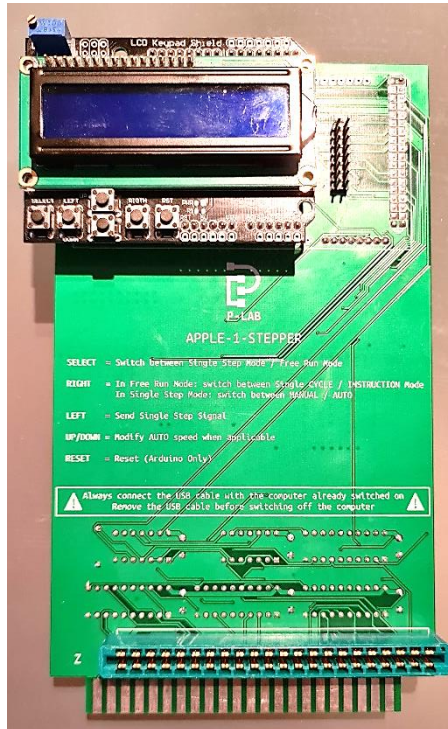


APPLE-1-STEPPER

v1.2



Have you ever had the desire to observe in real time the operation of a microprocessor while it executes a program, with the precision of the single clock cycle?

Or: to observe the instructions executed at boot?

Have you ever felt frustrated because your program just doesn't work the way you want it to, and you can't find the problem?

Even more: your computer is not working as it should. You would like to see what is going on the BUSES and control lines... but you do not have a logic analyzer?

Would you like to be able to type on the Apple-1 using an external USB terminal to load short programs?

1. DESCRIPTION

Apple-1-Stepper is a hardware device that connects to Apple-1 computers, either Originals or Replicas.

It allows you to:

- stop the execution of instructions by the microprocessor for debugging, education or troubleshooting purposes and then resume them later,
- control manually (or by an internal timer) the execution of instructions,
- display the status and the content of Data and Address BUSes the instructions being executed, the status of the R/W line, the status of IRQ and NMI signals, both on LCD display and on serial terminal.

With the optional additional keyboard connection, you will be also able to:

- load pre-installed short diagnostic programs,
- write and send programs to the Apple-1 from external devices.

2. DAMAGES FROM ELECTROSTATIC DISCHARGES

Apple-1-Stepper is sensitive to static electricity, just like your Apple computer, and may be damaged by it.

Before any operation on your devices, you must discharge the static electricity accumulated by your body and prevent it from building up again.

We do not accept any responsibility for damage, even serious or fatal, caused to people / things / intellectual property during the installation or use of this device.

3. IMPORTANT WARNINGS

To prevent any damage to your Apple-1 computer, if you plan to use the USB port of the *Apple-1-Stepper* always follow the instructions below:

ALWAYS DISCONNECT the USB cable from the computer **BEFORE** turning off the Apple-1.

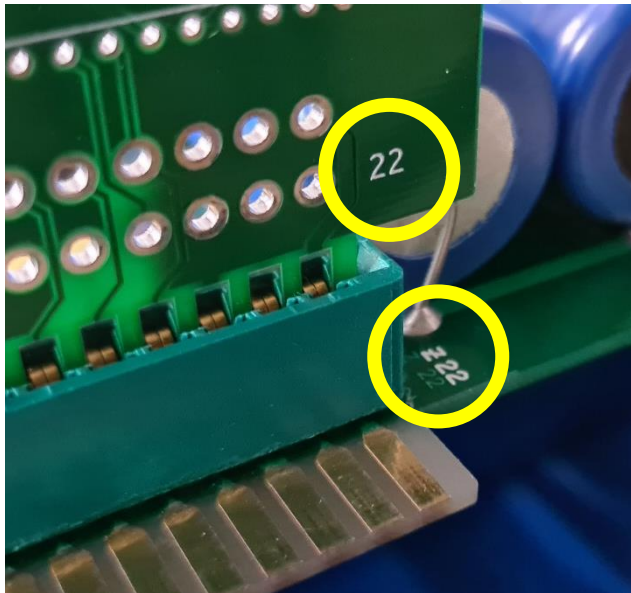
NEVER CONNECT the USB cable to the computer when the Apple-1 is turned off.

The correct power on/off sequence is the following:

- 1) **Switch on** the Apple-1,
- 2) **Connect the USB** cable to the computer.
... use the computer. When you decide to turn it off:
- 3) **Disconnect the USB** cable from the computer,
- 4) **Switch off** Apple-1.

4. INSTALLATION

The card must be inserted as shown in the picture:



The number "22" on the card and the number "22" on the motherboard must be on the same side.

The number "22" on the card must therefore face the outer side of the Apple-1 motherboard.

The indication must be followed even when the side connector is used.

Using the card in combination with PASSIVE BUS EXTENDERS could lead to malfunctions: do not use this kind of devices.

WARNING: Switching on the computer with the card oriented incorrectly **INSTANTLY DAMAGES** the computer and the card itself.

DESIGN & THEORY

The strangely shaped board, which will be described shortly, can be inserted into the female connector, traditionally reserved for the Cassette Interface, or connected to the motherboard using the side connector.

In this case it will be possible to make coexist *Apple-1-Stepper* with other boards, such as the already mentioned Cassette Interface, or Juke-Box / CFFA1 boards or anything else.

The shape of the board therefore makes it possible to accommodate additional cards in the traditional female connector, without any risk of unwanted physical/electrical contact between them.

The maximum height allowed for additional expansion boards is 100mm.

Unlike other similar projects involving clock manipulation, Apple-1 presents some major difficulties:

- The traditionally used microprocessor, NMOS 6502, does not allow to reduce the clock to zero, or the internal registers will be corrupted,
- Apple-1 also uses Dynamic RAM. The essential cycles required for its refresh are dependent on the system clock, if this is slowed down too much or even stopped, the memory contents will be corrupted.
- The timing of the video signal components also depends on the main oscillator.

It is therefore not possible to manipulate the system clock in the literal sense of the term.

From a functional point of view the board implements the circuit suggested in the Apple-1 Operating Manual, plus many automations that will be soon described.

The proposed circuit does not modify the system clock but uses the RDY line of the microprocessor to stop operations at the appropriate time.

The Arduino MEGA on the board is responsible for the management of this circuit, also keeping track of what happens on the Data and Address BUS and other control lines.

It also manages the LCD display and its keypad, the USB communication and the interface to the Apple-1 keyboard.

All of the Apple-1's BUS lines have been electrically decoupled to reduce the possibility of *ringing / cross talking* and allow other boards to be used simultaneously without any problems.

5. OPERATIONS

Apple-1-Stepper can be controlled in two ways: via the keypad under the LCD display and via the terminal connected to the USB interface of the Arduino MEGA.

At startup it is set up in a totally passive mode (FREE RUN) and does not perform any manipulation of the RDY line nor any reading of the BUS and its control signals.

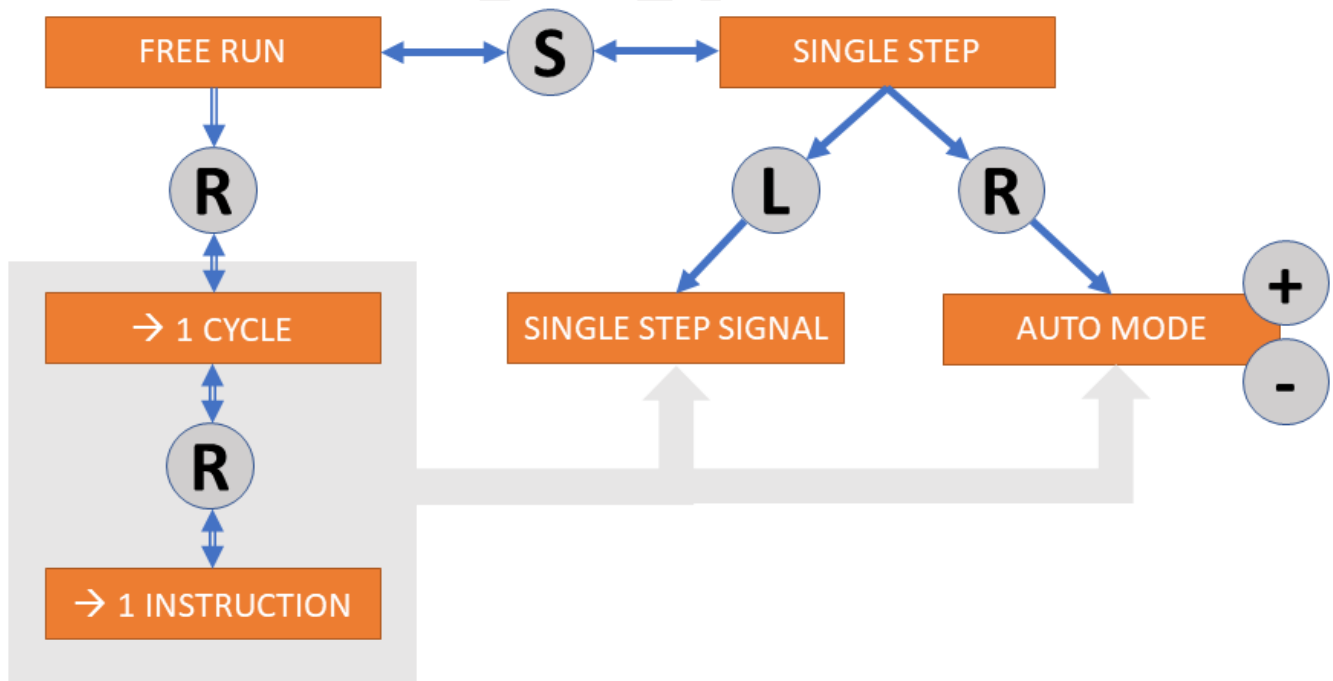
The device, as shown in the diagram in the manual, when in SINGLE STEP mode can operate in two modes:

- SINGLE CYCLE: at each activation, manual or timed, the 6502 performs a single clock cycle.
- SINGLE INSTRUCTION: Each time the 6502 is activated, either manually or by timer, it will execute as many clock cycles as necessary to complete the instruction being read.

This setting can only be made in FREE RUN mode. It will be considered by the device when it will be switched in SINGLE STEP mode.

5.1 OPERATIONS USING THE LCD DISPLAY BUTTONS

The following diagram shows the states of the device in relation to the pressure of the buttons SELECT/LEFT/RIGHT/UP (+)/DN (-).



In **FREE RUN** mode:

- Pressing the **RIGHT** button allows you to choose between the SINGLE CYCLE / SINGLE INSTRUCTION mode that will be used later. This selection does not affect the working mode of the device, which remains in FREE RUN.



P-LAB APPLE-1
RUN -> 1-CYCLE



P-LAB APPLE-1
RUN -> 1-INSTR

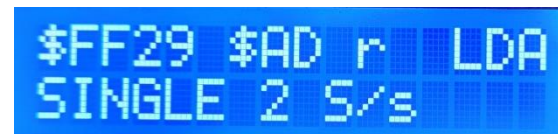
- Pressing the SELECT key **stops the microprocessor operations and puts it in SINGLE STEP mode**, waiting.



P-LAB APPLE-1
SINGLE 2 S/s

In **SINGLE STEP** mode:

- The pressing of the **LEFT** key sends to the microprocessor the signal to **execute a single clock cycle or a single instruction** as previously set in FREE RUN mode.



\$FF29 \$AD r LDA
SINGLE 2 S/s

At the same time, the contents of the Data and Addresses BUSes are shown on the LCD display in hexadecimal format. If the contents of the location are read during the fetch phase of the 6502, the corresponding Opcode is also displayed. Pressing the **LEFT** key repeatedly causes the program to be executed one step at a time.

- Pressing the **RIGHT** button puts the device in AUTO mode.



\$FF20 \$EB r BBB
AUTO 8 S/s

This mode allows you to automatically send the signal for the execution of a single cycle (or instruction).

It is possible to select between 1 and 900 steps/second.

The frequency is adjusted at will, using the **UP** and **DOWN** buttons according to preset values.

At high frequencies the LCD display will be ineffective, so it is suggested to use this option when connected to a terminal, in order to capture every single step.

A subsequent press of the **RIGHT** button puts the device back into manual mode.

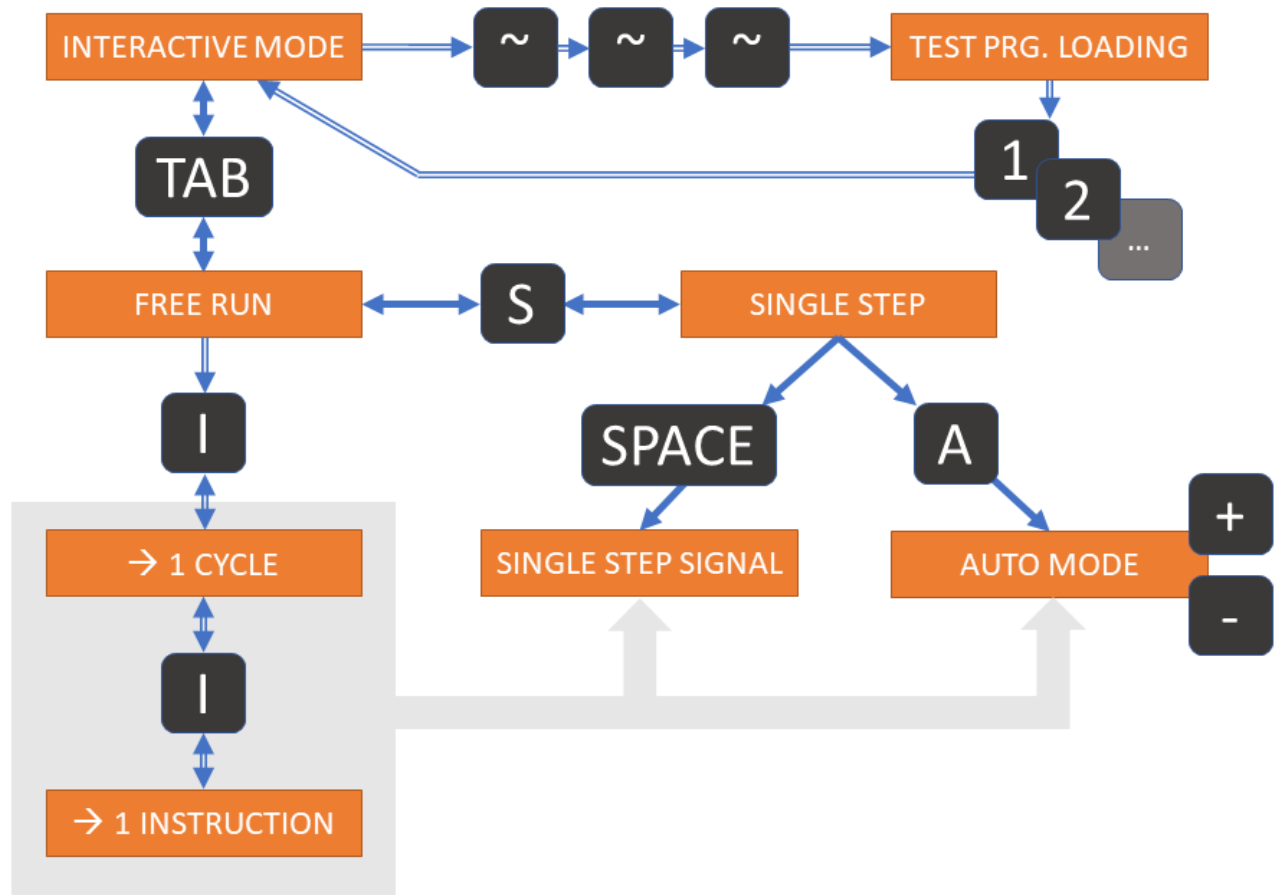
- Pressing the SELECT button sets the device to FREE RUN mode.

5.2 OPERATIONS VIA USB SERIAL TERMINAL

The operating modes of the *Apple-1-Stepper* can also be selected via a serial terminal connected through the traditional USB port of Arduino.

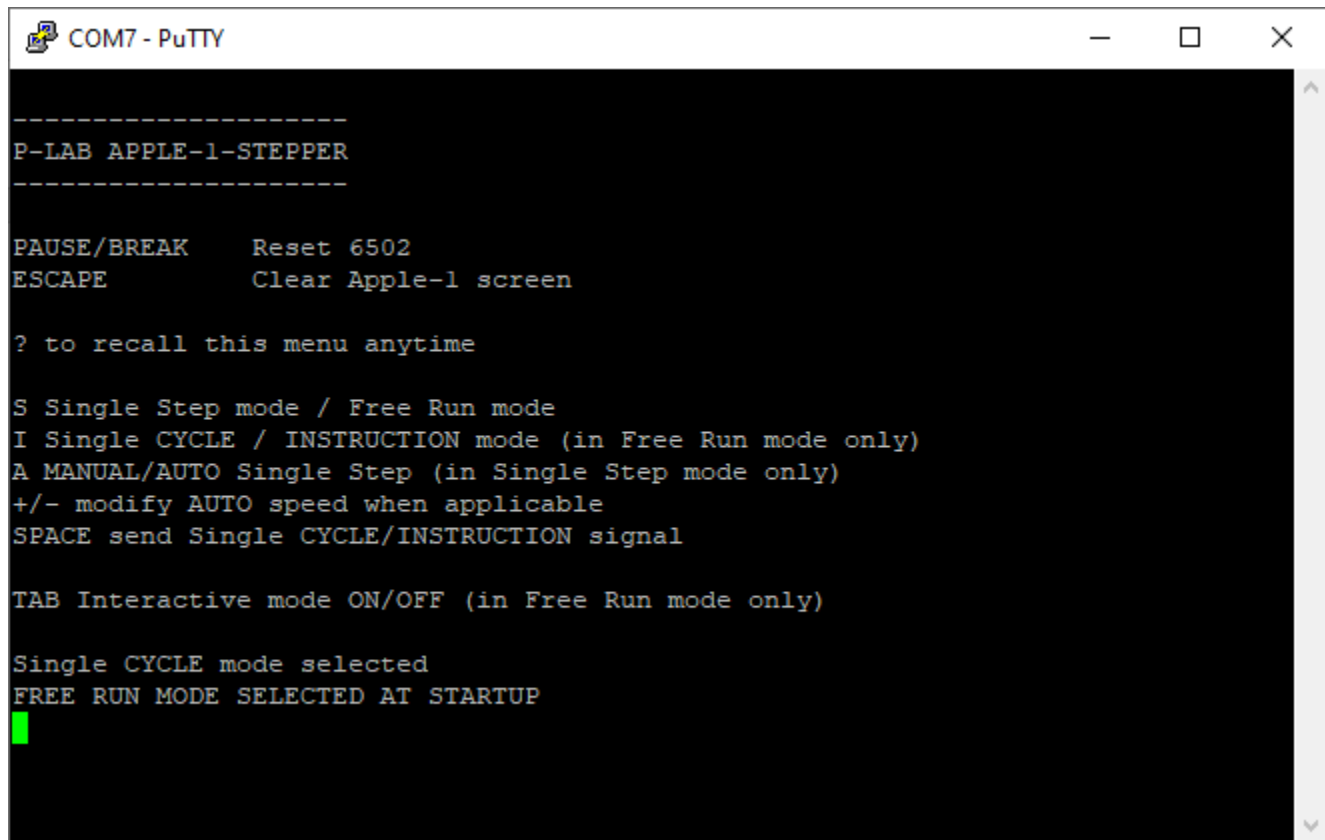
After connecting the cable according to the indications of paragraph 3, open a terminal program (e.g. PuTTY or minicom) and configure it to communicate with Arduino. The connection speed is **115200** baud.

Due to the increased functionality available in this mode, this is the functional diagram:



5.2.1 BASIC OPERATIONS

At startup, the device is set to command mode, and the following screen is displayed:



```
-----  
P-LAB APPLE-1-STEPPER  
-----  
  
PAUSE/BREAK      Reset 6502  
ESCAPE           Clear Apple-1 screen  
  
? to recall this menu anytime  
  
S Single Step mode / Free Run mode  
I Single CYCLE / INSTRUCTION mode (in Free Run mode only)  
A MANUAL/AUTO Single Step (in Single Step mode only)  
+/- modify AUTO speed when applicable  
SPACE send Single CYCLE/INSTRUCTION signal  
  
TAB Interactive mode ON/OFF (in Free Run mode only)  
  
Single CYCLE mode selected  
FREE RUN MODE SELECTED AT STARTUP  
█
```

In this mode the commands are expressed by a single letter, in particular:

S to switch from FREE RUN mode to SINGLE STEP mode (similarly to the SELECT key of the LCD display).

I to select the SINGLE CYCLE mode or the SINGLE INSTRUCTION mode, like the RIGHT key on the LCD display. This selection is only available in FREE RUN mode.

A allows you to switch to AUTO mode (when you are already in SINGLE STEP mode). Again, the **+** and **-** keys allow you to increase or decrease the speed of the operations.

SPACE, when pressed in SINGLE STEP mode (but not in AUTO mode) allows you to send the signal to the microprocessor to process a single cycle or a single instruction.

In Command mode the **?** key allows you to recall the summary. Each operation causes a confirmation message on the screen.

Two special keys have also been mapped as follows, and are always active:
(Please note: these keys may not work with some terminal programs).

PAUSE/BRK causes the 6502 and basically the whole Apple-1 to RESET, but not the Apple-1-Stepper. This feature is useful for seeing the consistency of the computer bootstrap, for example.

ESCAPE causes the Apple-1's screen to be erased. This key is effective only if the Apple-1-Stepper connection to the "keyboard" socket is present (see next sections).

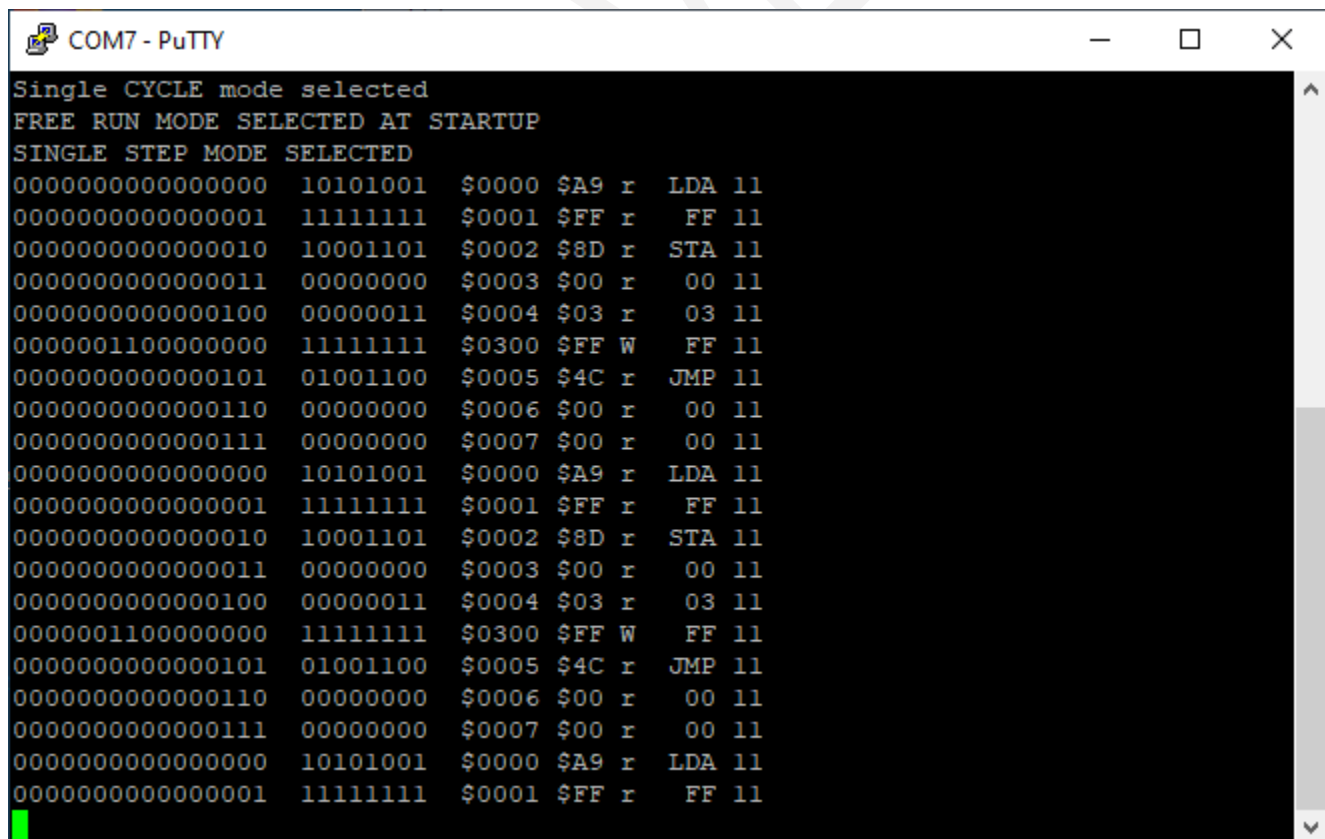
To explain how data are displayed on the screen, let us run a simple test program, which loads the FF value into the Accumulator then deposits it into memory location \$0300 and starts over:

```
0000 LDA #$FF
0002 STA $0300
0005 JMP $0000
```

The Machine Code (hexadecimal) to be entered in Apple-1 will then be:

```
0: A9 FF 8D 00 03 4C 00 00
```

If we run it with the OR command it will run in an infinite loop. Using Apple-1-Stepper in SINGLE CYCLE mode, this is what we will see in the terminal after pressing SPACE a bunch of times:



```
COM7 - PuTTY
Single CYCLE mode selected
FREE RUN MODE SELECTED AT STARTUP
SINGLE STEP MODE SELECTED
0000000000000000 10101001 $0000 $A9 r LDA 11
0000000000000001 11111111 $0001 $FF r FF 11
0000000000000010 10001101 $0002 $8D r STA 11
0000000000000011 00000000 $0003 $00 r 00 11
0000000000000100 00000011 $0004 $03 r 03 11
0000001100000000 11111111 $0300 $FF W FF 11
0000000000000101 01001100 $0005 $4C r JMP 11
0000000000000110 00000000 $0006 $00 r 00 11
0000000000000111 00000000 $0007 $00 r 00 11
0000000000000000 10101001 $0000 $A9 r LDA 11
0000000000000001 11111111 $0001 $FF r FF 11
0000000000000010 10001101 $0002 $8D r STA 11
0000000000000011 00000000 $0003 $00 r 00 11
0000000000000100 00000011 $0004 $03 r 03 11
0000001100000000 11111111 $0300 $FF W FF 11
0000000000000101 01001100 $0005 $4C r JMP 11
0000000000000110 00000000 $0006 $00 r 00 11
0000000000000111 00000000 $0007 $00 r 00 11
0000000000000000 10101001 $0000 $A9 r LDA 11
0000000000000001 11111111 $0001 $FF r FF 11
```

The information on the screen can be described as follows:

- The first sixteen bits represent the sixteen lines of the Addresses BUS;
- The next eight bits represent the eight lines of the Data BUS;
- The next two hexadecimal values are Address and Data converted from the previous binary values;
- The next field is the status of the R/W line. If it is "r" the microprocessor has read from the indicated address, if it is "w" the microprocessor has written the Data in the indicated location (**65C02 only**, see Appendix 1);
- Next, if applicable: the mnemonic code of the Opcode present in the data field is shown. The decoding occurs only if the microprocessor is in the fetch phase. Otherwise the pure hexadecimal value is shown.
- The last two bits represent the state of the IRQ line and the NMI line respectively. Both are normally unused, unless combined with expansion cards (for example: Apple-1 Wi-Fi Modem). Normally their value is 1.

INTERPRETING RESULTS

What is displayed in the screenshot above is a good example of how the various events are displayed and looks quite easy to follow the flow of the program. This approach can be useful when debugging programs written in *Assembly*, for example.

However, things can be a bit more complicated than that: many instructions take more than one clock cycle to be interpreted and executed. Unfortunately things can be even more misleading.

Let's take the following test program (without any usefulness), which after a `NOP` instruction first loads the value `$FF` into the Accumulator, then the value `$00`.

```
0000 NOP
0001 LDA #$FF
0003 LDA #$00
```

0: EA A9 FF A9 00 is the Machine Code. One would expect an equal number of clock cycles used by the two LDA instructions. But...

0000000000000000	11101010	\$0000	\$EA	r	NOP	11	
0000000000000001	10101001	\$0001	\$A9	r	A9	11	
0000000000000001	10101001	\$0001	\$A9	r	LDA	11	
0000000000000010	11111111	\$0002	\$FF	r	FF	11	
0000000000000011	10101001	\$0003	\$A9	r	LDA	11	
0000000000000100	00000000	\$0004	\$00	r	00	11	

3 Cycles

2 Cycles

You can clearly see that the first instruction takes three clock cycles to complete, the second only two.

The above is perfectly normal and is a feature provided by the 6502 design.

Basically, and without getting too technical, *under certain circumstances* the 6502 can read the next Opcode while it is still finishing executing the current instruction. This overlap feature has been designed to optimize access to the BUS and minimize the idle time.

The translation of Opcode `A9` in mnemonic `LDA` occurs -for the first instruction- only in the second line, that is when the microprocessor changes the value of the SYNC line that triggers Arduino's Opcode decode.

This behavior is perfectly normal and built-in the 6502 design.

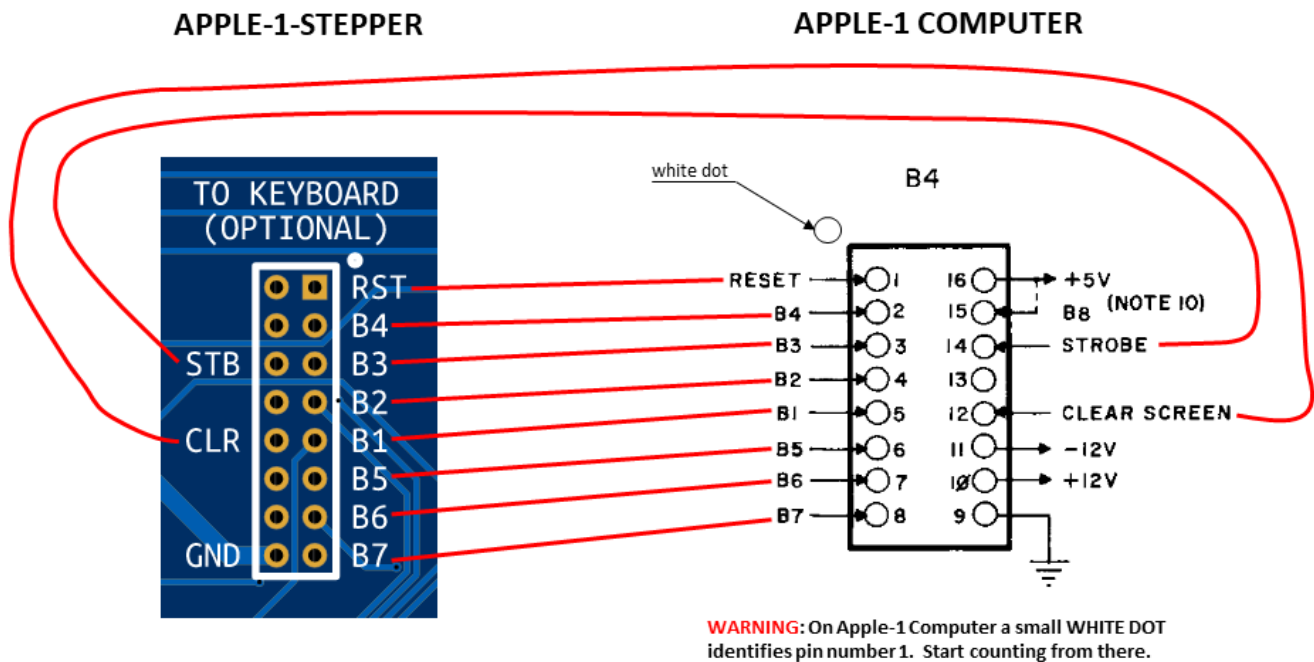
At the user interpretation level, nothing changes except that the *apparent* length in terms of used clock cycles can sometimes be one cycle shorter than expected.

5.2.2 ADVANCED OPERATIONS (INTERACTIVE MODE)

You can use *Apple-1-Stepper* to write your own programs directly on to the Apple-1 using the USB terminal.

This mode is called INTERACTIVE MODE.

In order to take advantage of it, it is necessary to connect the Apple-1-Stepper dedicated connector to the Apple-1 “keyboard” connector by using a dedicated cable. It must be arranged with the following connections highlighted in red:



As pointed out in the note, *pay special attention* to the orientation of the keyboard connector of the computer since from the observer's point of view it will be rotated 180 degrees compared to the drawing above.

Use the white dot on the PCB as a reference to count the pins.

WARNING! Incorrect connection will result in immediate and serious damages to the computer. Always triple check your connections before turning on the computer.

WARNING! Always make connections when the computer is switched off, taking all the precautions against electrostatic charge buildup already described in section 2 of this document.

It will only be possible to switch to INTERACTIVE MODE from FREE RUN mode, to do so simply press the **TAB** key. The following will appear:

```
NOW: Interactive Mode    ~~~ for programs
```

From that moment on, every character typed on the keyboard is sent to the Apple-1, which interprets it as if it had been typed on its own keyboard.

This feature is also useful for testing your own programs by loading them directly from an external computer, see Appendix 3. The output of the Apple-1 will always be the connected monitor.

Two short test programs are available. They are also transmitted automatically as if they were typed from the keyboard.

To access the loading menu, press the tilde **~** key three times in succession:

```
Please choose one:

1  S. Wozniak ASCII Characters test
   - run with 0R

2  M. Willegal RAM test
   - set limits in zero page (i.e. 0: 0 E0 0 F0)
   - run with 280R and wait for PASS/error
```

There are also brief instructions for using the programs, which are:

- 1) Character Testing, included in the Apple-1 Operating Manual
- 2) RAM Test, by Mike Willegal

Choose the desired program using the 1 or 2 key, and wait until the load finishes. Then run it according to the instructions.

To return to Command mode (FREE RUN), press the **TAB** key again:

```
NOW: Stepper Command Mode
```

6. EXAMPLE OF APPLE-1 BOOT AFTER A RESET COMMAND

```
-----  
P-LAB APPLE-1-STEPPER  
-----  
  
PAUSE/BREAK      Reset 6502  
ESCAPE           Clear Apple-1 screen  
  
? to recall this menu anytime  
  
S Single Step mode / Free Run mode  
I Single CYCLE / INSTRUCTION mode (in Free Run mode only)  
A MANUAL/AUTO Single Step (in Single Step mode only)  
+/- modify AUTO speed when applicable  
SPACE send Single CYCLE/INSTRUCTION signal  
  
TAB Interactive mode ON/OFF (in Free Run mode only)  
  
Single CYCLE mode selected  
FREE RUN MODE SELECTED AT STARTUP  
SINGLE STEP MODE SELECTED  
1111111100101001 10101101 $FF29 $AD r LDA 11  
1111111100101001 10101101 $FF29 $AD r AD 11  
0000000111101110 11111111 $01EE $FF r FF 11  
0000000111101101 00101001 $01ED $29 r 29 11  
0000000111101100 01100001 $01EC $61 r 61 11  
1111111111111100 00000000 $FFFC $00 r 00 11  
1111111111111101 11111111 $FFFD $FF r FF 11  
1111111100000000 11011000 $FF00 $D8 r CLD 11  
1111111100000001 01011000 $FF01 $58 r 58 11  
1111111100000001 01011000 $FF01 $58 r CLI 11  
1111111100000010 10100000 $FF02 $A0 r A0 11  
1111111100000010 10100000 $FF02 $A0 r LDY 11  
1111111100000011 01111111 $FF03 $7F r 7F 11  
1111111100000100 10001100 $FF04 $8C r STY 11  
1111111100000101 00010010 $FF05 $12 r 12 11  
1111111100000110 11010000 $FF06 $D0 r D0 11  
1101000000010010 01111111 $D012 $7F W 7F 11  
1111111100000111 10101001 $FF07 $A9 r LDA 11  
1111111100001000 10100111 $FF08 $A7 r A7 11  
1111111100001001 10001101 $FF09 $8D r STA 11  
1111111100001010 00010001 $FF0A $11 r 11 11  
1111111100001011 11010000 $FF0B $D0 r D0 11  
1101000000010001 10100111 $D011 $A7 W A7 11  
1111111100001100 10001101 $FF0C $8D r STA 11  
1111111100001101 00010011 $FF0D $13 r 13 11  
1111111100001110 11010000 $FF0E $D0 r D0 11
```

RESET VECTOR: \$FF00

BOOT STARTS AT \$FF00

7. CREDITS / ACKNOWLEDGEMENTS

This project was inspired by the work and videos of Ben Eater...

<https://eater.net>

...and by Erturk Kocalar of 8bitforce:

<https://www.8bitforce.com>

And, for course:

- Steve Wozniak for all his work, character test included 😊
- Mike Willegal for the RAM test program
<https://www.willegal.net/appleii/6502mem.htm>

Last but not least, for his support and collaboration from a distance:

- Massimiliano Larocca

Bibliography:

Apple-1 Operation Manual

<https://www.applefritter.com/files/a1man.pdf>

Western Design Center

https://www.westerndesigncenter.com/wdc/AN-002_W65C02S_Replacements.php

We hope you will enjoy using *Apple-1 -Stepper* !

APPLE-1-STEPPER

INFO [p-l4b @ protonmail.com](mailto:p-l4b@protonmail.com)

P-L4B @ PROTONMAIL.COM

APPENDIX 1 – NMOS 6502 vs CMOS 65C02

When the 6502 microprocessor was designed and developed, the ROM memories available on the market were much slower than the reading speed the microprocessor could achieve.

To ensure that the 6502 could read them as well, the RDY (Ready) function was implemented.

If this line is brought to logic level '0' under certain timings, the microprocessor suspends all activity, leaving the BUS and control lines in the state they are in, allowing the "slow" memory to complete its operations.

At the right moment, the RDY line is brought back to logic level '1' and the activities resume from where they stopped.

Oversimplifying, we can say that the RDY line behaves like a traffic light, but only for the microprocessor.

The rest of the traffic (the refresh of the 6502's registers, the refresh of the RAM, the generation of video signals, etc.) continues uninterrupted.

However, there are circumstances in which the semaphore is not considered: this is the case of memory WRITE operations.

If the RDY line is activated when the microprocessor is writing something to a certain memory area, the write operation will be completed, and operations will be suspended at the next read operation encountered.

Basically, all the write operations are always executed and never stopped or delayed.

They will be executed at full speed, because the NMOS 6502 was designed to work just like that.

Let's take a practical example.

The following example program in Assembly loads (LDA) in the Accumulator of 6502 the value \$FF, then deposits it (STA) in the location \$0300. Finally it performs an unconditional jump (JMP) to the beginning of the program (location \$0000), and then starts again.

```
0000    A9 FF      LDA #$FF
0002    8D 00 03   STA $0300
0005    4C 00 00   JMP $0000
```

In the memory of our Apple-1 we will therefore have the following content:

ADDRESS	DATA
0000	A9
0001	FF
0002	8D
0003	00
0004	03
0005	4C
0006	00
0007	00

When the program is run, the microprocessor *electrically* performs the following operations on the BUSES and R/W control line:

ADDRESS	R/W	DATA
0000	R	A9
0001	R	FF
0002	R	8D
0003	R	00
0004	R	03
0300	W	FF
0005	R	4C
0006	R	00
0007	R	00

As we can see, they are all Read operations because in fact the microprocessor is reading the program, except one line marked in bold with W (Write).

This is the operation with which the microprocessor *performs* the previous instruction STA \$0300: write the contents of the Accumulator (\$FF in this case) to memory address \$0300.

Considering what has been said, in which "steps" of our program the manipulation of RDY line will have effect, stopping the execution of the program?

ADDRESS	R/W	DATA	STOP
0000	R	A9	YES
0001	R	FF	YES
0002	R	8D	YES
0003	R	00	YES
0004	R	03	YES
0300	W	FF	NO
0005	R	4C	YES
0006	R	00	YES
0007	R	00	YES

The fact that the Write operation is performed at full clock speed prevents the Arduino from identifying it accurately and, above all, from reading the 16 address lines plus the 8 data lines in time before they change: it is too slow. Other solutions, based on different development platforms, might be successful, but they are out-of-scope in this project.

With only the RDY line manipulation (without touching the main Clock because it would have deleterious effects) it is therefore extremely complicated to capture Write events directly from the BUS at full speed.

We can therefore say that the ***write operations will be invisible to the Apple-1-Stepper.***

That said, the fact of not "seeing" the Data and Addresses BUSES during the Write phase takes away little value to the overall usefulness of the project, especially in terms of debugging programs or troubleshooting boards.

The 65C02 behaves differently from its predecessor: let's see how.

The 65C02 microprocessor is the CMOS version of the NMOS 6502 and is "almost" pin compatible with it, so that it can be easily adapted to run on Apple-1 computers (see Appendix 2).

Without going into the details of the differences between the two, which have been covered extensively by experts, we are interested in the fact that the **65C02 takes the RDY line into account** even during write cycles.

In the case of the previous example, therefore, it will be possible to stop the execution of the program at any point and display also the "physical" writing to address \$0300 by the STA instruction (and any other instruction that writes something somewhere, of course).

ADDRESS	R/W	DATA	STOP
0000	R	A9	YES
0001	R	FF	YES
0002	R	8D	YES
0003	R	00	YES
0004	R	03	YES
0300	W	FF	YES
0005	R	4C	YES
0006	R	00	YES
0007	R	00	YES

All memory operations are fully visible.

APPENDIX 2 – SUBSTITUTE NMOS 6502 WITH CMOS 65C02

The replacement of NMOS 6502 with CMOS 65C02 in the Apple-1 architecture is covered in the paper: https://www.westerndesigncenter.com/wdc/AN-002_W65C02S_Replacements.php

of which we report the main steps:

- 1.) VPB (Pin 1) - Carefully bend this pin outward before placing the chip in the socket.
- 2.) MLB (Pin 5) - Carefully bend this pin outward before placing the chip in the socket. The Apple-1 had pin 5 as VMA as part of the 6800 support. The 6502/W65C02 always has Valid Memory Addressing. The Apple-1 has a jumper that should be in place when using the 6502/W65C02S.
- 3.) BE (Pin 36) - Tie this pin to VDD (Pin 8).
- 4.) The Apple-1 has a 3K resistor pulling RDY (Pin 2) high. WDC does not recommend using the SToP or WAIt instruction on the Apple-1.

APPENDIX 3. PROGRAM LOADING VIA USB TERMINAL (Linux)

You can transfer programs using the INTERACTIVE MODE already described.

To do this:

- Leave the terminal program (e.g. minicom) open running as usual,
- Open another shell and with administrator privileges run the following script (let's suppose you called it `loader.sh`) followed by the name of the file you want to load on the Apple-1. E.g.:
`#sudo loader.sh demo.txt`
- Wait for the end of the loading process.

The script below reads the input file character by character and sends it to the Arduino MEGA which sends it to the Apple-1.

The pauses have been determined experimentally to cope with the limited input capabilities of the computer. The device file for the Arduino may be different, adjust it to your situation.

```
#!/bin/bash
#Send file to device character by character. Optimized for Apple-1
DEVICE="/dev/ttyACM0"
stty -F $DEVICE 115200 cs8 -cstopb -parenb
INPUT=$1
while IFS= read -r -n1 char
do
    if [ "$char" = "" ];
    then
        echo -n -e '\x8d' > $DEVICE
        sleep 0.5
        echo
    fi
    echo -n "$char" > $DEVICE
    echo -n "$char"
    sleep 0.06
done < "$INPUT"
exit 0
```