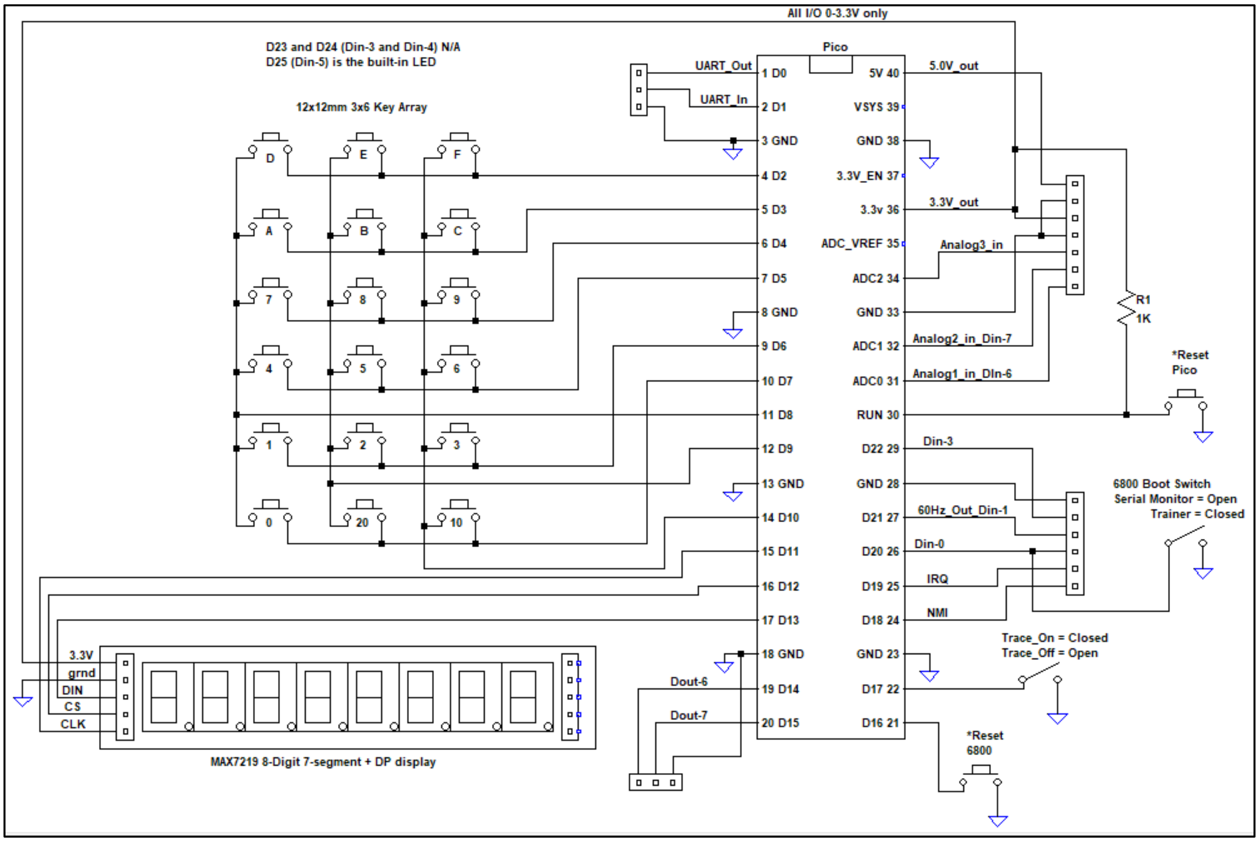
Pico 6800 Trainer Project

- Goals:
  - Memory address compliant with Heathkit® Model ET3400 M6800 microprocessor trainer keypad and 7-segment LED display.
  - Simulate the M6800 processor using a Raspberry Pi Pico and the Picobug M6800 program.
  - Use the second core on the Pico to perform keypad input and display output.
  - Minimize component count.
  - Write a trainer monitor program for memory/register examine and change.
  - Make new opcodes to simplify single stepping and switching between a user and system stack pointers.
  - Allow reset booting to a serial port monitor program or the trainer keypad monitor program.
    - Keep RAM programs intact.
  - Provide some example assembly language programs.
    - Display the binary value of input pins.
    - Display an ASCII text message.
    - Make a 5-minute chess clock using a 60Hz interrupt.
    - Read an ADC channel and display the decimal value.

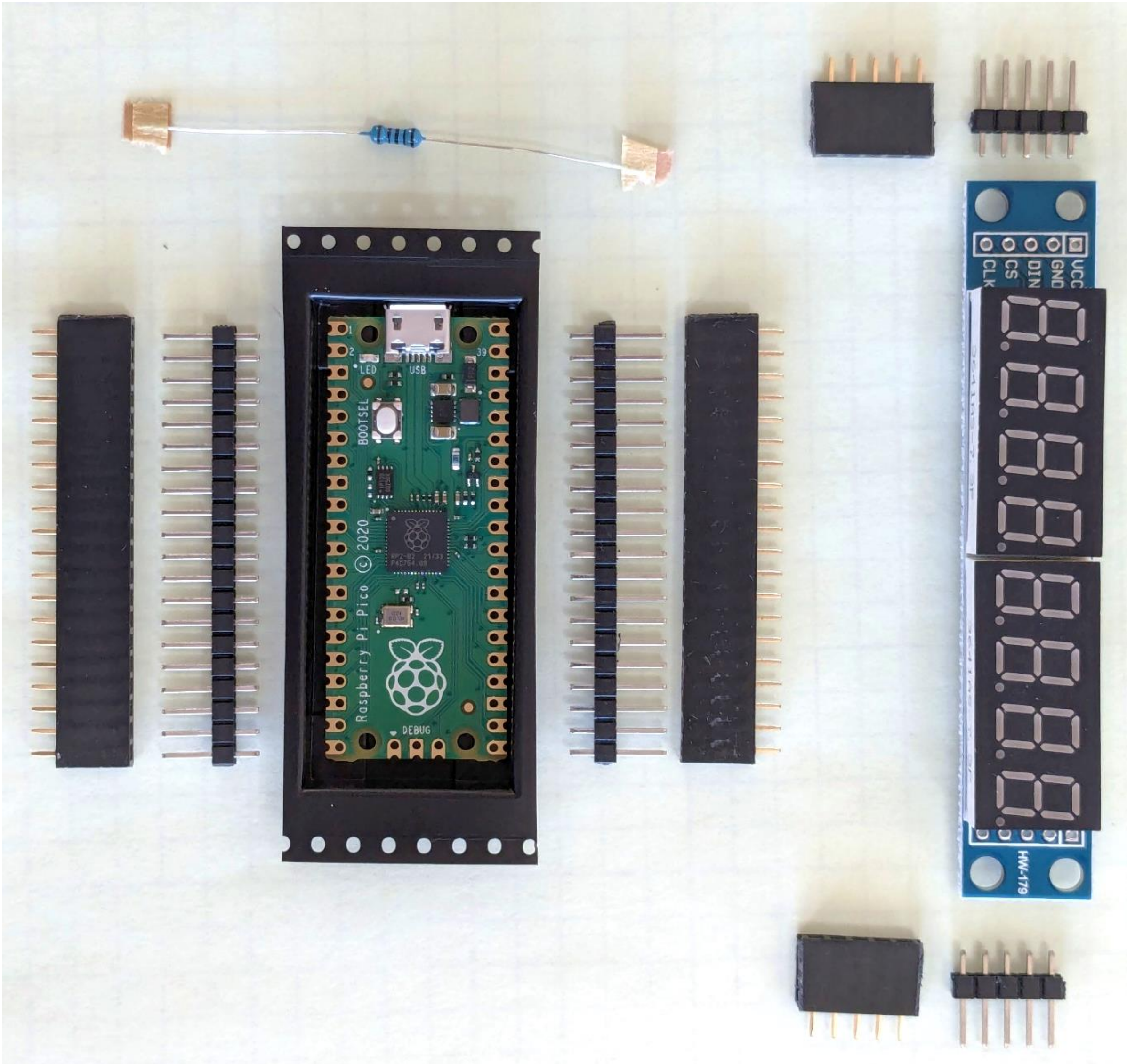


The project consumes much of the Pico I/O for key switch input and display output. Only D14 and D15 are available for digital output. Up to six digital inputs are present. All are dual use except for D22. The boot switch position is read on D20. A 60Hz square wave can be polled on D21. The LED state is available at D25 and two of the analog inputs at D26 and D27 may also be used as digital inputs.

When operating in the serial I/O mode, data is transferred through the USB and simultaneously through the UART at pins 1 and 2. The UART runs at 9600 BAUD. Either port can be used.

Key scanning uses pull up on digital inputs and scan outputs are switched from In to Out to generate an open collector type output that only pulls low. This eliminates external resistors and diode switching used in the Heathkit® trainer.

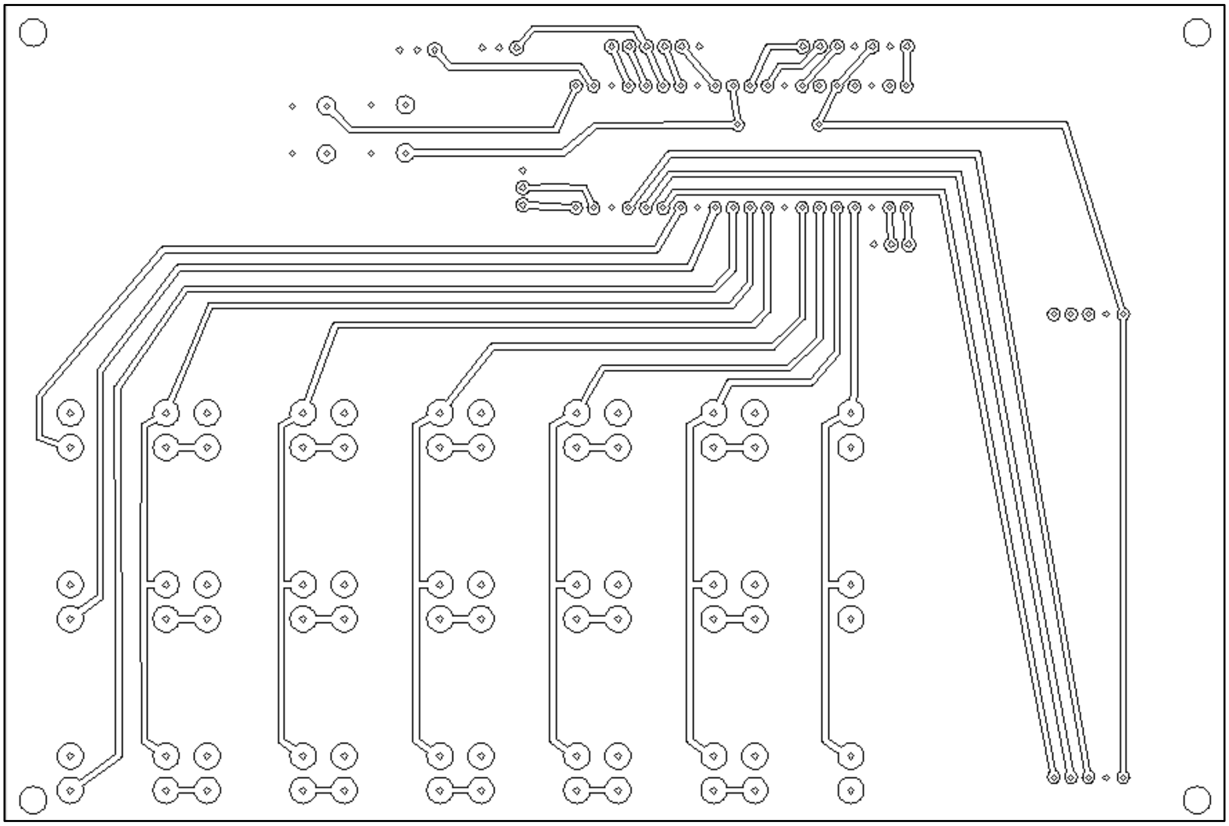
Two additional digits are present in the display for a total of eight. Memory mapping is used to make the serial module appear as addressable latches. The caveat is that the display is updated at a 1ms rate and not the memory write rate.



Parts:
1 - 1K resistor
2 – 5-pin female socket strips
2 – 5-pin male headers
2 - 20-pin female socket strips
2 – 20-pin male headers
1 – Pico RP2040 processor
1 – Max 7219 8-digit seven-segment plus decimal point display board

Parts:

18 – 12x12mm N.O. Momentary push button switches with caps (you need to make labels)
2 – 6mm N.O. Momentary push button switches
2- SPDT slide switches with 0.1” (2.54mm) pin spacing
4 – M3 spacers and screws
2 – 3-pin female socket strips
1 – 6-pin female socket strip
1 – 7-pin female socket strip



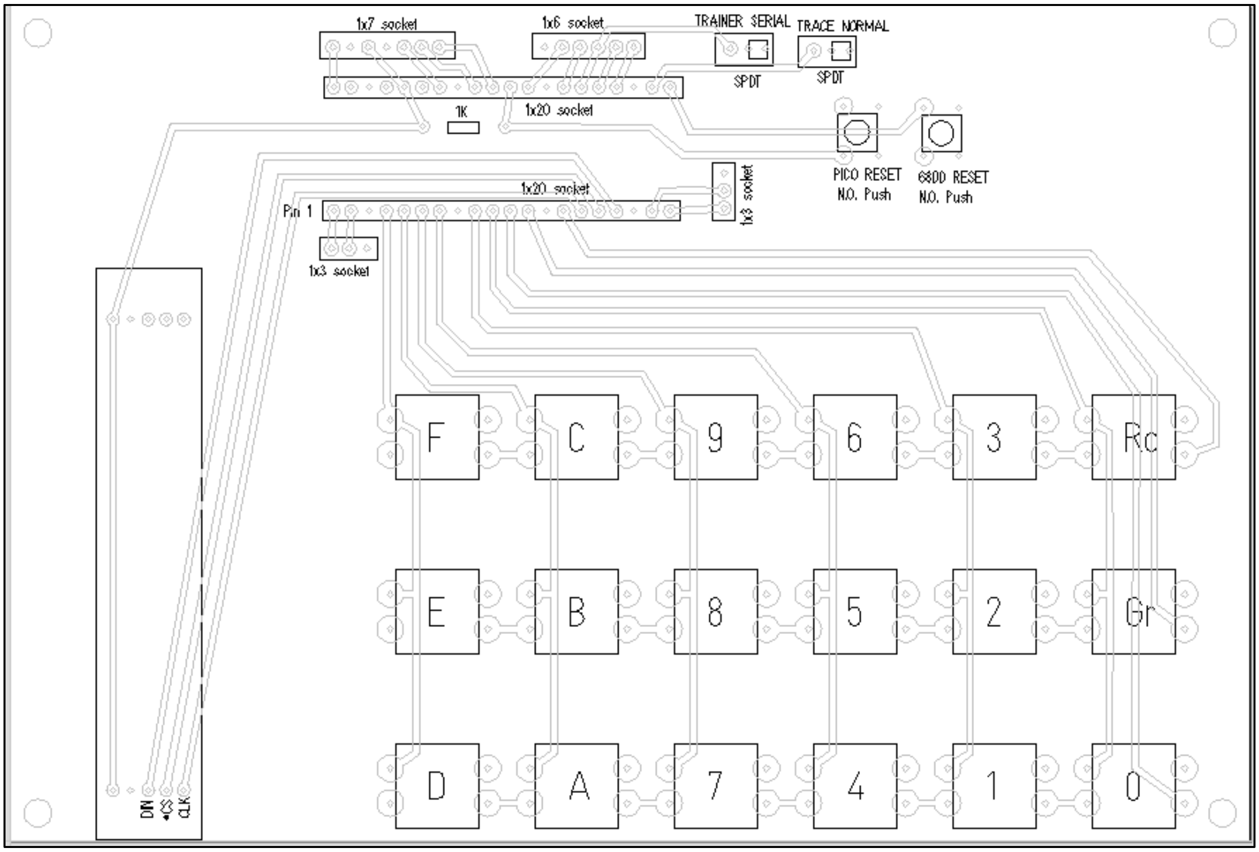
Parts:
1 – CNC milled circuit board 120mm x 180mm x 1.5mm single sided copper FR4

Files:

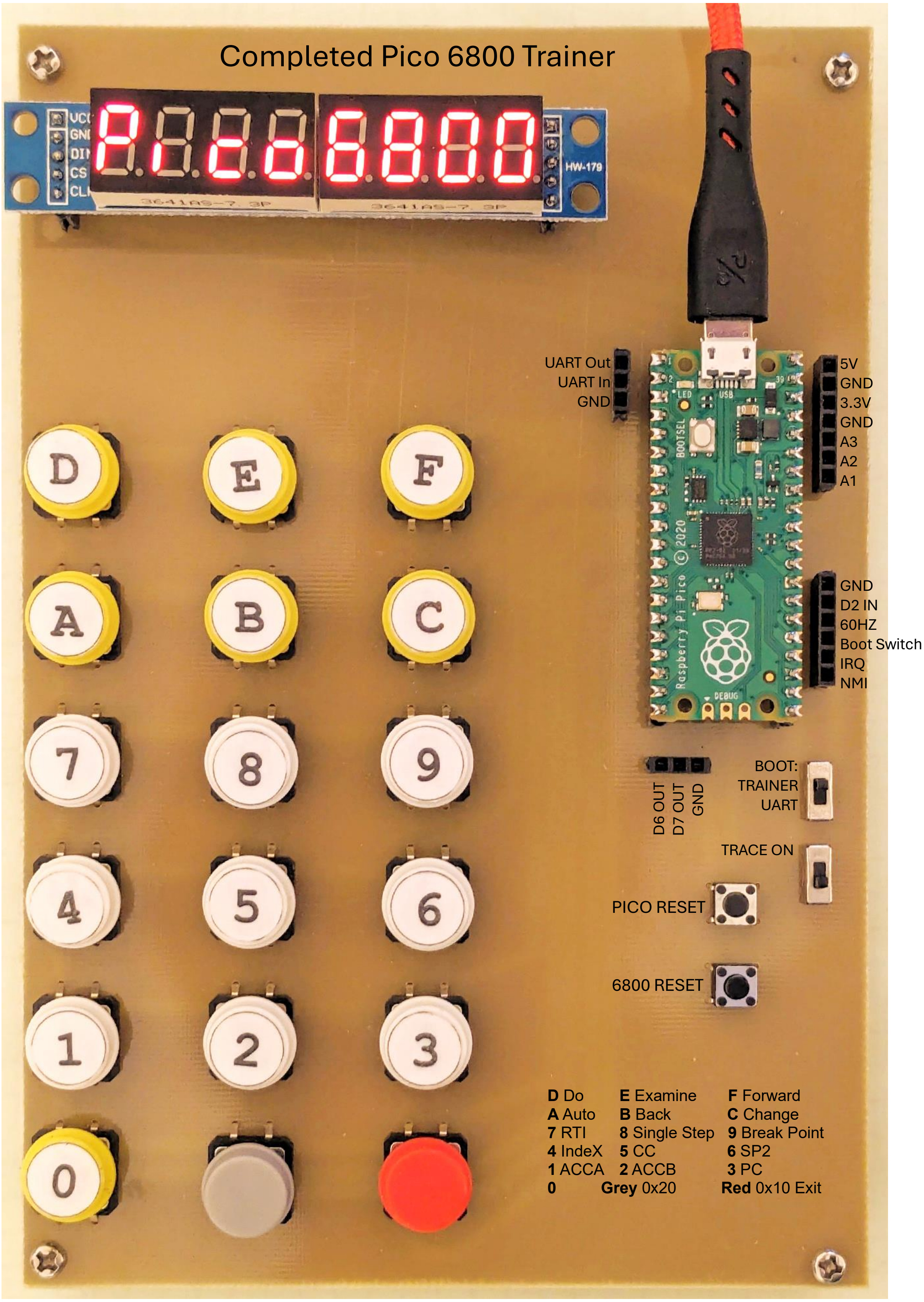
Profile_trainer.gcode	Profile cut for 0.2mm 30° engraving mill
Drill1_trainer.gcode	Drill for 0.7mm bit
Drill2_trainer.gcode	Drill for 0.9mm bit
Drill3_trainer.gcode	Drill for 1.1mm bit

If you do not have access to a CNC PCB mill, it is possible to hand wire the circuit on a breadboard with 0.1” hole spacing. The 12mm key switches will require drilling of the board. However, smaller 6mm switches can be used.

The completed board is shown on a following page. The ”key labels.pdf” file is a print sheet for the keys, switches, sockets and shortcuts. Protect the print with some clear tape on top. Uses double sided tape on the back. Cut out the labels and stick them to the keys and PCB.



A top view of the board showing the layout for the components.



Completed Pico 6800 Trainer

To simplify coding the trainer monitor, new Op Codes were added to the simulator. A second stack pointer is used to keep track of user code. The regular stack pointer is used for system monitor code.

SS2: op code **$02**

Called from system monitor program to run one opcode for user program

- pushes registers on SP
  - PCL → [SP]; SP- -
  - PCH → [SP]; SP- -
  - XL → [SP]; SP- -
  - XH → [SP]; SP- -
  - ACCA → [SP]; SP- -
  - ACCB →[SP]; SP- -
  - CC → [SP]; SP- -
- pulls registers from SP2
  - SP2++; [SP2] →CC
  - SP2++; [SP2] →ACCB
  - SP2++; [SP2] →ACCA
  - SP2++; [SP2] →XH
  - SP2++; [SP2] →XL
  - SP2++; [SP2] →PCH
  - SP2++; [SP2] →PCL
- executes opcode
  - OP(PC)
  - PC updated for next, branches, jumps and interrupts
- pushes registers on SP2
  - PCL → [SP2]; SP2- -
  - PCH → [SP2]; SP2- -
  - XL → [SP2]; SP2- -
  - XH → [SP2]; SP2- -
  - ACCA → [SP2]; SP2- -
  - ACCB →[SP2]; SP2- -
  - CC → [SP2]; SP2- -
- pulls registers from SP
  - SP++; [SP] →CC
  - SP++; [SP] →ACCB
  - SP++; [SP] →ACCA
  - SP++; [SP] →XH
  - SP++; [SP] →XL
  - SP++; [SP] →PCH
  - SP++; [SP] →PCL

Returning control to system monitor functions.

T2S: op code **$03**

Transfers Index register X to SP2

T2X: op code **$04**

Transfers SP2 to Index register X

RS2: op code **$05**

Run code from SP2

pushes registers on SP

pulls registers from SP2

RS1: op code **$15**

Run code from SP. Used instead of RTI to return to system monitor program.

pushes registers on SP2

pulls registers from SP1

If you use an assembler to generate code, use the **FCB** operator to generate the new op code byte.

The Trainer Monitor program has four basic functions: 1) Read a key, 2) Display a value, 3) Change a value and 4) Run code. The operation of each key function is broken down into pseudo code:

- Key 1:
    - Print string “ACCA==”
    - Get 8-bit value from [SP2+3]
    - Print as 2 hex digits
    - If next Key==C
        - Decimal Points On
        - Get 2 hex values
        - Save at [SP2+3]
    - Else Exit

- Key 2:
    - Print string “ACCB==”
    - Get 8-bit value from [SP2+2]
    - Print as 2 hex digits
    - If next Key==C
        - Decimal Points On
        - Get 2 hex values
        - Save at [SP2+2]
    - Else Exit

- Key 3:
    - Print string “PC==”
    - Get 16-bit value from [SP2+6]
    - Print as 4 hex digits
    - If next Key==C
        - Decimal Points On
        - Get 4 hex values
        - Save at [SP2+6]
    - Else Exit

- Key 4:
    - Print string “ir==”
    - Get 16-bit value from [SP2+4]
    - Print as 4 hex digits
    - If next Key==C
        - Decimal Points On
        - Get 4 hex values
        - Save at [SP2+4]
    - Else Exit

- Key 5:
    - Get 8-bit value from [SP2+1]
    - Test bits b5 b4 b3 b2 b1 b0
        - If set, print condition code letter h i n 0 v c
        - Else print _
    - Print 2 hex values
    - If next Key==C
        - Decimal Points On
        - Get 2 hex values
        - Save at [SP2+1]
    - Else Exit

- Key 6:
    - Get SP2 value
    - Print 4 hex values
    - Print string “==SP”
    - Wait for Key press then Exit

- Key 7:
    - Print string “user   ”
    - Execute RS2 instruction (run using stack 2)

- Key 8:
    - Execute SS2 instruction (run single instruction using stack 2)
    - Print 4 hex values for address of next instruction
    - Print string “==”
    - Print 2 hex values for next OP code

- Key 9:
  - Print string “br. _ _ _ _”
  - Get 4 hex values for address of break point insertion
  - If previous address == 0xFFFF
    - Save Op code from insertion address
    - Save insertion address
    - Insert RS1 (run from stack) Op code 0x15 at address
    - Exit
  - Else
    - Restore saved Op code at previous address
    - Save new insertion address
    - Save Op code from insertion address
    - Insert RS1 (run from stack) Op code 0x15 at address
    - Exit
- Key A:
  - Print string “Addr_ _ _ _”
  - Get 4 hex digits
  - Loop
    - Print 4 hex digits (current memory address)
    - Print string “==”
    - Print 2 hex digits (value at current memory address)
      - Decimal Points On
    - Get 2 hex values
      - Store new value at current memory address
      - Increment memory address
      - If one next two key values was 10 or 20 (red or grey keys) Exit

- Key D:
  - Print string “ do _ _ _ _”
  - Get 4 hex digits
  - Execute RS2 instruction (run using stack 2) Op code 0x05

- Key E:
    - Print string “Addr_ _ _ _”
    - Get 4 hex digits
    - Loop
        - Print 4 hex digit address
        - Print string “==”
        - Print 2 hex digit value at address
        - If Key==B decrement address
        - If Key==F increment address
        - If Key==C
            - Decimal Points On
            - Get 2 hex values
            - Save value at address
        - If any other Key, Exit

The assembly listing for the monitor program is provided in the text file TRAINER8.LST.

The monitor has several useful subroutines for reading and displaying data. The two most important are OUTCH for displaying ASCII text and GETKEY for reading a key press.

OUTCH FB70

PRINTS ASCII CHARACTER IN ACCA ON THE DISPLAY. THE DISPLAY POINTER IS THEN MOVED ONE DIGIT TO THE RIGHT. A CARRIAGE RETRUN 0x0D MOVES THE POINTER TO THE LEFT MOST DIGIT. A LINE FEED 0x0A MOVES THE CHARACTERS UP ONE (CLEARING THE DISPLAY). THUS, IT WORKS LIKE A ONE LINE CRT TERMINAL.

OUT2H FBEA

PRINTS THE VALUE IN ACCA AS TWO HEX CHARACTERS. CALLS OUTCH USING THE SAME DIGIT POINTER.

RESET FC00

RESET STACK POINTERS, FILL STACK WITH RETURN TO 0000, BREAK ADDRESS TO FFFF
START TRAINER MONITOR

PRTSTR FC8A

INDEX REG = ADRESS OF STRING, UP TO 8 ASCII CHARACTERS, ZERO TERMINATES FOR LESS THAN 8, PRINTS ON DISPLAY LEFT TO RIGHT, USES X AND ACCA

GETKEY FCC3

WAITS FOR KEY PRESS, RETURNS 0x00 to 0x0F IN ACCA, IF 0 AND 1 PRESSED, RETURNS 0x10 AND STORES IN FLAG, USES ACCB

MKASCI FCDA

TAKES 0x00 to 0x0F IN ACCA, RETURNS ASCII CHARACTER IN ACCA

GET2H FCE4

GETS TWO HEX DIGITS, STORES RESULT IN TEMP+1, RETURNS IN ACCA, PRINTS VALUES ON THE TWO RIGHT-MOST 7-SEG DIPLAYS, USES ACCA, ACCB, TEMP+1, TEMP+2

GET4H FD05

GETS FOUR HEX DIGITS, STORES RESULT IN TEMP AND TEMP+1, RETURNS LOWER BYTE IN ACCA, PRINTS VALUES ON THE FOUR RIGHT-MOST 7-SEG DIPLAYS, USES ACCA, ACCB, TEMP, TEMP+1, TEMP+2

CHG2H FD28

PRINTS ASCII STRING POINTED TO BY X, DISPLAYS DATA IN TEMP+1, IF ‘C’ KEY IS PRESSED: DECIMAL POINTS ON DATA ARE LIT AND GET2H IS CALLED; ELSE EXIT, USES X, ACCA, ACCB, TEMP+1, TEMP+2

CHG2HNP FD2B

SAME AS CHG2H WITHOUT STRING PRINT

CHG4H FDF8

PRINTS ASCII STRING POINTED TO BY X, DISPLAYS DATA IN TEMP, TEMP+1, IF ‘C’ KEY IS PRESSED: DECIMAL POINTS ON DATA ARE LIT AND GET4H IS CALLED; ELSE EXIT, USES X, ACCA, ACCB, TEMP, TEMP+1, TEMP+2

PRTDAT FE90

PRINTS HEX DATA FROM TEMP+1 ON TWO LEFT MOST 7-SEG, LIGHTS DECIMAL POINTS, USES ACCA, TEMP+1

PRTADD FEAB

PRINTS FOUR HEX VALUES FROM ADDRESS STARTING AT THE RIGHT MOST 7-SEG, PRINTS ‘==’, LOADS DATA POINTED TO BY ADDRESS AND STORES IT AT TEMP+1, USES ACCA, X, ADDRESS, TEMP+1

The interrupt vector table starts at address 0x100. If you use interrupts, jump instructions to your code must be placed at the starting address for the vector. E.G. your NMI code is at 0x0200 then, starting at address 0x0106 use 7E,02,00 which is JMP $0200.

IRQ VECTOR **0x0103**
SWI VECTOR **0x0100**
NMI VECTOR **0x0106**
RESET VECTOR **0xFC00**

Example use of Auto Entry mode:

Key	Display	Enter
	rEAdy	
A	Addr_ _ _ _	0000
	0000==0.0.	86
	0001==0.0.	2C
	0002==0.0.	20
	0003==0.0.	FE
Red	0003==G.0.	
Red	rEAdy	

The program above loads 2C hex into ACCA then branches to address 0002 indefinitely. Use these keystrokes to single step, run, reset and interrupt the code execution:

Key	Display	Enter	Comment
	rEAdy		
3	PC==0200		
C	PC==0.2.0.0.	0000	C allows changing memory and some registers
	rEAdy		
1	ACCA==00	0	
	rEAdy		
8	0003==2.0.	0	Single step shows the address of the next op code
	rEAdy		
1	ACCA==2C	C	ACCA has loaded the immediate value 2C
	ACCA==2.C.	00	
D	do _ _ _ _	0000	The DO command runs the code starting at 0000
	uSEr	6800 reset	Press reset to exit
	rEAdy		
1	ACCA==00	0	Reset restores stack pointers, registers are not valid
9	br. _ _ _ _	0002	Use the break point insert to stop your program
D	do _ _ _ _	0000	
	rEAdy		
1	ACCA==2C	0	ACCA is correct using a break point
	rEAdy		
7	uSEr		Return removes the break point and continues
	rEAdy		
E	Addr_ _ _ _	0000	Examining memory shows the break point was
	0000==86	F	removed and op code restored
	0001==2C	F	
	0002==20	F	
	0003==FE	0	
	rEAdy		

APP9: Read digital input port at address 0xF011 and display the Binary value. The input byte is rotated left. If a carry occurs, a one is printed else a zero is printed. The index register X points to the digit on the display. You can use the Boot switch to toggle bit-0.

```
A 0000
A 0000                         ORG    $0000
A 0000 CEC100    START         LDX    #$C100       *point to left digit
A 0003 F6F011                  LDAB   $F011        *get port data
A 0006 59        LOOP          ROLB                *move MSB to carry
A 0007 2404                    BCC    ZERO
A 0009 8631                    LDAA   #'1          *carry set load 1
A 000B 2002                    BRA    PRNTIT
A 000D 8630      ZERO          LDAA   #'0          *carry clear load 0
A 000F A700      PRNTIT        STAA   0,X          *display character
A 0011 08                      INX                 *move one digit right
A 0012 8CC108                  CPX    #$C108       *repeat for all 8
A 0015 26EF                    BNE    LOOP
A 0017 20E7                    BRA    START        *keep doing it
A 0019
A 0019
A                              END
```

To enter the program, start the Auto entry mode and the beginning address of 0000. Note: some programs may have several parts at different addresses. The bold hex values are the data to enter. For the above program CE, C1, 00, F6, F0, 11, 59 ... 20, E7 is the data to enter. Once the data is entered, press the red key twice to exit. Press the ‘D’ Do key and enter the 0000 program start address to run it. Move the Trainer/UART switch to observe the bit-0 change. Leave the switch in the Trainer position. Pushing the 6800 Reset button will bring you back to the monitor.

APP2: Print a scrolling message. Wait for a key press then return to the system monitor.

```
103 A      0000                ORG   $0000
104 A 0000 860A       START     LDAA  #$0A
105 A 0002 BDFB70               JSR   OUTCH      *CLEAR DISPLAY
106 A 0005 860D                 LDAA  #$0D       *MOVE TO LEFT DIGIT
107 A 0007 CE0027               LDX   #STRBEG    *ADDRESS OF STRING
108 A 000A BDFC8A     LOOP      JSR   PRTSTR
109 A 000D DFE7                 STX   ADDR       *SAVE X
110 A 000F CE8000               LDX   #$8000
111 A 0012 09         LOOP2     DEX              *DELAY LOOP
112 A 0013 01                   NOP
113 A 0014 01                   NOP
114 A 0015 01                   NOP
115 A 0016 01                   NOP
116 A 0017 26F9                 BNE   LOOP2
117 A 0019 DEE7                 LDX   ADDR       *RESTORE X
118 A 001B 08                   INX              *MOVE ON CHARATER UP
119 A 001C DFE7                 STX   ADDR
120 A 001E 8C003A               CPX   #STREND    *TEST IF DONE
121 A 0021 26E7                 BNE   LOOP
122 A 0023 BDFCC3               JSR   GETKEY     *WAIT FOR A KEY PRESS
123 A 0026 15                   FCB   RS1        *RETURN TO MONITOR
124 A 0027
125 A 0027 4D45535341 STRBEG    FCC   'MESSAGE IN A BOTTLE'
           474520494E
           204120424F
           54544C45
126 A 003A 2020202020 STREND    FCC   '        '
           202020
127 A 0042
128 A                           END
```

Five Minute Chess Clock Using 60Hz and NMI:

```
A      0106                 ORG  NMIV
A 0106 7E0110               JMP  IRQS    *Jump to NMI service
A 0109
A      0110                 ORG  $0110   *Service NMI
A 0110 7D0006    IRQS       TST  FLG     *Wait for 1st key press
A 0113 270C                 BEQ  RET0
A 0115 2B05                 BMI  ITS2    *Check which player
A 0117 CE0000               LDX  #ADR1   *Get clock for player 1
A 011A 2006                 BRA  OVER
A 011C CE0003    ITS2       LDX  #ADR2   *Get clock for player 2
A 011F 2001                 BRA  OVER
A 0121 3B        RET0       RTI
A 0122 6A02      OVER       DEC  2,X     *Update 60Hz counter
A 0124 26FB                 BNE  RET0    *Not 1 second yet
A 0126 863C                 LDAA #60     *restore counter
A 0128 A702                 STAA 2,X
A 012A 8699                 LDAA #$99    *10's complement -1
A 012C AB01                 ADDA 1,X     *Add it to seconds
A 012E 19                   DAA          *Make result decimal
A 012F 8199                 CMPA #$99    *Check for 60 seconds
A 0131 2604                 BNE  NOMIN
A 0133 8659                 LDAA #$59    *Restore seconds
A 0135 6A00                 DEC  0,X     *Subtract one minute
A 0137 A701      NOMIN      STAA 1,X     *Save seconds
A 0139 A600                 LDAA 0,X     *Get minutes
A 013B 2A05                 BPL  PRTTIM  *Still time left if +
A 013D 4F                   CLRA         *Out of time zero clock
A 013E A700                 STAA 0,X
A 0140 A701                 STAA 1,X
A 0142 860D      PRTTIM     LDAA #$0D    *Move to leftmost digit
A 0144 BDFB70               JSR  OUTCH
A 0147 B60000               LDAA ADR1    *Get player 1 minutes
A 014A BDFCDA               JSR  MKASCI  *Convert to ASCII char
A 014D BDFB70               JSR  OUTCH   *Display it
A 0150 B60001               LDAA ADR1+1  *Get player 1 seconds
A 0153 BDFBEA               JSR  OUT2H   *Display them
A 0156 8620                 LDAA #$20    *Move over 2 digits
A 0158 BDFB70               JSR  OUTCH
A 015B BDFB70               JSR  OUTCH
A 015E B60003               LDAA ADR2    *Get player 2 minutes
A 0161 BDFCDA               JSR  MKASCI
A 0164 BDFB70               JSR  OUTCH   *Display it
A 0167 B60004               LDAA ADR2+1  *Get player 2 seconds
A 016A BDFBEA               JSR  OUT2H   *Display them
A 016D 865F                 LDAA #'_     *Make a clock flag
```

```
A 016F 7D0006           TST  FLG    *Show the flag for the
A 0172 2B04             BMI  PLR2   *Active player
A 0174 B7C103           STAA ASCI3
A 0177 3B               RTI
A 0178 B7C104    PLR2   STAA ASCI4
A 017B 3B               RTI
A 017C
A      0200             ORG  $0200
A 0200 CE0000    START  LDX  #$0000 *Load the base address
A 0203 6F01             CLR  1,X    *Clear the minutes
A 0205 6F04             CLR  4,X
A 0207 6F06             CLR  6,X
A 0209 863C             LDAA #$3C   *Set the 60Hz counters
A 020B A702             STAA 2,X    *to 60 decimal
A 020D A705             STAA 5,X
A 020F 8605             LDAA #$05   *Set the player minutes
A 0211 A700             STAA 0,X    *to 5
A 0213 A703             STAA 3,X
A 0215 860A             LDAA #$0A   *Clear the display
A 0217 BDFB70           JSR  OUTCH
A 021A BDFCC3    GKEY   JSR  GETKEY *Wait for a player key
A 021D 4D               TSTA        *Not started is 0x00
A 021E 2707             BEQ  ISF    *Key 0 starts player 2
A 0220 8601             LDAA #$01   *other Keys player 1
A 0222 B70006           STAA FLG    *0x01 = player 1
A 0225 20F3             BRA  GKEY
A 0227 8682      ISF    LDAA #$82   *0x82 = player 2
A 0229 B70006           STAA FLG
A 022C 20EC             BRA  GKEY
A 022E
A 022E
A 022E
A      0000      ADR1   EQU  $0000  *Player 1 clock m:ss:60
A      0003      ADR2   EQU  $0003  *Player 2 clock m:ss:60
A      0006      FLG    EQU  $0006  *Active player
A 022E
A 022E
A                       END
```

Do 0200 to start. Then, jumper a wire from NMI to the 60Hz clock output. Press the 0 or red key to start the opponent's clock. To restart, remove the jumper and press the 6800 Reset. Do 0200 will start the program and the jumper needs replacing.

App11 voltmeter

```
A      0000                ORG     $0000
P 0000 0001        SIGN    RMB     1
P 0001 0001        OVFLO   RMB     1
P 0002 0001        CARRY   RMB     1
P 0003 0001        HIBYT   RMB     1
P 0004 0001        LOBYT   RMB     1
P 0005 0001        RSLTH   RMB     1
P 0006 0001        RSLTL   RMB     1
P 0007 0002        A       RMB     2
A
A      0200                ORG     $0200
A 0200 860A        START   LDAA    #$0A
A 0202 BDFB70              JSR     OUTCH
A 0205 C601        LOOP    LDAB    #$01
A 0207 F7F020              STAB    $F020 *TRIGGER CONVERT
A 020A B6F020              LDAA    $F020
A 020D F6F021              LDAB    $F021
A 0210 D704                STAB    LOBYT *SAVE ADC DATA
A 0212 9703                STAA    HIBYT
A 0214 58                  ASLB          *MULTIPLY BY 4
A 0215 49                  ROLA
A 0216 58                  ASLB
A 0217 49                  ROLA
A 0218 9707                STAA    A     *SAVE IN A
A 021A D708                STAB    A+1
A 021C DB04                ADDB    LOBYT *SUM ADC+ADC*4
A 021E D704                STAB    LOBYT
A 0220 9903                ADCA    HIBYT *SAVE
A 0222 9703                STAA    HIBYT
A 0224 9607                LDAA    A     *GET ADC*4
A 0226 D608                LDAB    A+1
A 0228 58                  ASLB          *MULTIPLY BY 4
A 0229 49                  ROLA
A 022A 58                  ASLB
A 022B 49                  ROLA
A 022C DB04                ADDB    LOBYT *SUM ADC+ADC*4+ADC*8
A 022E D704                STAB    LOBYT
A 0230 9903                ADCA    HIBYT
A 0232 9703                STAA    HIBYT
A 0234 BD0240              JSR     FRACT *CONVERT TO DECIMAL
A 0237 CEF000              LDX     #$F000
A 023A 09          DLAY    DEX
A 023B 26FD                BNE     DLAY
A 023D 7E0205              JMP     LOOP
A 0240
A 0240 CE0000      FRACT   LDX     #0000
```

```
A 0243 DF05                 STX     RSLTH
A 0245 4F                   CLRA
A 0246 8D0A                 BSR     MULT10 *MULTIPLY BY 10
A 0248 8D08                 BSR     MULT10 *TO GET EACH DIGIT
A 024A 8D06                 BSR     MULT10
A 024C 8D04                 BSR     MULT10
A 024E BD0288               JSR     PRT4H  *PRINT THE DIGITS
A 0251 39                   RTS
A 0252
A 0252 4F         MULT10    CLRA
A 0253 780004               ASL     LOBYT  *MULTIPLY BY 2
A 0256 790003               ROL     HIBYT
A 0259 49                   ROLA
A 025A 9702                 STAA    CARRY
A 025C 9701                 STAA    OVFLO
A 025E D604                 LDAB    LOBYT
A 0260 9603                 LDAA    HIBYT
A 0262 58                   ASLB           *MULTIPLY BY 4 MORE
A 0263 49                   ROLA
A 0264 790001               ROL     OVFLO
A 0267 58                   ASLB
A 0268 49                   ROLA
A 0269 790001               ROL     OVFLO
A 026C DB04                 ADDB    LOBYT  *ADD *2 + *8
A 026E D704                 STAB    LOBYT
A 0270 9903                 ADCA    HIBYT
A 0272 9703                 STAA    HIBYT
A 0274 9601                 LDAA    OVFLO
A 0276 9902                 ADCA    CARRY
A 0278 C604                 LDAB    #4
A 027A 780006     SHFT4     ASL     RSLTL  *SHIFT DECIMAL
A 027D 790005               ROL     RSLTH  *RESULT BY 1 DIGIT
A 0280 5A                   DECB
A 0281 26F7                 BNE     SHFT4
A 0283 9A06                 ORAA    RSLTL  *ADD NEW DIGIT
A 0285 9706                 STAA    RSLTL
A 0287 39                   RTS
A 0288
A 0288 860D       PRT4H     LDAA    #$0D   *MOVE TO THE LEFT
A 028A BDFB70               JSR     OUTCH  *MOST DIGIT
A 028D 8620                 LDAA    #$20
A 028F BDFB70               JSR     OUTCH  *PRINT A SPACE
A 0292 9605                 LDAA    RSLTH  *GET THE HIGH BYTE
A 0294 44                   LSRA           *UPPER NIBBLE
A 0295 44                   LSRA
A 0296 44                   LSRA
A 0297 44                   LSRA
A 0298 8AB0                 ORAA    #$B0   *CONVERT TO # & DP
```

```
A 029A BDFB70          JSR     OUTCH *DISPLAY IT
A 029D 9605            LDAA    RSLTH *GET THE HIGH BYTE
A 029F 840F            ANDA    #$0F
A 02A1 8A30            ORAA    #$30  *CONVERT TO #
A 02A3 BDFB70          JSR     OUTCH *DISPLAY IT
A 02A6 9606            LDAA    RSLTL *GET THE LOW BYTE
A 02A8 BDFBEA          JSR     OUT2H *DISPLAY IT
A 02AB 39              RTS
A 02AC
A                      END
```

How it works:

ADC number of bits = 10
Full Scale = 11 1111 1111_2
Let Full Scale approximate the fraction 0.999… = 1.0
To scale to 3.3V multiply by the fraction 0.010101000111101_2 = 0.33
Then, convert to a decimal fraction and move the decimal point

First, simplify the 0.33 fraction to .0101 0100 0000 0000_2 = 0.328
Multiply by 0.01_2 is divide by 4 or shift right 2
Multiply by 0.0001_2 is divide by 16 or shift right 4
Multiply by 0.000001_2 is divide by 64 or shift right 6
Note: the ADC MSB is shifted by 6 to the right .0000 0011 1111 1111_2
Multiply by 0.33 is ADC + (ADC<<2) + (ADC<<4) = RSLT

To convert the fraction to decimal, multiply by 10 = 1010_2
The overflow is the digit. Repeat three more times to get four digits
Multiply by 10 is (RSLT<<1) + (RSLT<<3)

To start, Do 0200. Then connect a 0.0V to 3.3V source to the A1 input. The wiper of a 1K potentiometer connected between GND and 3.3V can be used for testing. Do not apply more than 3.3V or less than 0V to the input.

Using the serial monitor to load programs:

Loading and saving programs as S-record files can be performed using the serial monitor mode. Refer to https://github.com/simple-circuit/picobug picobug.pdf for information on how to use the serial monitor mode.

- To load a program:
  - Move the Trainer/UART switch to the UART position and press Reset 6800
  - Start a serial terminal program such as simpleCRT.exe
  - Open the serial port associated with your Pico processor
  - Type 'L' at the > prompt
  - Select and send the S-record file associated with your program.
  - Move the Trainer/UART switch to the Trainer position and press Reset 6800
  - Press the 'D' button and enter the start address for your program.

The following S-record files for the examples are supplied:

APP9.S19	Binary Display	Start Address = 0000
APP2.S19	Scrolling Message	Start Address = 0000
CHSCLK.S19	5-Min Chess Clock	Start Address = 0200
APP11.S19	Voltmeter	Start Address = 0200