6800 Microprocessor Pocket Trainer

In this project we explore building a trainer for the Motorola 6800 microprocessor. The system uses no 6800 hardware. Instead, the CPU, ROM and RAM are simulated using a Raspberry Pi Pico. The second core handles I/O for keypad entry and a 7-segment character map that is placed on an OLED 128x32 i2C display. This I/O is mapped to RAM addresses.

The 6800 Processor

8-bit Registers: Accumulator A, Accumulator B, Condition Codes CC

16-bit Registers: Program Counter PC, Index Register X, Stack Pointer SP

I/O, RAM, ROM: Memory mapped to 64K x 8-bits

Although obsolete, the Motorola 6800 microprocessor is simple with only a few registers. This makes it ideal for learning about assembly coding. A few assembly language examples are covered. However, the focus is an overview of the trainer showing how to build and use it.

Resources:

https://github.com/simple-circuit/6800-Trainer

https://github.com/simple-circuit/picobug

https://deramp.com/downloads/

https://github.com/adafruit/Adafruit_SSD1306

https://github.com/CalPlug/Heathkit_ET-3400

Motorola, DS9471-R1

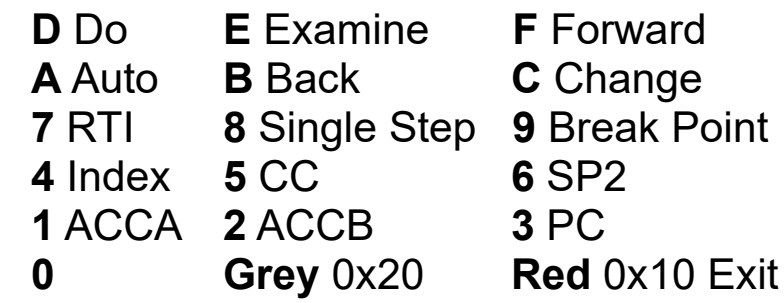
Ron Bishop, “Basic Microprocessors and the 6800”, Hayden,1979

Motorola, “M6800 Microcomputer System Design Data”, 1976

If you’re interested in building and programming the trainer, here are some resources. The one at the top contains everything needed to build the project.

The trainer can operate in the keypad entry mode or via the USB or serial UART interface. A slide switch allows reset booting to either mode. The trainer mode is standalone using keypad data entry and a simulated 7-sement display for output. Booting to the trainer mode brings you to the command input mode. Commands start with a single key press followed by hex data, address data or a key press to return to command mode.

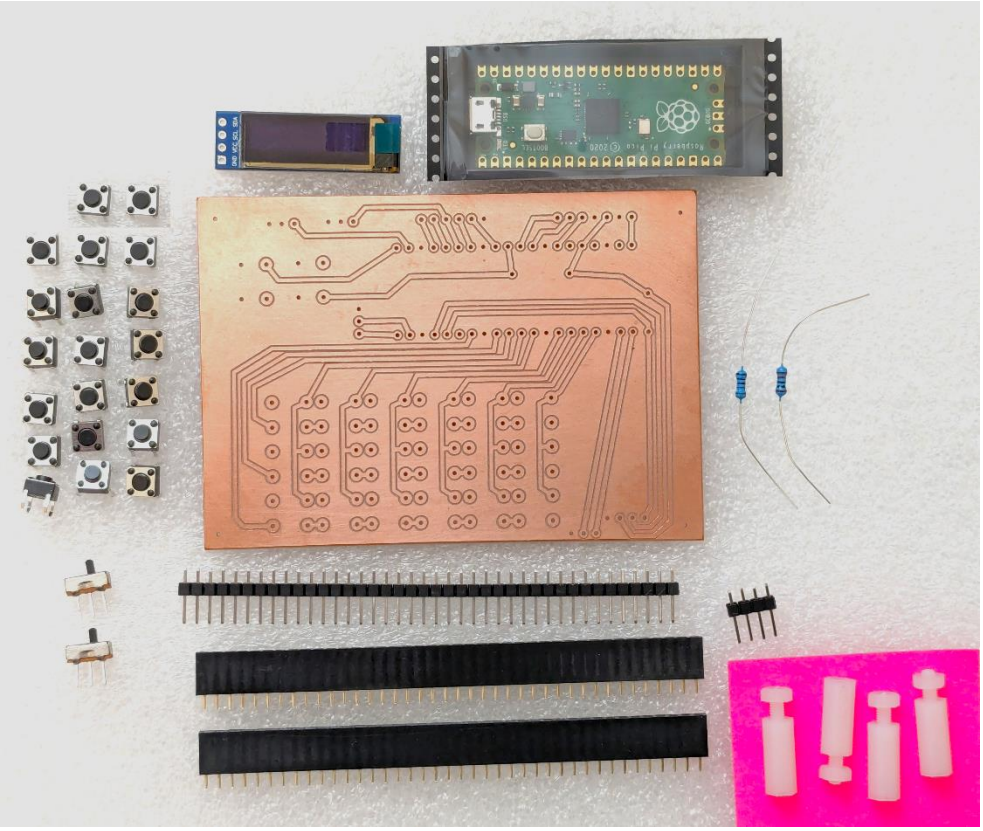
Key	Display	Data Entry
1	ACCA==hh	C to Change accumulator A, other keys return to Command Mode
2	ACCB==hh	C to Change accumulator B, other keys return to Command Mode
3	PC==hhhh	C to Change Program Counter, other keys return to Command Mode
4	ir==hhhh	C to Change Index Register X, other keys return to Command Mode
5	hiNoVc hh	C to Change Condition Code Register, other keys return to Command Mode
6	hhhh==SP	Any key return to Command Mode
7	RTI	Use after exit from single step to command mode to run the rest of the program. Use after a break point interrupt or user inserted 15 op code.
8	Single Step	Program starts at PC value. Press 8 for next step, else command mode.
9	BR. hhhh	Inserts SWI at address hhhh and saves the op code. After the interrupt, use RTI command. Inserting another breakpoint restores the op code and Adds a SWI at the new address.
A	ADDR_ _ _ _	Enter a hex address then hex data for that address. The address will auto Increment after each data entry. Press the red key twice to exit.
C	==h.h. or =h.h.h.h	Change the data or address for the current command. The dots indicate you are in the change mode.
D	do _ _ _ _	Start program execution at the entered hex address.
E	Addr_ _ _ _	Examine memory at entered hex address. Shows aaaa==dd C Change aaaa==d.d. B go back one address F go forward one address



Keypad cheat sheet.

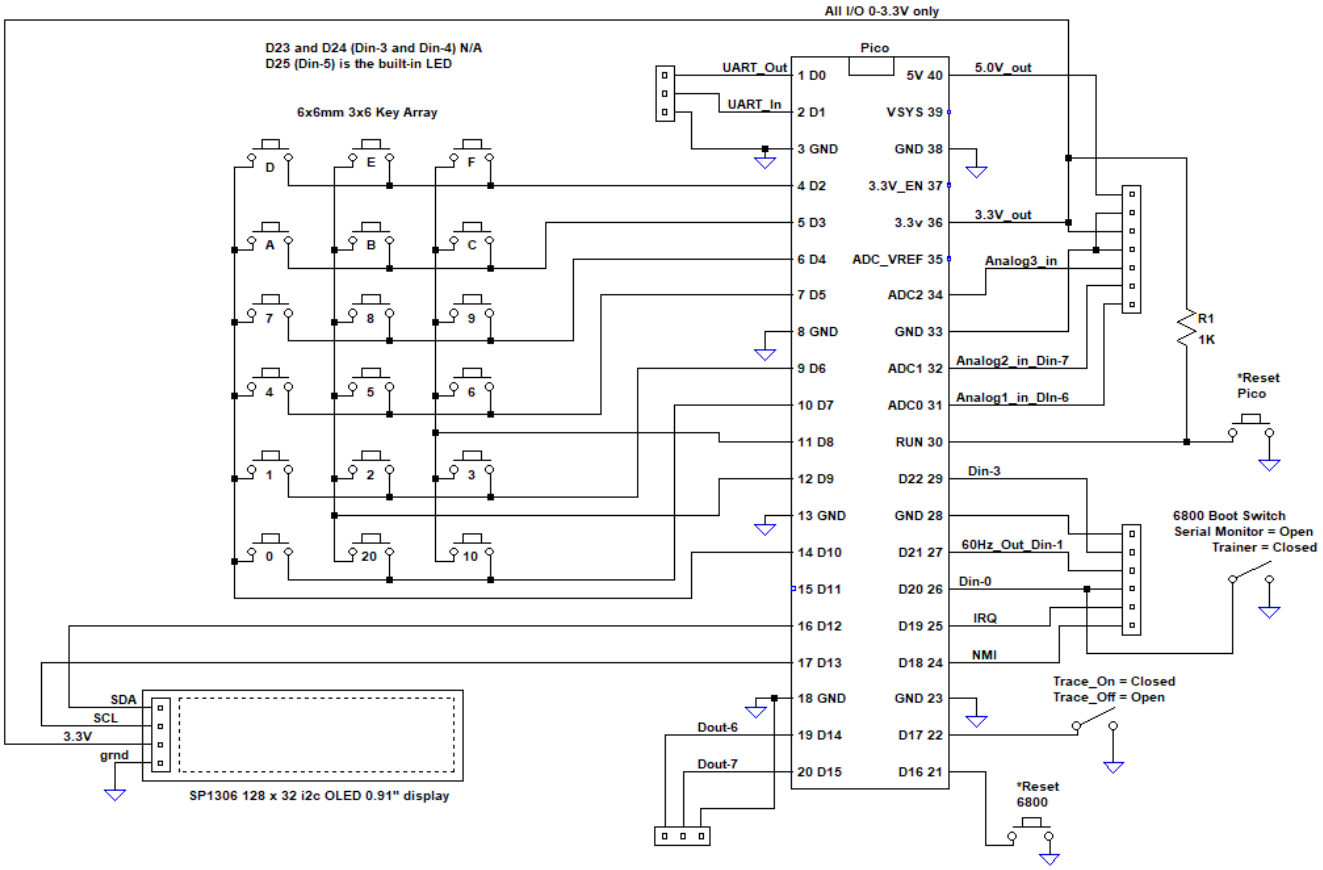
Note: Although this trainer operates like the Heathkit® ET-3400, the supplied monitor program is not compatible. The M6800 emulator implements added op codes. It uses a different process to call and perform keypad functions. However, the hardware appears at the proper memory locations.

Project information and source code may be found at https://github.com/simple-circuit/6800-Trainer . In addition to this small pocket 100mmx70mm version, plans are available for a larger 120mmx180mm board with a 7-segment LED display and 12x12mm keys.

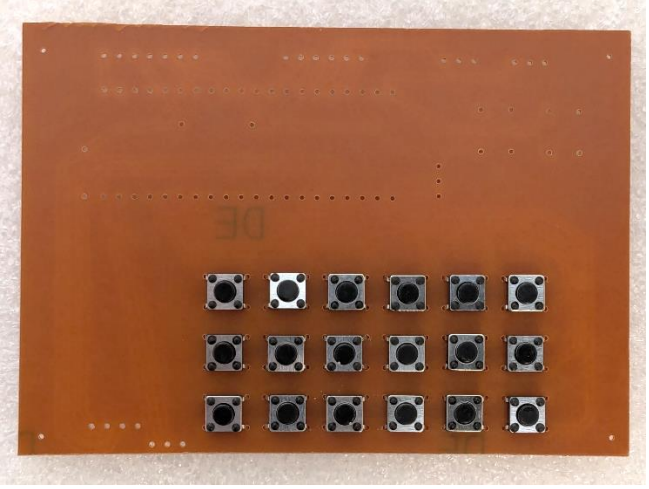


Here is what you need:

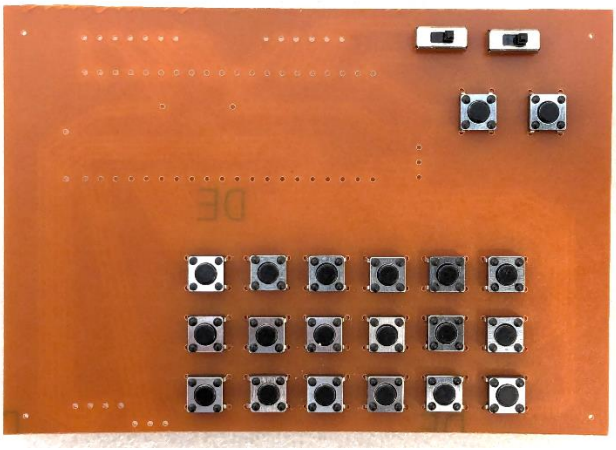
20 – 6x6mm 4-pin push button switches
2 – DPST slide switches with 0.1” pin spacing
1 – 1K resistor
1 – 10K resistor
2 – 40-pin female socket strips 0.1” spacing
1 – 40-pin male header 0.1” spacing
1 – SP1306 128x32 i2c OLED display 0.91”
1 – 4-pin male header with 0.1” spacing
1 – Raspberry Pi Pico RP2040 processor
1 – USB cable
1 – CNC milled 70x100mm 1.5mm thick single sided copper board
    Drill_tiny.gcode 0.8mm CNC drill file
    Profile_tiny.gcode 0.2mm 30° engraver mill
4 – M3 Spacers and nuts



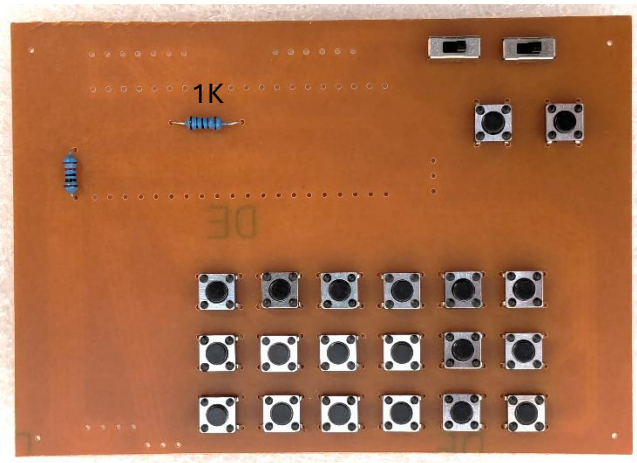
A schematic diagram for the trainer.



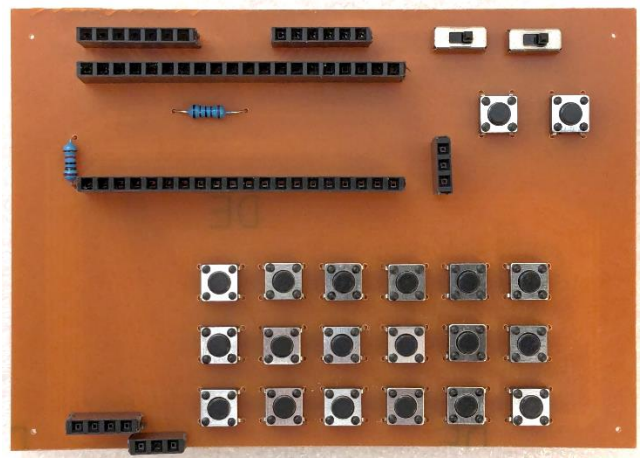
Install and solder the 18 push button switches. This is an array of 16 hex keys and two extra keys. One is used to exit the Auto data entry mode.



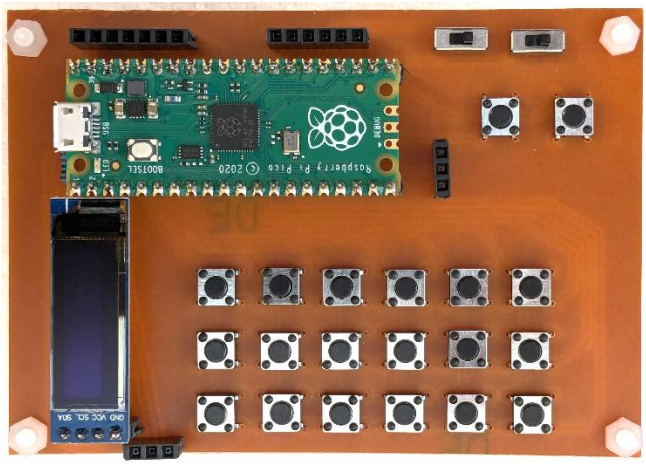
Next, install the two reset buttons switches and the two mode slide switches.



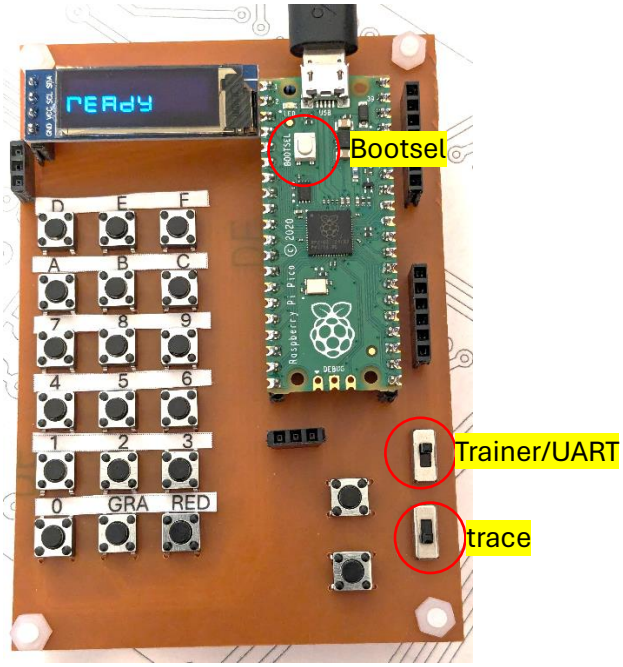
The 1K pull up resistor is used on the Pico reset line. The 10K pull up is optional. Pull up resistors are present on the display module.



Cut the 40-pin sockets to make the following strips: two 20-pin, two 3-pin, one 4-pin, one 6-pin and one 7-pin. Install and solder the socket strips.

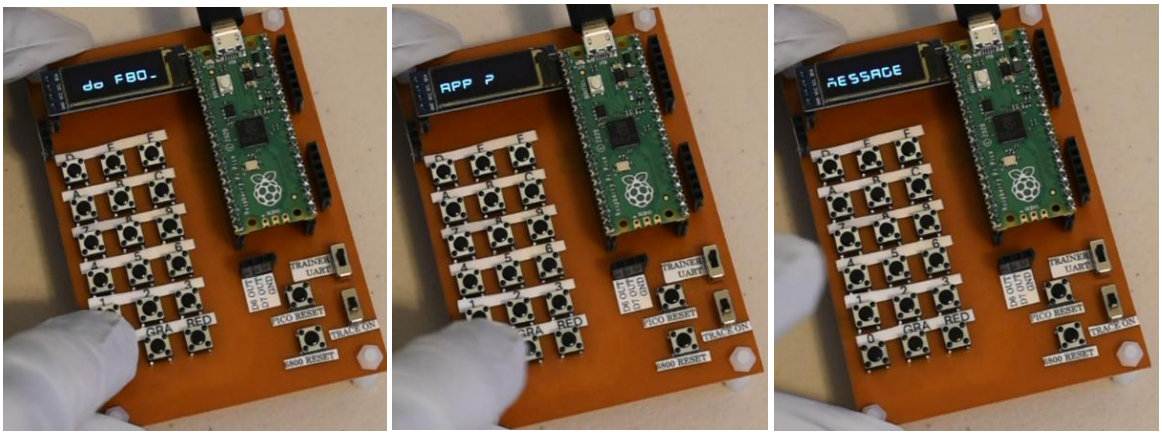


Cut the male header strip into two 20 pin sets. Install them into the 20-pin sockets. Install the 4-pin male header into the 4-pin female socket. Place the Pico on the header strip. Note the orientation. Then solder it to the pins. Place the display on the 4-pin header and solder it.

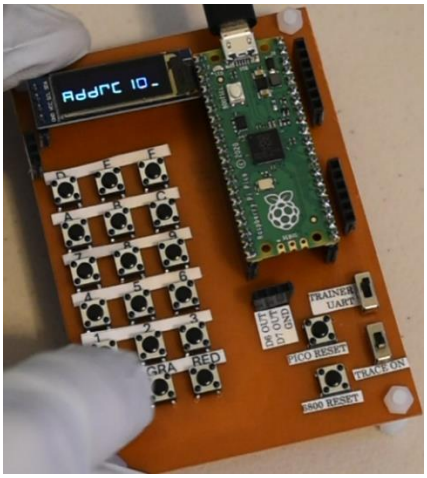


Place the trace slide switch in the down off position. Place the trainer/UART switch in the trainer up position. Connect a USB cable between the Pico and your computer while holding the white bootsel button down. Release the button. Drag the **sim680b_trainer_i2c_s.uf2** file into the Pico boot folder. The 4-pin OLED display has no reset line. It may occasionally fail to reset. Remove then plug in the USB cable to reset the display.

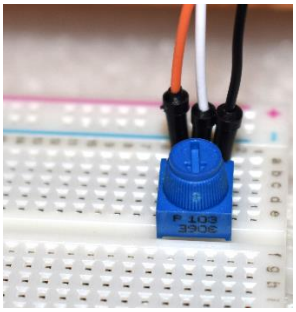
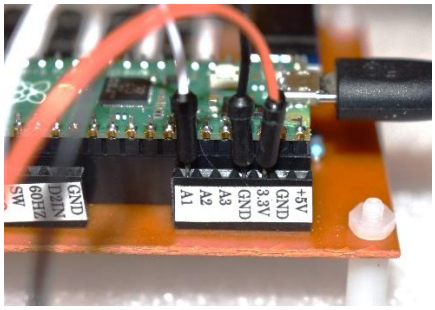
The mini_labels.pdf file contains print strips that can be cut into key labels. Use double sided sticky tape to attach. Fit them above the key row you are marking.



The trainer has three example programs in ROM memory. There is a program located at address 0xF800 that will move the programs to RAM memory. Press the D ‘Do’ key then press F 8 0 0. The program will prompt you for a key press. Press key 1 to load the message program. To run it, press D ‘Do’ then 0 0 0 0. The trainer starts execution at address 0x0000.

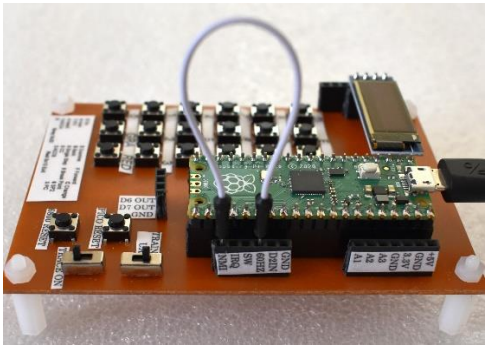


The display simulates an 8-character 7-segment display. To display in a standard font, press E ‘Examine’ then C 1 0 9. Press C ‘Change’ then 0 1. This loads 01 into memory C109 turning off the 7-segment line and enabling the text line. To clear the old display line, press F ‘Forward’ then C ‘Change’ and 1 1. Now press D ‘Do’ 0 0 0 0 to run the program with a text display.

Program 2 reads ADC channel A1 and converts the 10-bit binary value to the decimal voltage. Connect the end termination of a 1K potentiometer to ground. Next, the wiper connects to A1. Last, connect the other end termination to +3.3V. Do not connect to 5V as it will damage the input.

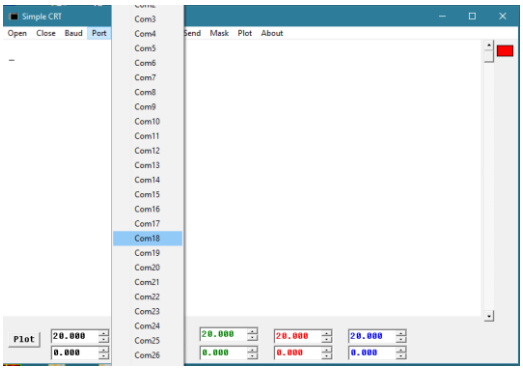
D ‘Do’ F 8 0 0 AND select App 2. D ‘Do’ 0 2 0 0 to run the program. Move the potentiometer to vary the voltage. The display shows the voltage which lies between zero and 3.3V.

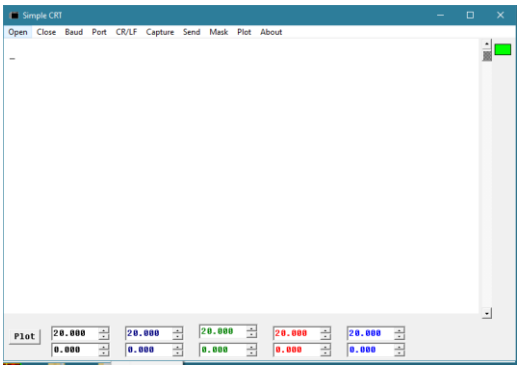


Press 6800 Reset button to exit the voltmeter. D ‘Do’ F 8 0 0 and select 3. Connect a wire to the 60Hz output pin. D ‘Do’ 0 2 0 0 then connect the other end of the wire to the NMI pin. This generates a non-maskable interrupt every 16.67ms. It will be used to time a 5-minute chess clock.

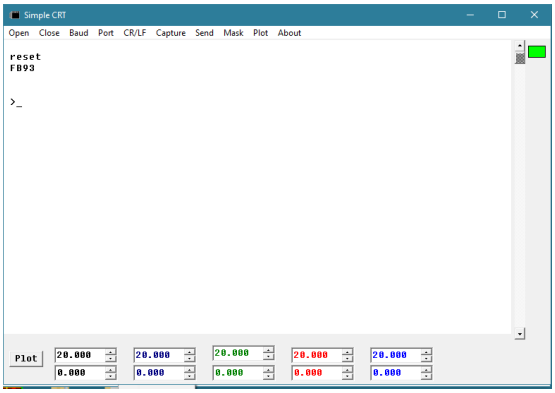
Press the zero key to start player-2's clock. When player-2 completes the move, pressing any key other than zero starts player-1's clock. The player's flag is fallen when the clock reads zero.

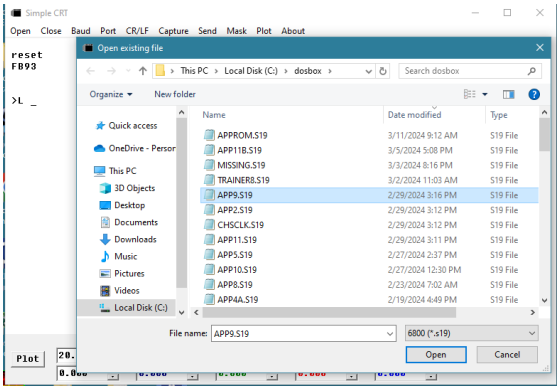
Press the 6800 reset button. The trainer returns to command mode but then the clock appears. This is because the interrupt routine is still running. Remove the jumper wire to stop interrupts.



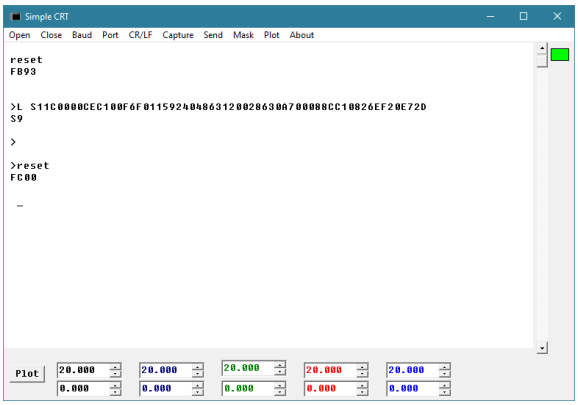


In Windows, start simpleCRT.exe and select your serial Port. Then, click open. The red box should turn from red to green. Slide the switch to UART and press 6800 reset.

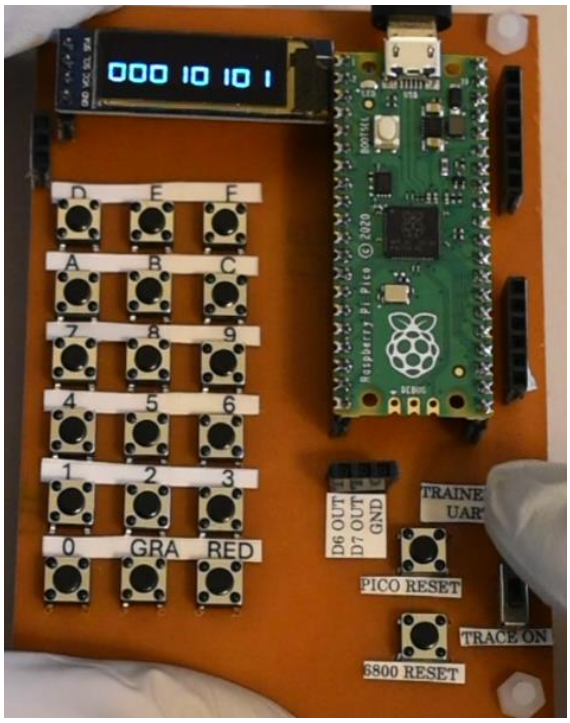




'reset' and FB93 are displayed. The hex value is the vector address for serial monitor program. Type 'L' at the prompt. Click Send then File. Select the *.S19 file type and locate the app9.S19 file. Open it. This sends a Motorola S-record file containing an assembly program.



Move the slide switch to Trainer and press the 6800 Reset button. Note the vector address is FC00 which is the trainer monitor program.



Press D ‘Do’ then 0 0 0 0 to run the program. It displays the binary values for the bits in the I/O located at hex address F011.

Starting from the left to right:

B7 = input A2
B6 = input A1
B5 = LED 0 is off, 1 is on
B4 = Vbus Power (should be 1)
B3 = PS mode (should be 0)
B2 = D2in
B1 = 60Hz square wave Clock, 16.67ms period
B0 = Trainer/UART slide switch. 0 is trainer, 1 is UART. You can use as a GP input in UART position.

Sliding the Trainer/UART switch will change bit b0. Bit b1 will toggle between zero and one as the 60Hz clock changes. You can pole this input for timing. Connect a wire to D2in and touch it to ground. This changes bit b2. Analog inputs A1 and A2 are associated with bits b6 and b7. They work like GP digital inputs but they are not pulled high. They just float.

Note: Only apply zero to 3.3V to the inputs.

The purpose of this video is to introduce the hardware that can simulate a 6800 based system. For those not familiar with coding a 6800 processor, you can start with a few immediate operand instructions. Immediate instructions operate on the memory byte following the single byte instruction. A few accumulator instructions are inherent as they need no additional memory input. This is because only the accumulator and condition codes are affected.

OP	Operand	Code	Operand	Comment
LDAA	#$C3	86	C3	Load accumulator A with hex C3
ADDA	#$44	8B	44	Add accumulator A to hex 44
ADCA	#$5A	89	5A	Add with carry to accumulator A hex 5A
ANDA	#$F0	84	F0	Logic AND accumulator A with hex F0
ORAA	#$80	8A	80	Logic OR accumulator A with hex 80
EORA	#$0F	88	3F	Exclusive OR accumulator A with hex 3F
SUBA	#$20	80	20	Subtract hex 20 from accumulator A
SBCA	#$52	82	52	Subtract with borrow from accumulator A
CMPA	#$7F	81	7F	Compare A to hex value 7F
RORA		46		Rotate accumulator A one bit right through the carry
ASRA		47		Shift accumulator A one bit right inserting same sign for bit-7
ASLA		48		Shift accumulator A one bit left inserting 0 for bit-0
ROLA		49		Rotate accumulator A one bit left through the carry
DECA		4A		Subtracts one from accumulator A
INCA		4C		Add one to accumulator A
TSTA		4D		Test accumulator A and set condition codes e.g. C, N, Z
CLRA		4F		Load zero into accumulator A
NEGA		40		Two’s complement accumulator A
COMA		43		One’s complement accumulator A
BRA *		20	FE	Branch -2 addresses after instruction (branch to self)

Example using LDAA immediate instruction with the program terminating in a branch always to self:

With the trainer showing ‘rEAdy’, press the A ‘auto’ insert key. Enter address 0000. The trainer waits for data entry starting at address 0000 hex. Enter the following bytes: 86, 4B, 20, and FE. Press the red button twice to exit back to ready. Press 3 to examine the program counter it should be set to 0000. If not use C ‘change’ and set it to zero. Press 0 to return to ready.

Press 8 to ‘single step’ the program. The op code at address 0000 is executed then the trainer returns showing the next in line op code. It is at address 0002 and the 20 op code is a branch always. The FE operand at address 0003 is the value -2. When the branch always instruction completes, the program counter is at address 0004. The instruction subtracts -2 from the program counter and execution resumes at address 0002. Press 8 a couple more times and the single step always returns to address 0002. Press 0 to return to the ready prompt.

Pressing 1 shows that 4B was loaded into accumulator A. Press D for ‘do’ and enter the start address 0000. The display shows ‘uSEr’ to indicate the programming is running from the user stack. It will endlessly loop to address 0002. Press the 6800 reset button to get the ready prompt.

Pressing 1 shows accumulator A has 00 not the 4B value loaded by the program. After pressing the 6800 reset button, the trainer program clears registers and assigns default values to the stack pointers. The single step operation runs one instruction op code and all registers are saved on the stack for display and use for the next single step.

Since the program counter was reset to zero, pressing the ‘8’ key will single step through the load instruction. Press 0 to get to the prompt, then press 1 to examine accumulator A.

When assembling programs by hand (looking up values), a coding sheet helps to organize the process. A coding sheet and example filled sheet are provided to help you write your own programs.

Micro Computer Coding Sheet

Page ____ of ____

Name____ Date _________________

Program____

Address	Op Code	Operand	Label	Instruction	Operand	Comment

Micro Computer Coding Sheet

Name_______Simple Circuit_____________________________ Date __________________

Program______Load Accumulator A____

Address	Op Code	Operand	Label	Instruction	Operand	Comment
0000				ORG	$0000	DEFINE START ADDRESS
0000	86	4B	START	LDAA	#$4B	LOAD HEX 4B IMMEDIATE TO A
0002	20	FE	HERE	BRA	HERE	BRANCH ALWAYS -2 RELATIVE
0004						BRANCH STARTING LOCATION IS
						NEXT INSTRUCTION ADDRESS 0004

Memory Maps:

RAM: $0000 to $DFFF with holes for I/O
ROM: $E000 to $FFFF with holes for I/O

RAM area Memory mapped peripherals handled by core 2:

$C000 Key Value
$C003 Key Column 3
$C005 Key Column 2
$C006 Key Column 1
$C100-$C107 write ASCII value to display (mapped to 7-seg memory, cleared after write)
$C108 LED on/off
$C109 $00 = Text and 7-seg, $80 = 7-seg only, $01 = text only
$C10A a non-zero write clears 7-segment memory
$C120-$C127 digit7 7-segments g-a and dp (persistent)
$C130-$C137 digit6 7-segments g-a and dp
$C140-$C147 digit5 7-segments g-a and dp
$C150-$C157 digit4 7-segments g-a and dp
$C160-$C167 digit3 7-segments g-a and dp
$C170-$C177 digit2 7-segments g-a and dp
$C180-$C187 digit1 7-segments g-a and dp
ROM memory mapped areas handled by 6800 simulator core 1:

$F000 UART Status register b1 transmit ready, b0 receive ready (load with extended address only)
$F001 UART Transmit data (store with extended address only)
$F003 UART Received data (load with extended address or ANDB instructions)
$F010 I/O out bits b7 and b6 (store with extended address or STAA 0,X indexed address)
       Writes to D15 and D14 Pico pins
$F011 I/O in bits b7 to b0 loads D27 to D20 Pico pin values
$F020 ADC address 1,2 or 3 (store with extended address or STAA 0,X indexed address) to start
       Load returns ADC high byte (10-bit data)
$F021 Load returns ADC low byte

Tips on assembly with CRASM:

All of the 6800 programs for the 6800 trainer were assembled using the Motorola asmhc11.exe assembler program running in dosbox. This program is no longer available for download. As an alternative, Linux users can install then run the cross assembler crasm version 1.8-1. However, the cross assembler is not fully compatible with the Motorola pseudo mnemonics.

To convert from the Motorola Assembler format to crasm:

- Add these options to the beginning of the program:
  - CPU 6800
  - OUTPUT SCODE
- Comment lines: replace:
  - All * with ;
- ORG $1234, where $1234 is a hex address, use:
  - * = $1234
  - CODE
- Replace: END with CODE
- Replace form constant character:
  - FCC 'abcd' is now ASC "abcd"
- Replace form constant byte:
  - FCB change to DB
- Reserve memory byte RMB is not supported. Instead use:
  - label = address
  - For the next label, adjust the address for the required number of bytes of the previous label

Assembly example:

Text assembly program is in folder '6800 trainer'. The input file name is app2x.ASC. The S-record output file is app2x.s19. The -s option suppresses warnings (which can be excessive for comments). The list output is redirected to the file app2x.lst.

pi@raspberrypi:~ $cd '6800 trainer'

pi@raspberrypi:~/6800 trainer $ crasm -o app2x.s19 -s app2x.ASC >> app2x.lst