8-bit Microprocessor Trainer on SMART Response XE

The 8-bit Trainer is a ported version of my 8-bit Microprocessor Trainer available on Tindie (https://www.tindie.com/products/subsystems/8-bit-microprocessor-trainer/)

The chip shortage has hit even my little company and the computer chips I could get at about $2 each have jumped to $80. There is no way I can produce these boards for a reasonable price. Then I remembered another product I offer, the Arduino BASIC port to the SMART Response XE schoolroom computer. Why can't I adapt my 8-bit software to this device so I can continue to offer it? Turns out, there is no reason.

This device offers a couple of benefits over the original design. Because it has more EEPROM, I can now store 10 separate programs instead of 2. The GUI is pretty sleek looking. It is a battery powered unit so you are not tethered to a USB. Finally, it is easily hackable if you grow tired of the 8-bit computer simulation.

On the main screen, I have mimicked the control switches of the original design.

INC: Save the current instruction to the current address, increment to the next address, display the data there. (The del key also functions as INC. It is in a comfortable place for entering code)

RST: Reset. This will bring the computer back to address 00. It will also break out of a program while running. (note: the DLYA and DLYB use the Arduino delay(ms) function which is a blocking function. You may need to wait until a long delay is over before you can break)

RUN: Runs the current program from the current address. If you want to run from the beginning, be sure to use RST before RUN.

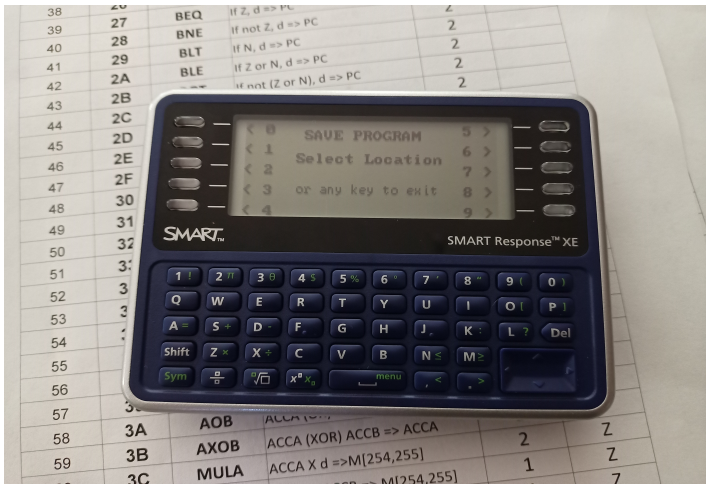
ADD: Move the Program Counter to the address currently displayed in the data HEX.

PWR: This will power down the device. To wake it up, press the button on top of the unit.

CLR: This will clear the current program (set all addresses to 0 and set the current address at 00) This command will require you to verify after you press CLR.

SAV: Save the current program to one of the ten slots available.

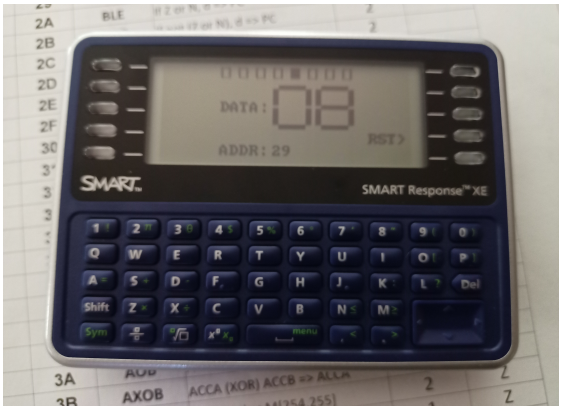
LOD: Load program in one of the ten slots to current memory.



Since the programs are stored in the units EEPROM, it will take a couple of seconds to save. Loading is quick.

If you press SAV or LOD and want to cancel, press any other key besides the function keys along the LCD.

When you run a program, the screen will look as follows:



The current address of execution will always be show on the bottom. RST is available on the right to exit the program. Any data posted to the display will show up on the 8 boxes on the top (this mimicks the 8 LEDs of the original trainer) and on the large HEX display. There are a few other times that options will appear on the right. For instance, when you use INA (input data into ACCA), there will be a little ENT shown on the lower right. When you have placed the desired data on the HEX display, push the ENT function button to enter it. Or when you select the WAIT instruction, you will see a GO on the lower right function button. To continue program execution, select the GO.

I have also added a few new instructions. There were a couple I really wish I had as I was developing

programs for the computer. They are:

CLS: This replaces the BRT command in the original. CLS will clear the contents of the HEX display during runtime.

VBAT: This replaces the original TMP instruction as there is no temperature sensor on the SRXE. This will return a value of the battery voltage into ACCA.

The original computer didn't have a STACK. I wrote in a 32 byte stack for you to be able to push and pull data from. Unlike most microprocessors, this does not use your available RAM. I create a separate array to hold these values so you don't get the fun of tracking whether your stack is starting to overwrite your program.

PSHA: Push ACCA to the STACK.

POPA: This will pop off a byte of data from the STACK and place it in ACCA.

PSHB: Push ACCB to the STACK.

POPB: This will pop off a byte of data from the STACK and place it in ACCB.

PSHY: Push REGY to the STACK.

POPY: This will pop off a byte of data from the STACK and place it in REGY.

Finally, I added the ability to call a subroutine. When you use CALL, the Program counter will be loaded with the next byte of your program (the address of your subroutine).

CALL: Call a subroutine.

RTN: Return from the subroutine. When you call this instruction, the Program counter will be loaded with the address saved when the subroutine was called. Program execution will pick up after the subroutine call.

I hope you enjoy this unit and learn about how computers work on the lowest level (machine language). Below are some sample programs to get you started. Also, see the documentation for the original unit to get more of a beginners introduction.

1: Battery Voltage Measurement:

The following program will get the battery voltage and return a displayed number between 99 and 0 to give you an idea of the battery level. Recommend loading this into slot 9 and tracking voltage periodically to get an idea of when you are really getting low.

Address	Data	Command
00	**0B**	LDY
01	**FF**	
02	**09**	LDA
03	**00**	
04	**1C**	AIM
05	**14**	SWP
06	**08**	VBAT
07	**30**	SUBA
08	**8D**	
09	**11**	DECA
0A	**27**	BEQ
0B	**1B**	
0C	**14**	SWP
0D	**0E**	INCA
0E	**18**	CMPA
0F	**0A**	
10	**27**	BEQ
11	**15**	
12	**14**	SWP
13	**22**	JMP
14	**09**	
15	**1B**	MIA
16	**0E**	INCA
17	**1C**	AIM
18	**0C**	CLA
19	**22**	JMP
1A	**12**	
1B	**1B**	MIA
1C	**32**	SFLA

1D	**32**	SFLA
1E	**32**	SFLA
1F	**32**	SFLA
20	**2F**	ADAB
21	**00**	AOUT
22	**24**	STOP

2: Guess the number:

The computer generates a random number and you need to guess it. Make your guess by entering a HEX number and click ENT. If you are too high, the left most bit will be 1 and the HEX display will show 80. If you are too low, the HEX display will show the right most bit as will be 1 and the display will show 01. When you guess correctly, the display will light all bits and show FF on the HEX display.

Address	Data	Command
00	**3E**	RNDA
01	**14**	SWP
02	**03**	INA
03	**1A**	CMAB
04	**29**	BLT
05	**0E**	
06	**27**	BEQ
07	**14**	
08	**09**	LDA
09	**80**	
0A	**00**	AOUT
0B	**23**	WAIT
0C	**22**	JMP
0D	**02**	
0E	**09**	LDA
0F	**01**	
10	**00**	AOUT
11	**23**	WAIT
12	**22**	JMP
13	**02**	
14	**09**	LDA
15	**FF**	
16	**00**	AOUT
17	**24**	STOP

3: Solar System Flyby:

This is a program I developed for my STARSHIP STEM learning kit. It simulates a high speed (really high speed) flyby of the solar system starting at the Sun. It has you fly in your trans-light ship from the Sun to Neptune in just 1 minute. When you run the program, the display will show all bits lit up for 2 seconds and then you launch. Each time you pass a planet, the next bit will light up. Isn't it amazing how close the inner planets are to the outer?

Address	Data	Command
00	**0A**	LDB
01	**14**	
02	**09**	LDA
03	**FF**	
04	**00**	AOUT
05	**26**	DLYB
06	**09**	LDA
07	**01**	
08	**0A**	LDB
09	**08**	
0A	**45**	CALL
0B	**29**	
0C	**0A**	LDB
0D	**07**	
0E	**45**	CALL
0F	**29**	
10	**0A**	LDB
11	**06**	
12	**45**	CALL
13	**29**	
14	**0A**	LDB
15	**0B**	
16	**45**	CALL
17	**29**	
18	**0A**	LDB
19	**49**	
1A	**45**	CALL

1B	**29**	
1C	**0A**	LDB
1D	**62**	
1E	**45**	CALL
1F	**29**	
20	**0A**	LDB
21	**C5**	
22	**45**	CALL
23	**29**	
24	**0A**	LDB
25	**CB**	
26	**45**	CALL
27	**29**	
28	**24**	STOP
29	**26**	DLYB
2A	**00**	AOUT
2B	**32**	SFLA
2C	**46**	RTN