PC-2 Assembly language

Articles by Bruce Elliott

from TRS-80 Microcomputer News

- *March 1983*
- *April 1983*
- *May 1983*
- *September 1983*
- *October 1983*
- *February 1984*

WWW.
PC-1500
.INFO

Magazines from :
http://www.colorcomputerarchive.com/coco/Documents/Magazines/TRS-80%20Microcomputer%20News/

PC-2 Assembly Language

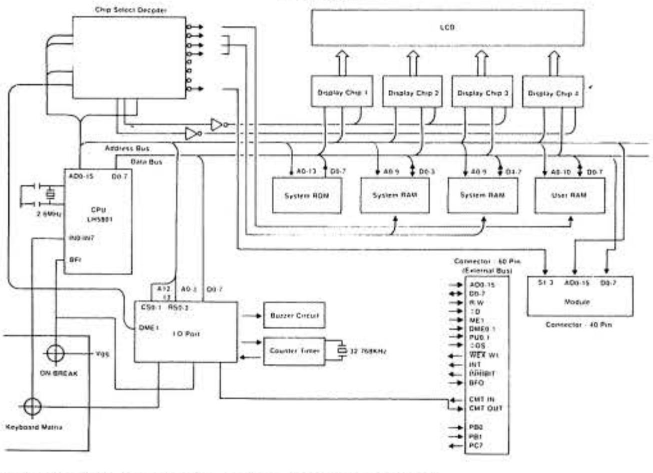
Article by Bruce Elliott

This is the first in a series of articles which will describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of the MPU and does not imply that all of these things can be done with a PC-2. Some known precautions when working with the PC-2 include:

- Po—This signal is not supplied to an external output pin on the PC-2.
- TI—The Timer Interrupt service routine is not available on the PC-2. If a Timer Interrupt occurs, an RTI is immediately executed.
- NMI—The Non-Maskable Interrupt is not available to the programmer on the PC-2.
- The MPU signals BRQ and BAK are not supplied to the external output pins.
- Though ME0 is available as an output from the MPU, DME0 (from one of the support chips) performs a similar function and should be used.

Please understand that the information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but can not reply to questions outside the scope of these articles. Further, published copies of *TRS-80 Microcomputer News* are the only source of this information, and we will not be maintaining back-issues.

PC-2 BLOCK DIAGRAM



OUTLINE OF THE 8-BIT CMOS MPU

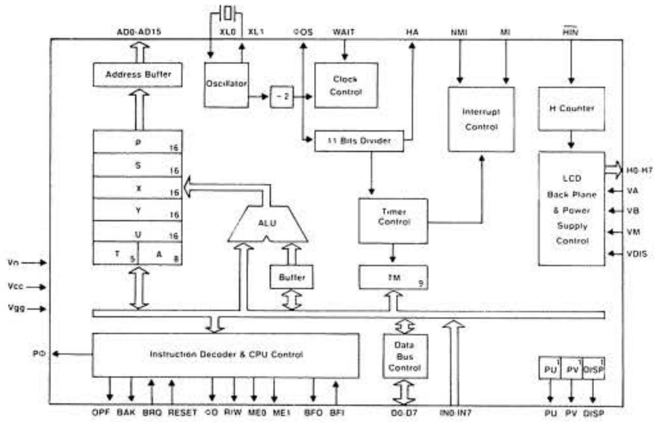
The 8-bit MPU chip (LH5801) uses CMOS static technology. This gives the MPU the low power dissipation inherent to CMOS technology. The MPU incorporates the LCD backplane signal generator, input port, external latch clock and the timer.

The MPU features:

- 16 bit address bus
- 8 bit data bus
- 8 bit input port
- DMA and multiprocessor capabilities
- Contains a WAIT function for memory access control
- LCD backplane control
- Clock frequency of 2.6 MHz.
  a. Internal machine cycle of 1.3MHz.
  b. Minimum instruction execution time of 1.3 microseconds.

In the PC-2, the MPU performs the following functions:

- Key input routine
- Acknowledges remaining program lines
- Interprets program execution statements
- Interprets cassette control statements
- Interprets printer control statements
- Interprets command statements
- Display processing routine
- Arithmetic routines
- Print routine
- Instructs I/O chip to perform serial communications, sound buzzer, and control counter/timer



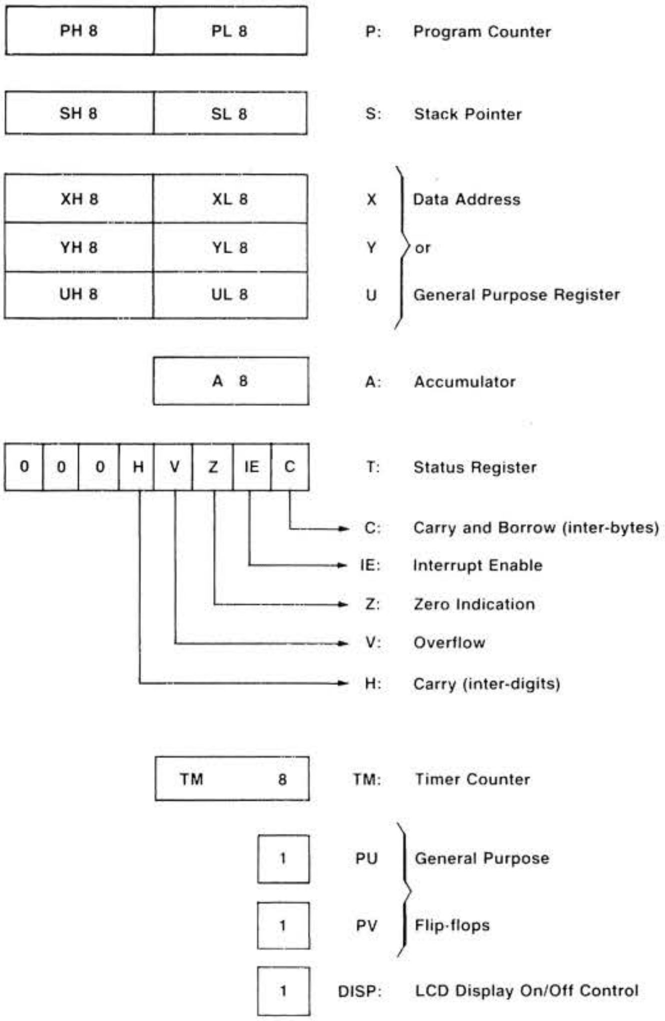
8 Bit CMOS MPU Block Diagram

MPU SIGNALS

ΦD—Output disable signal, when this signal is active, the data bus is in the output mode.

ΦOS—This clock signal is in phase with the internal basic clock and is supplied to the outside system. 2MHz of the clock frequency is supplied when a 4MHz crystal is being used between XL0 and XL1. Since PC-2 uses a 2.6MHz chip, the clock frequency is 1.3MHz.

AD0-AD15—Address bus. The address bus is tri-state and goes into the high impedance state when a Bus Request, BRQ, is issued.



MPU Internal Registers and Flip-flops

BAK—The BAK output is synchronized with the internal clock. When BAK goes high, the Address Bus, Data Bus, ME0, ME1, R/W, and ΦD all turn to the high impedance state Not used in PC-2.

BFO, BFI—BFO is an output of the BF flip-flop and BFI is an input to the BF flip-flop. The BF flip-flop is normally used for the memory backup system. In the PC-2, BFI is connected to the ⟨**BREAK**⟩ key, and goes "high" when the ⟨**BREAK**⟩ key is depressed. BFO, in the PC-2, is connected to the Chip Select Circuit and the Expansion Port.

BRQ—Bus Request. The MPU responds to the BRQ by turning BAK (Bus Acknowledge) high. Not used in PC-2. Tied to GND.

D0-D7—Bidirectional data bus through which data is written to or read from external memory.

DISP—A flip-flop which is used to control the on and off action of the L.C.D. Instructions are provided to set and reset this flip-flop.

GND—Ground

H0-H7—These are the LCD backplane signals.

HA—Output of the MPU internal driver. Divider output of 625 Hz in the PC-2. Used by the display chips.

HIN—LCD backplane signal and an input to the counter that generates H0-H7. This is connected to HA in the PC-2.

IN0-IN7—This is the input port which the MPU uses to bring 8-bit data into the internal accumulator. Internal pull-up resistance is present. In the PC-2, the input port is connected to the keyboard.

ME0, ME1—The Memory Enable signals used by the MPU to directly access a maximum of 128K bytes in external memory. In the PC-2, ME0 is connected to the chip select circuit and to the ME1 input of the I/O chip. In the PC-2, ME1 is connected to the ME0 input of the I/O chip and the expansion port.

MI—The Maskable Interrupt Input signal. The MPU will respond to this interrupt request when the Interrupt Enable flag (IE) is on. Interrupt processing will begin at the address indicated by FFF8 and FFF9. In the PC-2 this is connected to the INT output of the I/O Chip.

NMI—The Non-Maskable Interrupt Input. The MPU will respond unconditionally, and interrupt processing will begin at the address indicated by the contents of FFFC and FFFD. Not used in the PC-2, tied to GND.

OPF—Operation Code Fetch. Allows the MPU to fetch an operation (instruction) code. OPF appears when an instruction code is fetched, during address data and immediate data operations, and when the second byte of a two step instruction is being fetched. Not used in the PC-2.

PΦ—External latch clock. The contents of the accumulator is transferred on the data bus when this clock is in the high state, and can be used as an output port when an external latch IC is present. Not used in the PC-2.

PU, PV—These are MPU internal flip-flops. Set and reset instructions are provided for both PU and PV. In the PC-2, both PU and PV are connected to the expansion port. PU is one of the enable signals for the printer ROM.

R/W—Memory Read/Write Signal.

RESET—MPU reset input which causes the MPU to reset when a high signal is received. Program execution begins at the memory address pointed to by the contents of FFFE (low order) and FFFF (high order.) Execution begins at the indicated address when the RESET input changes from a high to a low state. On the PC-2 this is connected to the All Reset Switch.

VA—Power Supply to the LCD. High voltage for segment signals, 1.2—2.2 volts.

VB—Power Supply to the LCD. Low voltage for segment signals, .2—1.2 volts.

Vcc— +4.7 volts

VDIS—Power Supply to the LCD. +3.7 volts.

Vgg— +4.7 volts

VM—Power Supply to the LCD. An intermediate voltage used for the common and segment signals. .8—1.6 volts.

WAIT—When the MPU receives a high signal at the WAIT input, the MPU internal clock is halted to stop microprogram execution inside the MPU. WA is an internal flip-flop which accepts the WAIT input at the falling edge of the clock oOS and stops the MPU clock when it is in a high state. Connected to the WAIT output of the I/O chip in the PC-2. This informs the CPU when memory or an I/O device is not ready.

XL0, XL1—Crystal connection pins. PC-2 uses a 2.6MHz crystal which operates the MPU at a 1.3MHz clock frequency. XL0—Input, XL1—Output

MPU DESIGNATIONS

A : "A" represents the 8-bit register (accumulator) used for retention of arithmetical results or for data transfer with external (non-MPU) memory.

DISP: LCD display on/off control

P : "P" represents the 16-bit register (program counter) that indicates the next address that follows the currently executing instruction, and is automatically incremented by one when the next instruction is fetched. The maximum 64K bytes addressed by ME0 is addressable by P and constitutes the program area.

PH: High order 8 bits of the program counter

PL: Low order 8 bits of the program counter

PU: General purpose flip-flop

PV: General purpose flip-flop

R : represents any one of the X, Y, or U 16-bit registers. These registers can also be used as data pointers. When X, Y, or U are used as data pointers, it becomes possible to issue Memory Enable signals, ME0 and ME1, independently. A maximum of 128K bytes of memory area is available to X, Y, and U (a maximum of 64K bytes in the memory area accessed by ME0 and another 64K bytes in the memory area accessed by ME1.)

RH: represents any one of the high order XH, YH, or UH 8-bit registers.

RL: represents any one of the low order XL, YL, or UL 8-bit registers.

S : "S" represents the 16-bit register (stack pointer) that indicates the next available stack address for the push-down or pop-up stack in memory. The maximum 64K bytes addressed by ME0 is available as the stack area.

SH: High order 8 bits of the stack pointer

SL: Low order 8 bits of the stack pointer

T : "T" represents the 5-bit register (status register or flags) designed to hold status information such as: carry (C), borrow (H), zero (Z), overflow (V), and interrupt enable (IE). The flags (C, H, Z, V), other than the interrupt enable, can be tested by the conditional branch or conditional subroutine jump instructions.

TM: "TM" is the 9-bit polynomial counter (timer counter.)

U : 16-bit register

UH: High order 8 bits of register U

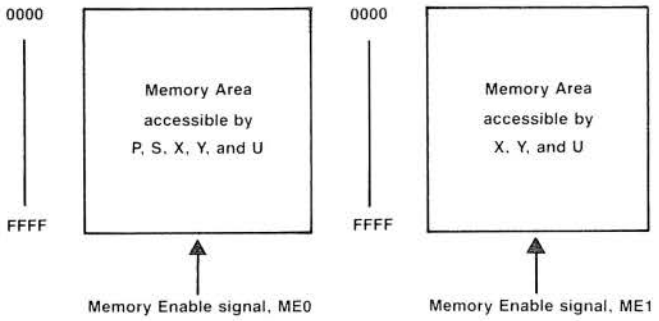
UL: Low order 8 bits of register U
X : 16-bit register
XH: High order 8 bits of register X.
XL: Low order 8 bits of register X.
Y : 16-bit register
YH: High order 8 bits of register Y
YL: Low order 8 bits of register Y

OPERATIONAL SYMBOLS

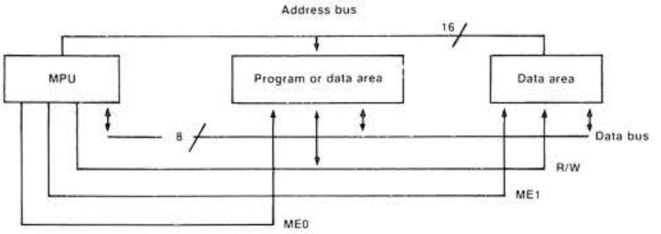
→ : Signal or data flow
← : Signal or data flow
• : Logical AND
v : Logical OR
⊕ : Exclusive OR
+ : Arithmetic addition
− : Arithmetic subtraction

MEMORY AND ADDRESS REPRESENTATION

Since the Memory Enable signals, ME0 and ME1, are output from the MPU, the PC-2 microprocessor can directly access any area within 128K bytes. ME0 takes care of one 64K byte memory area and ME1 another 64K byte memory area. However, ME0 is dedicated to program or data areas and ME1 to data area only.



Memory Area accessible by MPU

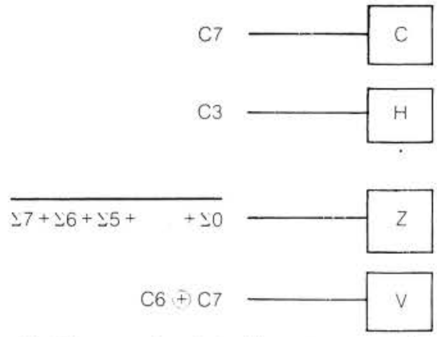


(R) : The contents of the ME0 accessible memory that can be specified by the register R.
#(R): The contents of the ME1 accessible memory that can be specified by the register R.
(ab): "a" is a number that represents the high order 8 bits of the address and "b" low order 8 bits of the address. Together, they indicate the contents of the memory that can be represented by the 16 combined bits of a and b (ME0 accessible).
#(ab): Same as the above, except that it can be accessed by ME1.
ab : used in defining the conditional jumps and subroutine calls to designate the two hex digits which comprise a single byte immediate value "i".

STATUS FLAGS

The status flags, C, V, H, Z, and IE are contained in the 5-bit status register. The contents of C, V, H, and Z may change upon completion of an arithmetic instruction.

Assume that the added results of each bit of the 9-bit full adder are as follows:
Σ7, Σ6, Σ5, Σ4, Σ3, Σ2, Σ1, Σ0, with carry of C7, C6, C5, C4, C3, C2, C1, C0.
The input conditions for each of flags shall be as described below:


C7
C
C3
H
$\overline{\Sigma 7 + \Sigma 6 + \Sigma 5 + \quad + \Sigma 0}$
Z
C6 ⊕ C7
V

(1) Carry flag C—The carry flag C is either set or reset depending on the presence of a carry in C7 (8th bit).
(2) Half carry flag H—The half carry flag H is either set or reset depending on the presence of a carry in C3.
(3) Zero flag Z—The zero flag Z is dependent on the arithmetic results, it will be set when the result is zero, otherwise, it will be reset.
(4) Overflow flag V—The overflow flag V is set when the arithmetic results of one byte is in overflow, provided that the 8th bit is used for a sign with rest of the 7 bits for used for numeric representation.

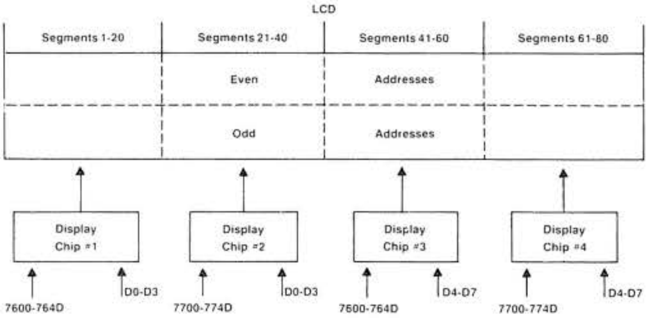
I/O PORT CHIP

Contains:

- two 8 bit bi-directional ports, labeled PA and PB. Each bit in these two ports can be programmed as either an input or an output. The CPU can access PA or PB as one location in memory. PA is used for the keyboard strobe and PB is used for cassette, counter/timer, and as an interrupt input.
- one 8 bit output port labeled PC. PC can be accessed as one location in memory and is used for counter/timer control and to sound the buzzer.
- Two interrupt request inputs, used with ⟨**BREAK**⟩ and IRQ inputs from the expansion port.
- one interrupt request output connected to the CPU.
- CPU WAIT control output. Outputs two memory enable signals, DME0 and DME1, which are used with memories that have slow access times.
- Controls serial communications. The two wait input lines, W0 and W1, are used in serial communications.

LCD DISPLAY CHIPS

Four display chips used for displaying information on the LCD, and as memory space for fixed memories E$ - Z$. Display chips 1 and 3 are used for the LCD display, indicators, and fixed memories E$ - O$. Display chips 2 and 4 are used for the LCD display and for fixed memories P$ - Z$.



OTHER PARTS OF THE PC-2 SYSTEM

- Chip Select Decoder Circuit
- 16K System ROM
- 1K System RAM (two 5514 RAM chips). This RAM is used for fixed memories A$ - D$, fixed memories A - Z, stack space, the 80 character input buffer, and is used by FOR-NEXT statements.
- 2K User RAM (one 6116 RAM chip). This RAM is used for fixed memories A27 or A$27 and above as well as being used for Reserve, Program and Variable memory.
- Buzzer circuit
- Counter/Timer circuit
- Module port
- Expansion port
- Keyboard

Memory Map:

0000 - 3FFF Module ROM - 16K

4000 - 47FF User RAM - 2K

```
4000 - 4007 Reserve Memory pointers
4008 - 4021 Menu 1
4022 - 403B Menu 2
403C - 4055 Menu 3
4056 - 40C3 Function Key Definitions
40C4        0 to mark end of function key definitions
40C5 - 47FF Program (Variable) Memory
```

4800 - 6FFF Module RAM

7000 - 75FF Duplicate of 7600 - 7BFF

7600 - 76FF Display Chip 1 & 3

```
7600 - 764D LCD Display - Sections 1 & 3
764E        Indicator
            Bit 0 - Busy
            Bit 1 - Shift
            Bit 2 - Japanese
            Bit 3 - Small
            Bit 4 - III
            Bit 5 - II
            Bit 6 - I
            Bit 7 - Def
764F        Indicator
            Bit 0 - De
            Bit 1 - G
            Bit 2 - Rad
            Bit 3 -
            Bit 4 - Reserve
            Bit 5 - Pro
            Bit 6 - Run
            Bit 7 -
7650 - 765F E$
7660 - 766F F$
7670 - 767F G$
7680 - 768F H$
7690 - 769F I$
76A0 - 76AF J$
76B0 - 76BF K$
76C0 - 76CF L$
76D0 - 76DF M$
76E0 - 76EF N$
76F0 - 76FF O$
```

7700 - 77FF Display Chips 2 & 4

```
7700 - 774D LCD Display - Sections 2 & 4
774E - 774F Not used
7750 - 775F P$
7760 - 776F Q$
7770 - 777F R$
7780 - 778F S$
7790 - 779F T$
77A0 - 77AF U$
77B0 - 77BF V$
77C0 - 77CF W$
77D0 - 77DF X$
77E0 - 77EF Y$
77F0 - 77FF Z$
```

7800 - 7BFF System Memory - 1K

```
7800 - 78BF System Memory - 192 Bytes
7863        RAM top - High order 8 bits
7864        RAM bottom - High order 8 bits
7865 - 7866 Beginning of BASIC program
7867 - 7868 End of BASIC program
7869 - 786A Head address of a BASIC program to perform editing based
            on keyboard entries
786B        Beep On/Beep off
7875        LCD Cursor Position
7879        Cassette parameter F/F
7880        LCD display parameter F/F
7890 - 7893 Used by RIGHT$, LEFT$, MID$
7894        String Buffer Pointer 7894 = 10H
7899 - 789A Start of variable storage area
789B        Error Code = ERR/2 + 1
78C0 - 78CF A$
78D0 - 78DF B$
78E0 - 78EF C$
78F0 - 78FF D$
7900 - 7907 A
7908 - 790F B
7910 - 7917 C
7918 - 791F D
7920 - 7927 E
7928 - 792F F
7930 - 7937 G
7938 - 793F H
7940 - 7947 I
7948 - 794F J
7950 - 7957 K
7958 - 795F L
7960 - 7967 M
7968 - 796F N
7970 - 7977 O
7978 - 797F P
7980 - 7987 Q
7988 - 798F R
7990 - 7997 S
7998 - 799F T
79A0 - 79A7 U
79A8 - 79AF V
79B0 - 79B7 W
79B8 - 79BF X
79C0 - 79C7 Y
79C8 - 79CF Z
```

79D0 - 7BFF System Memory - 560 Bytes

```
79E0 - 79E1 Printer X-axis position relative to origin
79E2 - 79E3 Printer Y-axis position relative to origin
79E4 - 79E5
79E6        Printer HCURSOR value
79E7 - 79E8
79E9        Printer pen up/down
79EA        Printer line type
79EB - 79EF
79F0        Printer Text/Graphic mode
79F1
79F2        Printer ROTATE value
79F3        Printer pen color
79F4        Printer CSIZE
7A00 - 7A07 Numeric Data Buffer or String pointer
7A10 - 7A17 Numeric Data Buffer or String pointer
7B10 - 7B4F String Buffer
7B60 - 7B67 Tape out Synchronization header
7B68 -      Tape out file mode
7B69 - 7B78 Tape out file name
7B79 - 7B84 Tape out header (available to user)
7B85 - 7B86 Tape out # bytes in BASIC file -1
7B87 - 7B88 Tape out end header
7B91 - 7BA0 Tape in file name
7BA1 - 7BAB Tape in user header
7BAC - 7BAD Tape in # bytes in BASIC file -1
7BAE - 7BAF Tape in end header
7BB0 - 7BFF 80 Character Display Buffer
```

7C00 - 7FFF Duplicate of 7800 - 7BFF

8000 - BFFF Expansion ROM - 16K

```
A519        Change printer pen color
A769        Printer motor off
A781        Send ASCII character to printer (no LF)
A8DD        Move pen
A9F1        Send line feed (LF) to printer
AA04        Send n line feeds to printer
AAD9        Pen Up/Down
ABCB        Switch printer from graphic to text mode
ABEF        Switch printer from text to graphic mode
BBD6        Write tape synchronization header
BBF5        Finalization of tape I/O control
BCE8        Read tape synchronization header/search for filename
BD3C        Read/Write file to tape
BDCC        Send a character to tape
BDF0        Read a character from tape
BF11        Turn tape drive on
BF43        Turn tape drive off
```

C000 - FFFF System Program ROM - 16K

```
D0D2        Magnitude Comparison for Numeric Values
D0F9        Magnitude Comparison for Character Strings
D2EA        Search for program line number
D461        Find address of variable
D925        String concatenation
D9B1        CHR$
D9CF        STR$
D9D7        VAL
D9DD        ASC if YL = 60H, LEN if YL = 64H
D9F3        RIGHT$, LEFT$, MID$
E243        Keyboard Scan - wait for character
E33F        Auto Power Off
E42C        Keyboard Scan - no wait
E8CA        Display contents of display buffer
ED00        Output n characters to LCD using current cursor location
ED3B        Output n characters to LCD beginning at cursor = 0
ED4D        Output one char to LCD and increment cursor position by one
ED57        Output one character to LCD
ED95        Convert two bytes of ASCII code (0-9,A-F) into one byte of hex data
EDEF        Output one graphic column to current cursor position
EFB6        X - Y → X
EFBA        X + Y → X
F00B        I/OP Flag 2
F01A        X * Y → X
F084        X / Y → X
F0E9        SQR X → X
F161        LN X → X
F165        LOG X → X
F1CB        EXP X → X
F1D4        10 ^ X → X
F391        COS X → X
F39E        TAN X → X
F3A2        SIN X → X
F492        ACS X → X
F496        ATN X → X
F49A        ASN X → X
F531        DEG X → X
F564        DMS X → X
F597        ABS X → X
F59D        SGN X → X
F5BE        INT X → X
F89C        Exponentiation (X ^ Y → X)
```

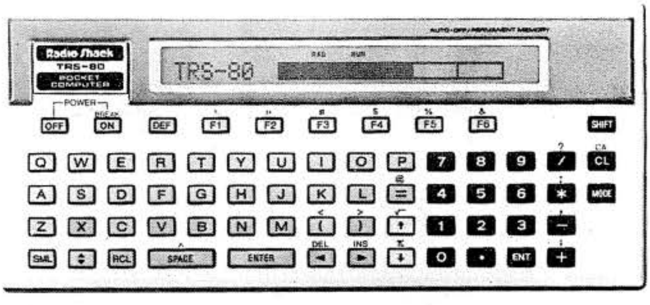
FF00 - FFF6 Vectors for jumps and calls

FFF8 - FFF9 Start Address for MI routine

FFFA - FFFB Start Address for the Internal Timer

FFFC - FFFD Start Address for the NMI routine

FFFE - FFFF Start address for the RESET routine



PC-2 Assembly Language–Part 2

Article by Bruce Elliott

This is the second in a series of articles which will describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of the MPU, and does not imply that all of these things can be done with a PC-2.

The information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but can not reply to questions outside the scope of these articles. Further, published copies of *TRS-80 Microcomputer News* are the only source of this information, and we will not be maintaining back issues.

Instruction Set

LOGICAL OPERATIONS

ADC—The contents of the internal register (RL or RH), or the contents of external memory [(R), #(R), (ab), or #(ab)] is added into the accumulator including the carry C. The result is stored in the accumulator. Flags C, H, Z, and V may change after the execution of this instruction.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ADC XL	A + XL + C → A	02	1	6
ADC YL	A + YL + C → A	12	1	6
ADC UL	A + UL + C → A	22	1	6
ADC XH	A + XH + C → A	82	1	6
ADC YH	A + YH + C → A	92	1	6
ADC UH	A + UH + C → A	A2	1	6
ADC (X)	A + (X) + C → A	03	1	7
ADC (Y)	A + (Y) + C → A	13	1	7
ADC (U)	A + (U) + C → A	23	1	7
ADC (ab)	A + (ab) + C → A	A3 a b	3	13
ADC #(X)	A + #(X) + C → A	FD 03	2	11
ADC #(Y)	A + #(Y) + C → A	FD 13	2	11
ADC #(U)	A + #(U) + C → A	FD 23	2	11
ADC #(ab)	A + #(ab) + C → A	FD A3 a b	4	17

ADI—Performs immediate addition to the accumulator or to external memory [(R), #(R), (ab), or #(ab)]. Changes may take place in C, H, Z, or V. The carry flag C will be included in the immediate addition to the accumulator.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ADI A,i	A + i + C → A	B3 i	2	7
ADI (X),i	(X) + i → (X)	4F i	2	13
ADI (Y),i	(Y) + i → (Y)	5F i	2	13
ADI (U),i	(U) + i → (U)	6F i	2	13
ADI (ab),i	(ab) + i → (ab)	EF a b i	4	19
ADI #(X),i	#(X) + i → #(X)	FD 4F i	3	17
ADI #(Y),i	#(Y) + i → #(Y)	FD 5F i	3	17
ADI #(U),i	#(U) + i → #(U)	FD 6F i	3	17
ADI #(ab),i	#(ab) + i → #(ab)	FD EF a b i	5	23

ADR—The content of the accumulator is added into the register R in 16 bits. Change may take place in C, H, Z, or V.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ADR X	XL + A → XL	FD CA	2	11
ADR Y	YL + A → YL	FD DA	2	11
ADR U	UL + A → UL	FD EA	2	11

Comment—RH + 1 → RH if C7 = 1 (no change in CVHZ)

AND—The content of the accumulator is logically ANDed with the content of external memory [(R), #(R), (ab), or #(ab)] and the result is stored in the accumulator. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
AND (X)	A ∧ (X) → A	09	1	7
AND (Y)	A ∧ (Y) → A	19	1	7
AND (U)	A ∧ (U) → A	29	1	7
AND (ab)	A ∧ (ab) → A	A9 a b	3	13
AND #(X)	A ∧ #(X) → A	FD 09	2	11
AND #(Y)	A ∧ #(Y) → A	FD 19	2	11
AND #(U)	A ∧ #(U) → A	FD 29	2	11
AND #(ab)	A ∧ #(ab) → A	FD A9 a b	4	17

Comment—∧ represents the AND operation

ANI—Logical AND of the accumulator and an immediate value, or of external memory [(R), #(R), (ab), or #(ab)] and an immediate value with the results stored in the accumulator or external memory as indicated. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ANI A,i	A ∧ i → A	B9 i	2	7
ANI (X),i	(X) ∧ i → (X)	49 i	2	13
ANI (Y),i	(Y) ∧ i → (Y)	59 i	2	13
ANI (U),i	(U) ∧ i → (U)	69 i	2	13
ANI (ab),i	(ab) ∧ i → (ab)	E9 a b i	4	19
ANI #(X),i	#(X) ∧ i → #(X)	FD 49 i	3	17
ANI #(Y),i	#(Y) ∧ i → #(Y)	FD 59 i	3	17
ANI #(U),i	#(U) ∧ i → #(U)	FD 69 i	3	17
ANI #(ab),i	#(ab) ∧ i → #(ab)	FD E9 a b i	5	23

DCA—The content of external memory [(R) or #(R)] including the carry C is added to the accumulator in the binary-coded-decimal (BCD) system and the result is stored in the accumulator. Change may take place in C, H, Z, or V.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
DCA (X)	A + (X) + C → A	8C	1	15
DCA (Y)	A + (Y) + C → A	9C	1	15
DCA (U)	A + (U) + C → A	AC	1	15
DCA #(X)	A + #(X) + C → A	FD 8C	2	19

DCA #(Y)	A + #(Y) + C → A	FD 9C	2	19
DCA #(U)	A + #(U) + C → A	FD AC	2	19

DCS—The content of the external memory [(R) or #(R)], including the carry C is subtracted from the content of the accumulator in the BCD system, and the result is stored in the accumulator. Change may take place in C, H, Z, or V.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
DCS (X)	A − (X) − $\overline{C}$ → A	0C	1	13
DCS (Y)	A − (Y) − $\overline{C}$ → A	1C	1	13
DCS (U)	A − (U) − $\overline{C}$ → A	2C	1	13
DCS #(X)	A − #(X) − $\overline{C}$ → A	FD 0C	2	17
DCS #(Y)	A − #(Y) − $\overline{C}$ → A	FD 1C	2	17
DCS #(U)	A − #(U) − $\overline{C}$ → A	FD 2C	2	17

DEC—Decrements the accumulator or the register (RL, RH, or R). Change may take place in C, V, H, and Z for the decrement of the accumulator, or the register, RL or RH. But no change takes place in flags when the 16-bit R is decremented.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
DEC A	A − 1 → A	DF	1	5
DEC XL	XL − 1 → XL	42	1	5
DEC YL	YL − 1 → YL	52	1	5
DEC UL	UL − 1 → UL	62	1	5
DEC XH	XH − 1 → XH	FD 42	2	9
DEC YH	YH − 1 → YH	FD 52	2	9
DEC UH	UH − 1 → UH	FD 62	2	9
DEC X	X − 1 → X	46	1	5
DEC Y	Y − 1 → Y	56	1	5
DEC U	U − 1 → U	66	1	5

EAI—The accumulator is EXCLUSIVE ORed with an immediate value and the result is stored in the accumulator. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
EAI i	A ⊕ i → A	BD i	2	7

Comment— ⊕ - represents the XOR operation

EOR—Logical EXCLUSIVE OR (XOR) of the accumulator with external memory [(R), #(R), (ab), or #(ab)] is performed and the result is stored in the accumulator. Change may take place in the Z flag.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
EOR (X)	A ⊕ (X) → A	0D	1	7
EOR (Y)	A ⊕ (Y) → A	1D	1	7
EOR (U)	A ⊕ (U) → A	2D	1	7
EOR (ab)	A ⊕ (ab) → A	AD a b	3	13
EOR #(X)	A ⊕ #(X) → A	FD 0D	2	11
EOR #(Y)	A ⊕ #(Y) → A	FD 1D	2	11
EOR #(U)	A ⊕ #(U) → A	FD 2D	2	11
EOR #(ab)	A ⊕ #(ab)→A	FD AD a b	4	17

INC—Increments the accumulator or the register (RL, RH, or R). Change may take place in C, V, H, and Z for an increment of the accumulator, or the registers, RL or RH. But no change takes place in flags when the 16-bit register R is incremented.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
INC A	A + 1 → A	DD	1	5
INC XL	XL + 1 → XL	40	1	5
INC YL	YL + 1 → YL	50	1	5
INC UL	UL + 1 → UL	60	1	5
INC XH	XH + 1 → XH	FD 40	2	9
INC YH	YH + 1 → YH	FD 50	2	9
INC UH	UH + 1 → UH	FD 60	2	9
INC X	X + 1 → X	44	1	5
INC Y	Y + 1 → Y	54	1	5
INC U	U + 1 → U	64	1	5

ORA—The accumulator is logically ORed with external memory [(R), #(R), or (ab)] and the result is stored in the accumulator. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ORA (X)	A v (X) → A	0B	1	7
ORA (Y)	A v (Y) → A	1B	1	7
ORA (U)	A v (U) → A	2B	1	7
ORA (ab)	A v (ab) → A	AB a b	3	13
ORA #(X)	A v #(X) → A	FD 0B	2	11
ORA #(Y)	A v #(Y) → A	FD 1B	2	11
ORA #(U)	A v #(U) → A	FD 2B	2	11
ORA #(ab)	A v #(ab) → A	FD AB a b	4	17

Comment—v - represents the OR operation

ORI—Logical OR of the accumulator or external memory [(R), #(R), (ab), or #(ab)] with an immediate value. The result is stored in the accumulator or the external memory as indicated. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ORI A,i	A v i → A	BB i	2	7
ORI (X),i	(X) v i → (X)	4B i	2	13
ORI (Y),i	(Y) v i → (Y)	5B i	2	13
ORI (U),i	(U) v i → (U)	6B i	2	13
ORI (ab),i	(ab) v i → (ab)	EB a b i	4	19
ORI #(X),i	#(X) v i → #(X)	FD 4B i	3	17
ORI #(Y),i	#(Y) v i → #(Y)	FD 5B i	3	17
ORI #(U),i	#(U) v i → #(U)	FD 6B i	3	17
ORI #(ab),i	#(ab) v i → #(ab)	FD EB a b i	5	23

SBC—The content of the internal register [RL or RH] or external memory [(R), #(R),(ab), or #(ab)] including the carry C is subtracted from the accumulator and the result is stored in the accumulator. Change may take place in C, H, Z, or V.

This operation can be expressed in the following manner: The complement of the contents in the internal register, RL or RH, or external memory, (R), #(R), (ab), or #(ab) is first obtained. Then the complement is added into the accumulator including the carry C, and the result is stored in the accumulator. Change may take place in C, H, Z, or V.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SBC XL	A − XL − $\overline{C}$ → A	00	1	6
SBC YL	A − YL − $\overline{C}$ → A	10	1	6
SBC UL	A − UL − $\overline{C}$ → A	20	1	6
SBC XH	A − XH − $\overline{C}$ → A	80	1	6
SBC YH	A − YH − $\overline{C}$ → A	90	1	6
SBC UH	A − UH − $\overline{C}$ → A	A0	1	6
SBC (X)	A − (X) − $\overline{C}$ → A	01	1	7
SBC (Y)	A − (Y) − $\overline{C}$ → A	11	1	7
SBC (U)	A − (U) − $\overline{C}$ → A	21	1	7
SBC (ab)	A − (ab) − $\overline{C}$ → A	A1 a b	3	13
SBC #(X)	A − #(X) − $\overline{C}$ →A	FD 01	2	11
SBC #(Y)	A − #(Y) − $\overline{C}$ →A	FD 11	2	11
SBC #(U)	A − #(U) − $\overline{C}$ →A	FD 21	2	11
SBC #(ab)	A − #(ab) − $\overline{C}$ →A	FD A1 a b	4	17

SBI—The immediate value including the carry C is subtracted from the accumulator and the result is stored in the accumulator. Change may take place in C, H, Z, or V.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SBI A,i	A−i−$\overline{C}$ → A	B1 i	2	7

COMPARISONS, BIT TESTS

BII—The accumulator or external memory [(R), #(R), (ab), or #(ab)] is logically ANDed with an immediate value. The result of the test is in the Z flag. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BII A,i	A ^ i → Z	BF i	2	7
BII (X),i	(X) ^ i → Z	4D i	2	10
BII (Y),i	(Y) ^ i → Z	5D i	2	10
BII (U),i	(U) ^ i → Z	6D i	2	10
BII (ab),i	(ab) ^ i → Z	ED a b i	4	16
BII #(X),i	#(X) ^ i → Z	FD 4D i	3	14
BII #(Y),i	#(Y) ^ i → Z	FD 5D i	3	14
BII #(U),i	#(U) ^ i → Z	FD 6D i	3	14
BII #(ab),i	#(ab) ^ i → Z	FD ED a b i	5	20

Comment—^ - represents the AND operation

BIT—The accumulator is logically ANDed with external memory [(R), #(R), (ab), or #(ab)]. The result is in Z. Change may take place in the Z flag only.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BIT (X)	A ^ (X) → Z	0F	1	7
BIT (Y)	A ^ (Y) → Z	1F	1	7
BIT (U)	A ^ (U) → Z	2F	1	7
BIT (ab)	A ^ (ab) → Z	AF a b	3	13
BIT #(X)	A ^ #(X) → Z	FD 0F	2	11
BIT #(Y)	A ^ #(Y) → Z	FD 1F	2	11
BIT #(U)	A ^ #(U) → Z	FD 2F	2	11
BIT #(ab)	A ^ #(ab) → Z	FD AF a b	4	17

CPA—Compares the contents of the accumulator with that of the register, RL or RH, or external memory, (R), #(R), (ab), or #(ab). Change may take place in C, V, H, or Z.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
CPA XL	A−XL	06	1	6
CPA YL	A−YL	16	1	6
CPA UL	A−UL	26	1	6
CPA XH	A−XH	86	1	6
CPA YH	A−YH	96	1	6
CPA UH	A−UH	A6	1	6
CPA (X)	A−(X)	07	1	7
CPA (Y)	A−(Y)	17	1	7
CPA (U)	A−(U)	27	1	7
CPA (ab)	A−(ab)	A7 a b	3	13
CPA #(X)	A−#(X)	FD 07	2	11
CPA #(Y)	A−#(Y)	FD 17	2	11
CPA #(U)	A−#(U)	FD 27	2	11
CPA #(ab)	A−#(ab)	FD A7 a b	4	17

Comment—If	C	Z	V	H
A>op	1	0	*	*
A=op	1	1	*	*
A<op	0	0	*	*

V and H may change depending upon the arithmetic result of the compare.

CPI—The content of the accumulator or the register RL or RH, is compared with the immediate value, i. Change may take place in C, V, H or Z.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
CPI A,i	A−i	B7 i	2	7
CPI XL,i	XL−i	4E i	2	7
CPI YL,i	YL−i	5E i	2	7
CPI UL,i	UL−i	6E i	2	7
CPI XH,i	XH−i	4C i	2	7
CPI YH,i	YH−i	5C i	2	7
CPI UH,i	UH−i	6C i	2	7

Comment—If	C	Z	V	H
(op) > i	1	0	*	*
(op) = i	1	1	*	*
(op) < i	0	0	*	*

V and H may change depending upon the arithmetic result of the compare.

LOADS, STORES

ATT—The content of the accumulator is transferred to the T register. All flags are subject to change depending on the content of A.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ATT	A → T	FD EC	2	9

Comment—T - Status Register

LDA—The content of the register, RL or RH, or external memory [(R), #(R), (ab), or #(ab)] is loaded into the accumulator. When the content loaded is "00", it sets the flag Z. No change is made with respect to other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LDA XL	XL → A	04	1	5
LDA YL	YL → A	14	1	5
LDA UL	UL → A	24	1	5
LDA XH	XH → A	84	1	5
LDA YH	YH → A	94	1	5
LDA UH	UH → A	A4	1	5
LDA (X)	(X) → A	05	1	6
LDA (Y)	(Y) → A	15	1	6
LDA (U)	(U) → A	25	1	6
LDA (ab)	(ab) → A	A5 a b	3	12
LDA #(X)	#(X) → A	FD 05	2	10
LDA #(Y)	#(Y) → A	FD 15	2	10
LDA #(U)	#(U) → A	FD 25	2	10
LDA #(ab)	#(ab) → A	FD A5 a b	4	16

LDE—The content of the register R is decremented upon loading the content of the external memory (R) into the accumulator. Change may take place only in the Z flag.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LDE X	(X) → A, X − 1 → X	47	1	6
LDE Y	(Y) → A, Y − 1 → Y	57	1	6
LDE U	(U) → A, U − 1 → U	67	1	6

LDI—The immediate value is loaded into the accumulator, register (RL or RH), or the stack pointer S. Only the immediate value being placed in S may contain 2 bytes. When using LDI A,i the Z flag may change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LDI A,i	i → A	B5 i	2	6
LDI XL,i	i → XL	4A i	2	6
LDI YL,i	i → YL	5A i	2	6
LDI UL,i	i → UL	6A i	2	6
LDI XH,i	i → XH	48 i	2	6
LDI YH,i	i → YH	58 i	2	6
LDI UH,i	i → UH	68 i	2	6
LDI S,i,j	i → SH, j → SL	AA i j	3	12

LDX—The content of the register R, stack pointer S, or program counter P is loaded into the X register. No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LDX X	X → X	FD 08	2	11
LDX Y	Y → X	FD 18	2	11
LDX U	U → X	FD 28	2	11
LDX S	S → X	FD 48	2	11
LDX P	P → X	FD 58	2	11

LIN—Increments R upon loading the content of the external memory (R) into the accumulator. Change may take place only in the Z flag.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LIN X	(X) → A, X+1 → X	45	1	6
LIN Y	(Y) → A, Y+1 → Y	55	1	6
LIN U	(U) → A, U+1 → U	65	1	6

POP—The contents placed on the stack by PSH is returned to the accumulator, A or the register, R. POP increments S by one in the case of the accumulator, and increments S by two in the case of a register. The Z flag may change as a result of the POP.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
POP A	(S+1) → A, S+1 → S	FD 8A	2	12
POP X	(S+1) → XH, (S+2) → XL, S+2 → S	FD 0A	2	15
POP Y	(S+1) → YH, (S+2) → YL, S+2 → S	FD 1A	2	15
POP U	(S+1) → UH, (S+2) → UL, S+2 → S	FD 2A	2	15

PSH—The content of the accumulator A or register R is stacked into the memory location specified by S. PSH decrements S by one in the case of the accumulator, and decrements S by two in the case of the register R. No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
PSH A	A → (S), S−1 → S	FD C8	2	11
PSH X	XL → (S), XH → (S−1), S−2 →S	FD 88	2	14
PSH Y	YL → (S), YH → (S−1), S−2 →S	FD 98	2	14
PSH U	UL → (S), UH → (S−1), S−2 →S	FD A8	2	14

SDE—The register R is decremented after the content of the accumulator is stored in external memory (R). No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SDE X	A → (X), X−1 → X	43	1	6
SDE Y	A → (Y), Y−1 → Y	53	1	6
SDE U	A → (U), U−1 → U	63	1	6

SIN—The register R is incremented after content of the accumulator is stored in external memory (R). No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SIN X	A → (X), X+1 → X	41	1	6
SIN Y	A → (Y), Y+1 → Y	51	1	6
SIN U	A → (U), U+1 → U	61	1	6

STA—The content of the accumulator is stored into register, RL or RH, or into external memory [(R), #(R), (ab), #(ab)]. No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
STA XL	A → XL	0A	1	5
STA YL	A → YL	1A	1	5
STA UL	A → UL	2A	1	5
STA XH	A → XH	08	1	5
STA YH	A → YH	18	1	5
STA UH	A → UH	28	1	5
STA (X)	A → (X)	0E	1	6
STA (Y)	A → (Y)	1E	1	6
STA (U)	A → (U)	2E	1	6
STA (ab)	A → (ab)	AE a b	3	12
STA #(X)	A → #(X)	FD 0E	2	10
STA #(Y)	A → #(Y)	FD 1E	2	10
STA #(U)	A → #(U)	FD 2E	2	10
STA #(ab)	A → #(ab)	FD AE a b	4	16

STX—The content of the X register is stored into register R, stack pointer S, or program counter P. No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
STX X	X → X	FD 4A	2	11
STX Y	X → Y	FD 5A	2	11
STX U	X → U	FD 6A	2	11
STX S	X → S	FD 4E	2	11
STX P	X → P	FD 5E	2	11

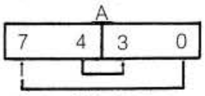
TTA—The content of the T register is transferred to the accumulator. The Z flag may change as a result of this operation.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
TTA	T → A	FD AA	2	9

Comment—T - Status Register

BLOCK TRANSFER, SEARCH

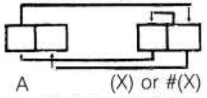
AEX—The high order 4 bit digit in the accumulator is exchanged with the lower order 4 bit digit.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
AEX		F1	1	6

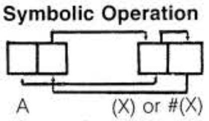
CIN—The content of the accumulator is compared with the content of the external memory (X), the flags C, V, H, and Z are set by the compare, then X register is incremented.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
CIN	A - (X), X+1 → X	F7	1	7

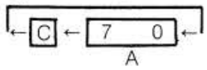
DRL—Performs digit-to-digit forward rotation between the accumulator and external memory, [(X) or #(X)]. No change takes place with respect to flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
DRL (X) DRL #(X)		D7 FD D7	1 2	12 16

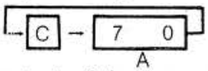
DRR—Performs digit-to-digit backward rotation between the accumulator and external memory [(X) or #(X)]. No change takes place with respect to flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
DRR (X) DRR #(X)		D3 FD D3	1 2	12 16

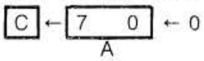
ROL—Forward rotation is made between the accumulator and the flag C. Flags C, V, H, and Z are subject to change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ROL		DB	1	6

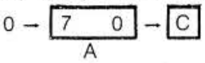
ROR—Backward rotation is made between the accumulator and the flag C. Flags C, V, H, and Z are subject to change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ROR		D1	1	9

SHL—The content of the accumulator is shifted to the left. Flags C, V, H, and Z are subject to change.

SHL		D9	1	6

SHR—The content of the accumulator is shifted to the right. Flags C, V, H, and Z are subject to change.

SHR		D5	1	9

TIN—The content of the external memory (X) is transferred into the external memory (Y), the X and Y registers are then incremented. No change takes place in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
TIN	(X) → (Y), X+1 → X, Y+1 → Y	F5	1	7

INPUT/OUTPUT

AM0—The contents of the accumulator is transferred timer. Since the timer is composed of a 9-bit polynomial counter, the content of the accumulator is set in the 1st through 8th bits of the counter and "0" is set in the 9th bit. It causes no change in flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
AM0	A → Timer (0-7) 0 → Timer (8)	FD CE	2	9

AM1—Same as AM0, except that "1" is set in the 9th bit. It causes no flag changes.

AM1	A → Timer (0-7) 1 → Timer (8)	FD DE	2	9

ATP—Sends the content of the accumulator to the external data bus. It causes no flag change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ATP	A → Data Bus	FD CC	2	9

CDV—Clears the internal divider. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
CDV	0 → Divider	FD 8E	2	8

HLT—The MPU is put into a halt state when this instruction is executed, except that the divider is still in operation. MPU operation can be resumed by means of the interrupt. No changes in flags occur.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
HLT		FD B1	2	9

ITA—The contents of the input IN is transferred to the accumulator. Change may take place in the Z flag, but there will be no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
ITA	IN → A	FD BA	2	9

NOP—No operation

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
NOP		38	1	5

OFF—Resets the BF flip-flop. It causes no change in the flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
OFF	0 → BF	FD 4C	2	8

RDP—Resets display flip-flop.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
RDP	0 → Display	FD C0	2	8

REC—Resets the carry flag C off. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
REC	0 → C	F9	1	4

RIE—Resets the Interrupt Enable (IE) flip-flop off. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
RIE	0 → IE	FD BE	2	8

RPU—Resets the general purpose flip-flop PU off. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
RPU	0 → PU	E3	1	4

RPV—Resets the general flip-flop PV off. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
RPV	0 → PV	B8	1	4

SDP—Sets display flip-flop.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SDP	1 → Display	FD C1	2	8

SEC—Sets the carry flag C on. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SEC	1 → C	FB	1	4

SIE—Sets the Interrupt Enable (IE) flip-flop on. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SIE	1 → IE	FD 81	2	8

SPU—Sets the general purpose flip-flop PU on. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SPU	1 → PU	E1	1	4

SPV—Sets the general purpose flip-flop PV on. It causes no change in other flags.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SPV	1 → PV	A8	1	4

PC-2 Assembly Language–Part 3

By Bruce Elliott

This is the third in a series of articles which will describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map, and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of MPU, and does not imply that all of these things can be done with a PC-2.

The information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but can not reply to questions outside the scope of these articles. Further, published copies of TRS-80 Microcomputer News are the only source of this information, and we will not be maintaining back issues. Parts One and Two of this series were published in the March and April issues, respectively.

JUMPS/BRANCHES

BCH—Causes a relative jump to a new program area that is determined by adding/subtracting the immediate value i to/from the program counter P.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BCH+i	P+i→P	8E i	2	8
BCH−i	P−i→P	9E i	2	9

BCR—Conditional relative jump instruction. The relative jump is made when "C=0". If "C=1", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BCR+i	if C=0, P+i→P	81 i	2	8-11
BCR−i	if C=0, P−i→P	91 i	2	8-11

Comment—If C=1, no jump

BCS—Conditional relative jump instruction. When the condition "C=1" is met, a relative jump is made to the program area that is found after adding/subtracting the immediate value i to/from the program counter P. If "C=0", control proceeds to the next instruction without making the relative jump. It causes no flag change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BCS+i	if C=1, P+i→P	83 i	2	8-11
BCS−i	if C=1, P−i→P	93 i	2	8-11

Comments—if C=0, no jump

BHR—A relative jump is made when "H=0". If "H=1", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BHR+i	if H=0, P+i→P	85 i	2	8-11
BHR−i	if H=0, P−i→P	95 i	2	8-11

Comment—if H=1, no jump

BHS—A relative jump is made when "H=1". If "H=0", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BHS+i	if H=1, P+i→P	87 i	2	8-11
BHS−i	if H=1, P−i→P	97 i	2	8-11

Comment—if H=0, no jump

BVR—A relative jump is made when "V=0". If "V=1", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BVR+i	if V=0, P+i→P	8D i	2	8-11
BVR−i	if V=0, P−i→P	9D i	2	8-11

Comment—if V=1, no jump

BVS—A relative jump is made when "V=1". If "V=0", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BVS+i	if V=1, P+i→P	8F i	2	8-11
BVS−i	if V=1, P−i→P	9F i	2	8-11

Comment—if V=0, no jump

BZR—A relative jump is made when "Z=0". If "Z=1", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BZR+i	if Z=0, P+i→P	89 i	2	8-11
BZR−i	if Z=0, P−i→P	99 i	2	8-11

Comment—if Z=1, no jump

BZS—A relative jump is made when "Z=1". If "Z=0", control proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
BZS+i	if Z=1, P+i→P	8B i	2	8-11
BZS−i	if Z=1, P−i→P	9B i	2	8-11

Comment—if Z=0, no jump

JMP—Causes a jump to a new program area implied by the immediate value in the second and third bytes. It causes no flag change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
JMP i,j	i→PH, j→PL	BA i j	3	12

LOP—This instruction causes a relative jump to a new program area if, when UL is reduced by 1, no borrow occurs (i.e., UL remains positive or zero). The new program area is determined by subtracting the immediate value i from P. If a borrow occurs when UL is reduced by 1, no jump takes place and execution proceeds to the next instruction. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LOP UL,i	UL − 1 → UL	88 i	2	8-11

Comment—if borrow = 1, no jump; if borrow = 0, P − i → P

CALLS

SJP—Makes a subroutine jump to the address specified by the immediate values i and j. At the same time, the address of the next instruction is stored in the stack. It causes no flag changes.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
SJP	PL → (S), PH → (S − 1), S − 2 → S, i → PH, j → PL	BE i j	3	19

VCR—Conditional vector subroutine jump. When "C = 0", the vector subroutine jump is performed. If "C = 1", the control proceeds to the next instruction. The Z flag is reset after the jump. VCR uses FF00 through FFF6 as its vector address table, and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VCR i	if C = 0, PH → (S − 1), PL → (S) (FFab) → PH, (FFab + 1) → PL S − 2 → S	C1 i	2	8-21

Comment—if C = 1, no jump, ab = Hex digits in i

VCS—Conditional vector subroutine jump. When "C = 1", it performs the vector subroutine jump. If "C = 0", the control proceeds to the next instruction. The Z flag is reset after the jump. VCS uses FF00 through FFF6 as its vector address table and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VCS i	if C = 1, PH → (S − 1), PL → (S) (FFab) → PH, (FFab + 1) → PL S − 2 → S	C3 i	2	8-21

Comment—if C = 0, no jump, ab = Hex digits in i

VEJ—Vector subroutine jump. VEJ is a one byte instruction which makes a subroutine jump based on a vectored address. The vector table is located in memory from FF00 to FFF6. The Z flag is reset after the vector jump is executed.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VEJ (ab)	PL → (S), S − 1 → S			
VEJ (C0)	PH → (S), S − 1 → S	C0	1	17
VEJ (C2)	(FFab) → PH	C2	1	17
VEJ (C4)	(FFab + 1) → PL	C4	1	17
VEJ (C6)		C6	1	17
VEJ (C8)		C8	1	17
VEJ (CA)		CA	1	17
VEJ (CC)		CC	1	17
VEJ (CE)		CE	1	17
VEJ (D0)		D0	1	17
VEJ (D2)		D2	1	17
VEJ (D4)		D4	1	17
VEJ (D6)		D6	1	17
VEJ (D8)		D8	1	17
VEJ (DA)		DA	1	17
VEJ (DC)		DC	1	17
VEJ (DE)		DE	1	17
VEJ (E0)		E0	1	17
VEJ (E2)		E2	1	17
VEJ (E4)		E4	1	17
VEJ (E6)		E6	1	17
VEJ (E8)		E8	1	17
VEJ (EA)		EA	1	17
VEJ (EC)		EC	1	17
VEJ (EE)		EE	1	17
VEJ (F0)		F0	1	17
VEJ (F2)		F2	1	17
VEJ (F4)		F4	1	17
VEJ (F6)		F6	1	17

Comment—Where, "ab" is the instruction code of VEJ.

VHR—Conditional vector subroutine jump. When "H = 0", the vector subroutine jump is performed. If "H = 1", the control proceeds to the next instruction. The Z flag is reset after the jump. VHR uses FF00 through FFF6 as its vector address table and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VHR i	if H = 0, PH → (S − 1), PL → (S) (FFab) → PH, (FFab + 1) → PL S − 2 → S	C5 i	2	8-21

Comment—if H = 1, no jump, ab = Hex digits in i

VHS—Conditional vector subroutine jump. When "H = 1", it performs the vector subroutine jump. If "H = 0", the control proceeds to the next instruction. The Z flag is reset after the jump. VHS uses FF00 through FFF6 as its vector address table and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VHS i	if H = 1, PH → (S − 1), PL → (S) (FFab) → PH, (FFab + 1) → PL S − 2 → S	C7 i	2	8-21

Comment—if H = 0, no jump; ab = Hex digits in i

VMJ—Vector subroutine jump. VMJ is the subroutine jump that branches to a vectored address, of which the high order byte is composed of "FF", and low order byte is composed of the immediate value i. Note that the Z flag is reset after the vector jump, when VMJ is executed. VMJ uses FF00 through FFF6 as its vector address table, and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VMJ i	PL → (S), S − 1 → S PH → (S), S − 1 → S (FFab) → PH (FFab + 1) → PL	CD i	2	20

Comments—ab = Hex digits in i

VVS—Conditional vector subroutine jump. When "V = 1", it performs the vector subroutine jump. If "V = 0", the control proceeds to the next instruction. The Z flag is reset after the jump. VVS uses FF00 through FFF6 as its vector address table and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VVS i	if V=1, PH→(S−1), PL→(S) (FFab)→PH, (FFab+1)→PL S−2→S	CF i	2	8-21

Comment—if V=0, no jump; ab = Hex digits in i

VZR—Conditional vector subroutine jump. When "Z=0", the vector subroutine jump is performed. If "Z=1", the control proceeds to the next instruction. The Z flag is reset after the jump. VZR uses FF00 through FFF6 as its vector address table and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VZR i	if Z=0, PH→(S−1), PL→(S) (FFab)→PH, (FFab+1)→PL S−2→S	C9 i	2	8-21

Comment—if Z=1, no jump; ab = Hex digits in i

VZS—Conditional vector subroutine jump. When "Z=1", it performs the vector subroutine jump. If "Z=0", the control proceeds to the next instruction. The Z flag is reset after the jump. VZS uses FF00 through FFF6 as its vector address table, and the values 00 through F6 are valid for the immediate value.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
VZS i	if Z=1, PH→(S−1), PL→(S) (FFab)→PH, (FFab+1)→PL S−2→S	CB i	2	8-21

Comment—if Z=0, no jump; ab = Hex digits in i

RETURNS

RTI—Return instruction from the interrupt subroutine to the main routine. All flags are subject to change.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
RTI	(S+1)→PH, (S+2)→PL, (S+3)→T, S+3→S	8A	1	14

RTN—Return instruction from a subroutine to the calling routine. RTN causes no changes in the flags.

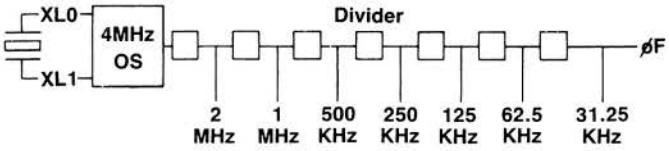
Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
RTN	(S+1)→PH, (S+2)→PL, S+2→S	9A	1	11

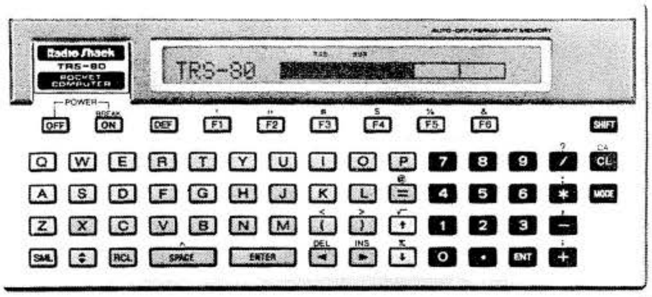
TIMER

The timer is composed of a 9-bit polynomial counter and the time duration can be set using the AM0 and AM1 instructions. This counter is in operation at all times, so it needs to be set to 000 (Hex) before being used. A timer interrupt request can be generated when the content of the counter is 1FF (Hex), if Interrupt Enable IE is on.

When a timer interrupt occurs, interrupt processing begins at the address specified in addresses FFFA and FFFB.

When a 4MHz crystal oscillator is used, the clock produces a oF of 31.25KHz with a cycle of 32 microseconds. In other words, the timer counter is incremented once every 32 microseconds.





National Computer Camp

National Computer Camp (NCC), which is believed to be the first computer camp in this country, is powering up for another series of computer camps for kids from ages nine to eighteen.

Campers at National Computer Camps learn to program the computer through a hands-on approach along with ample time on, and access to, the computer—two to three campers per computer. In the process, the campers come to understand the potential as well as the limitations involved in using computers.

1983 will be the sixth season of NCC, and for the first time, the camp will be held in three separate locations: Simsbury, CT; Atlanta, GA; and St. Louis, MO.

The sessions will be:

July 3—July 8
July 10—July 15
July 17—July 22
July 24—July 29
July 31—August 5

National Computer Camps
P.O. Box 585
Orange, CT 06477
1-203-795-9667

FLASH—Fourth Location Now Available

Dr. Zabinski has just informed us of a fourth National Computer Camp location for this summer. The fourth camp will be at Linfield College in McMinnville, Oregon. For further information, contact NCC at the address and phone number shown above.

PC-2 Assembly Language–Part 4

By Bruce Elliott

This is the fourth in a series of articles which describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map, and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of the MPU, and does not imply that all of these things can be done with a PC-2.

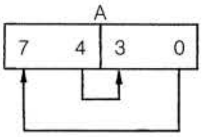
The information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but can not reply to questions outside the scope of these articles. Further, published copies of *TRS-80 Microcomputer News* are the only source of this information, and we will not be maintaining back issues. Parts One, Two and Three of this series were published in the March, April, and May 1983 issues, respectively.

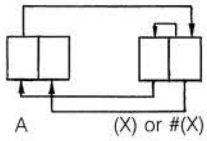
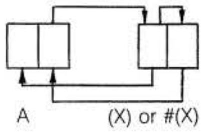
The first three articles described the MPU used in the PC-2, including information on the MPU's structure and its machine language. We also gave you details on the PC-2 memory map and the locations of ROM routines which are available. In this article we will present two lists which we hope will make finding a particular machine language instruction easier. We will also provide some information on how you might begin to use the information we have published.

ALPHABETIC OP-CODE LIST

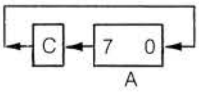
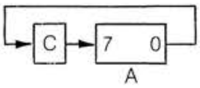
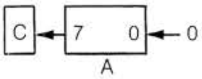
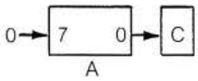
The following list presents the PC-2 machine language instructions alphabetically along with each code's symbolic operation and its hex op-code, and byte count.

Parts two and three of this series presented the same information arranged according to function and provided details on how the instructions work.

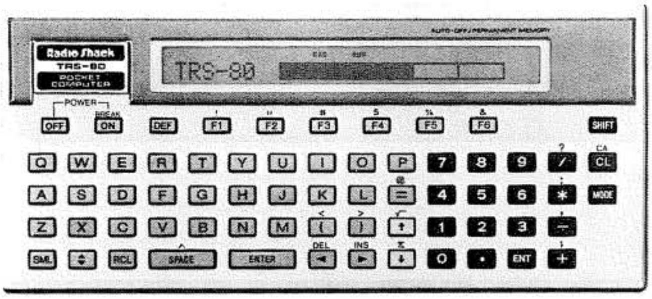
Mnemonic	Symbolic Operation	Hex Op-Code	Byte
ADC #(ab)	A + #(ab) + C → A	FD A3 a b	4
ADC #(U)	A+ #(U) + C → A	FD 23	2
ADC #(X)	A+ #(X) + C → A	FD 03	2
ADC #(Y)	A+ #(Y) + C → A	FD 13	2
ADC (ab)	A + (ab) + C → A	A3 a b	3
ADC (U)	A + (U) + C → A	23	1
ADC (X)	A + (X) + C → A	03	1
ADC (Y)	A + (Y) + C → A	13	1
ADC UH	A + UH + C → A	A2	1
ADC UL	A + UL + C → A	22	1
ADC XH	A + XH + C → A	82	1
ADC XL	A + XL + C → A	02	1
ADC YH	A + YH + C → A	92	1
ADC YL	A + YL + C → A	12	1
ADI #(ab),i	#(ab) + i → #(ab)	FD EF a b i	5
ADI #(U),i	#(U) + i → #(U)	FD 6F i	3
ADI #(X),i	#(X) + i → #(X)	FD 4F i	3
ADI #(Y),i	#(Y) + i → #(Y)	FD 5F i	3
ADI (ab),i	(ab) + i → (ab)	EF a b i	4
ADI (U),i	(U) + i → (U)	6F i	2
ADI (X),i	(X) + i → (X)	4F i	2
ADI (Y),i	(Y) + i → (Y)	5F i	2
ADI A,i	A + i + C → A	B3 i	2
ADR U	UL + A → UL	FD EA	2
ADR X	XL + A → XL	FD CA	2
ADR Y	YL + A → YL	FD DA	2
AEX	 A 7 4 3 0	F1	1
AM0	A → Timer (0-7) 0 → Timer (8)	FD CE	2
AM1	A → Timer (0-7) 1 → Timer (8)	FD DE	2
AND #(ab)	A ∧ #(ab) → A	FD A9 a b	4
AND #(U)	A ∧ #(U) → A	FD 29	2
AND #(X)	A ∧ #(X) → A	FD 09	2
AND #(Y)	A ∧ #(Y) → A	FD 19	2
AND (ab)	A ∧ (ab) → A	A9 a b	3
AND (U)	A ∧ (U) → A	29	1
AND (X)	A ∧ (X) → A	09	1
AND (Y)	A ∧ (Y) → A	19	1
ANI #(ab),i	#(ab) ∧ i → #(ab)	FD E9 a b i	5
ANI #(U),i	#(U) ∧ i → #(U)	FD 69 i	3
ANI #(X),i	#(X) ∧ i → #(X)	FD 49 i	3
ANI #(Y),i	#(Y) ∧ i → #(Y)	FD 59 i	3
ANI (ab),i	(ab) ∧ i → (ab)	E9 a b i	4
ANI (U),i	(U) ∧ i → (U)	69 i	2
ANI (X),i	(X) ∧ i → (X)	49 i	2
ANI (Y),i	(Y) ∧ i → (Y)	59 i	2
ANI A,i	A ∧ i → A	B9 i	2
ATP	A → Data Bus	FD CC	2
ATT	A → T	FD EC	2
BCH+i	P + i → P	8E i	2
BCH −i	P − i → P	9E i	2
BCR+ i	if C=0, P + i → P	81 i	2
BCR− i	if C=0, P − i → P	91 i	2
BCS+ i	if C=1, P + i → P	83 i	2
BCS− i	if C=1, P − i → P	93 i	2

Mnemonic	Symbolic Operation	Hex Op-Code	Byte
BHR+ i	if H=0, P + i → P	85 i	2
BHR− i	if H=0, P − i → P	95 i	2
BHS+ i	if H=1, P + i → P	87 i	2
BHS− i	if H=1, P − i → P	97 i	2
BII #(ab),i	#(ab) ∧ i → Z	FD ED a b i	5
BII #(U),i	#(U) ∧ i → Z	FD 6D i	3
BII #(X),i	#(X) ∧ i → Z	FD 4D i	3
BII #(Y),i	#(Y) ∧ i → Z	FD 5D i	3
BII (ab),i	(ab) ∧ i → Z	ED a b i	4
BII (U),i	(U) ∧ i → Z	6D i	2
BII (X),i	(X) ∧ i → Z	4D i	2
BII (Y),i	(Y) ∧ i → Z	5D i	2
BII A,i	A ∧ i → Z	BF i	2
BIT #(ab)	A ∧ #(ab) → Z	FD AF a b	4
BIT #(U)	A ∧ #(U) → Z	FD 2F	2
BIT #(X)	A ∧ #(X) → Z	FD 0F	2
BIT #(Y)	A ∧ #(Y) → Z	FD 1F	2
BIT (ab)	A ∧ (ab) → Z	AF a b	3
BIT (U)	A ∧ (U) → Z	2F	1
BIT (X)	A ∧ (X) → Z	0F	1
BIT (Y)	A ∧ (Y) → Z	1F	1
BVR+ i	if V=0, P + i → P	8D i	2
BVR− i	if V=0, P − i → P	9D i	2
BVS+ i	if V=1, P + i → P	8F i	2
BVS− i	if V=1, P − i → P	9F i	2
BZR+ i	if Z=0, P + i → P	89 i	2
BZR− i	if Z=0, P − i → P	99 i	2
BZS+ i	if Z=1, P + i → P	8B i	2
BZS− i	if Z=1, P − i → P	9B i	2
CDV	0 → Divider	FD 8E	2
CIN	A − (X), X+1 → X	F7	1
CPA #(ab)	A − #(ab)	FD A7 a b	4
CPA #(U)	A − #(U)	FD 27	2
CPA #(X)	A − #(X)	FD 07	2
CPA #(Y)	A − #(Y)	FD 17	2
CPA (ab)	A − (ab)	A7 a b	3
CPA (U)	A − (U)	27	1
CPA (X)	A − (X)	07	1
CPA (Y)	A − (Y)	17	1
CPA UH	A − UH	A6	1
CPA UL	A − UL	26	1
CPA XH	A − XH	86	1
CPA XL	A − XL	06	1
CPA YH	A − YH	96	1
CPA YL	A − YL	16	1
CPI A,i	A − i	B7 i	2
CPI UH,i	UH − i	6C i	2
CPI UL,i	UL − i	6E i	2
CPI XH,i	XH − i	4C i	2
CPI XL,i	XL − i	4E i	2
CPI YH,i	YH − i	5C i	2
CPI YL,i	YL − i	5E i	2
DCA #(U)	A + #(U) + C → A	FD AC	2
DCA #(X)	A + #(X) + C → A	FD 8C	2
DCA #(Y)	A + #(Y) + C → A	FD 9C	2
DCA (U)	A + (U) + C → A	AC	1
DCA (X)	A + (X) + C → A	8C	1
DCA (Y)	A + (Y) + C → A	9C	1
DCS #(U)	A − #(U) − $\overline{C}$ → A	FD 2C	2
DCS #(X)	A − #(X) − $\overline{C}$ → A	FD 0C	2
DCS #(Y)	A − #(Y) − $\overline{C}$ → A	FD 1C	2
DCS (U)	A − (U) − $\overline{C}$ → A	2C	1
DCS (X)	A − (X) − $\overline{C}$ → A	0C	1
DCS (Y)	A − (Y) − $\overline{C}$ → A	1C	1
DEC A	A − 1 → A	DF	1
DEC U	U − 1 → U	66	1
DEC UH	UH − 1 → UH	FD 62	2
DEC UL	UL − 1 → UL	62	1
DEC X	X − 1 → X	46	1
DEC XH	XH − 1 → XH	FD 42	2
DEC XL	XL − 1 → XL	42	1
DEC Y	Y − 1 → Y	56	1
DEC YH	YH − 1 → YH	FD 52	2
DEC YL	YL − 1 → YL	52	1
DRL #(X)		FD D7	2
DRL (X)		D7	1
DRR #(X)		FD D3	2
DRR (X)		D3	1
EAI i	A ⊕ i → A	BD i	2
EOR #(ab)	A ⊕ #(ab) → A	FD AD a b	4
EOR #(U)	A ⊕ #(U) → A	FD 2D	2
EOR #(X)	A ⊕ #(X) → A	FD 0D	2
EOR #(Y)	A ⊕ #(Y) → A	FD 1D	2
EOR (ab)	A ⊕ (ab) → A	AD a b	3
EOR (U)	A ⊕ (U) → A	2D	1
EOR (X)	A ⊕ (X) → A	0D	1
EOR (Y)	A ⊕ (Y) → A	1D	1
HLT		FD B1	2
INC A	A + 1 → A	DD	1
INC U	U + 1 → U	64	1
INC UH	UH + 1 → UH	FD 60	2
INC UL	UL + 1 → UL	60	1
INC X	X + 1 → X	44	1
INC XH	XH + 1 → XH	FD 40	2
INC XL	XL + 1 → XL	40	1
INC Y	Y + 1 → Y	54	1
INC YH	YH + 1 → YH	FD 50	2
INC YL	YL + 1 → YL	50	1
ITA	IN → A	FD BA	2
JMP i,j	i → PH, j → PL	BA i j	3
LDA #(ab)	#(ab) → A	FD A5 a b	4
LDA #(U)	#(U) → A	FD 25	2
LDA #(X)	#(X) → A	FD 05	2
LDA #(Y)	#(Y) → A	FD 15	2

Mnemonic	Symbolic Operation	Hex Op-Code	Byte
LDA (ab)	(ab) → A	A5 a b	3
LDA (U)	(U) → A	25	1
LDA (X)	(X) → A	05	1
LDA (Y)	(Y) → A	15	1
LDA UH	UH → A	A4	1
LDA UL	UL → A	24	1
LDA XH	XH → A	84	1
LDA XL	XL → A	04	1
LDA YH	YH → A	94	1
LDA YL	YL → A	14	1
LDE U	(U) → A, U − 1 → U	67	1
LDE X	(X) → A, X − 1 → X	47	1
LDE Y	(Y) → A, Y − 1 → Y	57	1
LDI A,i	i → A	B5 i	2
LDI S,i,j	i → SH, j → SL	AA i j	3
LDI UH,i	i → UH	68 i	2
LDI UL,i	i → UL	6A i	2
LDI XH,i	i → XH	48 i	2
LDI XL,i	i → XL	4A i	2
LDI YH,i	i → YH	58 i	2
LDI YL,i	i → YL	5A i	2
LDX P	P → X	FD 58	2
LDX S	S → X	FD 48	2
LDX U	U → X	FD 28	2
LDX X	X → X	FD 08	2
LDX Y	Y → X	FD 18	2
LIN U	(U) → A, U + 1 → U	65	1
LIN X	(X) → A, X + 1 → X	45	1
LIN Y	(Y) → A, Y + 1 → Y	55	1
LOP UL,i	UL − 1 → UL If borrow = 0, P − i → P	88 i	2
NOP		38	1
OFF	0 → BF	FD 4C	2
ORA #(ab)	A v #(ab) → A	FD AB a b	4
ORA #(U)	A v #(U) → A	FD 2B	2
ORA #(X)	A v #(X) → A	FD 0B	2
ORA #(Y)	A v #(Y) → A	FD 1B	2
ORA (ab)	A v (ab) → A	AB a b	3
ORA (U)	A v (U) → A	2B	1
ORA (X)	A v (X) → A	0B	1
ORA (Y)	A v (Y) → A	1B	1
ORI #(ab),i	#(ab) v i → #(ab)	FD EB a b i	5
ORI #(U),i	#(U) v i → #(U)	FD 6B i	3
ORI #(X),i	#(X) v i → #(X)	FD 4B i	3
ORI #(Y),i	#(Y) v i → #(Y)	FD 5B i	3
ORI (ab),i	(ab) v i → (ab)	EB a b i	4
ORI (U),i	(U) v i → (U)	6B i	2
ORI (X),i	(X) v i → (X)	4B i	2
ORI (Y),i	(Y) v i → (Y)	5B i	2
ORI A,i	A v i → A	BB i	2
POP A	(S + 1) → A, S + 1 → S	FD 8A	2
POP U	(S + 1) → UH, (S + 2) → UL,S + 2 → S	FD 2A	2
POP X	(S + 1) → XH, (S + 2) → XL,S + 2 → S	FD 0A	2
POP Y	(S + 1) → YH, (S + 2) → YL,S + 2 → S	FD 1A	2

Mnemonic	Symbolic Operation	Hex Op-Code	Byte
PSH A	A → (S), S − 1 → S	FD C8	2
PSH U	UL → (S), UH → (S − 1), S − 2 →S	FD A8	2
PSH X	XL → (S), XH → (S − 1), S − 2 →S	FD 88	2
PSH Y	YL → (S), YH → (S − 1), S − 2 →S	FD 98	2
RDP	0 → Display	FD C0	2
REC	0 → C	F9	1
RIE	0 → IE	FD BE	2
ROL		DB	1
ROR		D1	1
RPU	0 → PU	E3	1
RPV	0 → PV	B8	1
RTI	(S + 1) → PH, (S + 2) → PL, (S + 3) → T, S + 3 → S	8A	1
RTN	(S + 1) → PH, (S + 2) → PL, S + 2 → S	9A	1
SBC #(ab)	A − #(ab) − $\overline{C}$→ A	FD A1 a b	4
SBC #(U)	A − #(U) − $\overline{C}$→ A	FD 21	2
SBC #(X)	A − #(X) − $\overline{C}$→ A	FD 01	2
SBC #(Y)	A − #(Y) − $\overline{C}$→ A	FD 11	2
SBC (ab)	A − (ab) − $\overline{C}$→ A	A1 a b	3
SBC (U)	A − (U) − $\overline{C}$→ A	21	1
SBC (X)	A − (X) − $\overline{C}$→ A	01	1
SBC (Y)	A − (Y) − $\overline{C}$→ A	11	1
SBC UH	A − UH − $\overline{C}$→ A	A0	1
SBC UL	A − UL − $\overline{C}$→ A	20	1
SBC XH	A − XH − $\overline{C}$→ A	80	1
SBC XL	A − XL − $\overline{C}$→ A	00	1
SBC YH	A − YH − $\overline{C}$→ A	90	1
SBC YL	A − YL − $\overline{C}$→ A	10	1
SBI A, i	A − i − $\overline{C}$→ A	B1 i	2
SDE U	A → (U), U − 1 → U	63	1
SDE X	A → (X), X − 1 → X	43	1
SDE Y	A → (Y), Y − 1 → Y	53	1
SDP	1 → Display	FD C1	2
SEC	1 → C	FB	1
SHL		D9	1
SHR		D5	1

Mnemonic	Symbolic Operation	Hex Op-Code	Byte
SIE	1 → IE	FD 81	2
SIN U	A → (U), U+1 → U	61	1
SIN X	A → (X), X+1 → X	41	1
SIN Y	A → (Y), Y+1 → Y	51	1
SJP	PL → (S), PH → (S−1), S − 2 → S, i → PH, j → PL	BE i j	3
SPU	1 → PU	E1	1
SPV	1 → PV	A8	1
STA #(ab)	A → #(ab)	FD AE a b	4
STA #(U)	A → #(U)	FD 2E	2
STA #(X)	A → #(X)	FD 0E	2
STA #(Y)	A → #(Y)	FD 1E	2
STA (ab)	A → (ab)	AE a b	3
STA (U)	A → (U)	2E	1
STA (X)	A → (X)	0E	1
STA (Y)	A → (Y)	1E	1
STA UH	A → UH	28	1
STA UL	A → UL	2A	1
STA XH	A → XH	08	1
STA XL	A → XL	0A	1
STA YH	A → YH	18	1
STA YL	A → YL	1A	1
STX P	X → P	FD 5E	2
STX S	X → S	FD 4E	2
STX U	X → U	FD 6A	2
STX X	X → X	FD 4A	2
STX Y	X → Y	FD 5A	2
TIN	(X) → (Y), X+1 → X, Y+1 → Y	F5	1
TTA	T → A	FD AA	2
VCR i	if C=0, PH → (S−1), PL → (S) (FFab) → PH (FFab+1) → PL S − 2 →S	C1 i	2
VCS i	if C=1, PH → (S−1), PL → (S) (FFab) → PH (FFab+1) → PL S − 2 → S	C3 i	2
VEJ (C0)	PL → (S), S−1 → S PH → (S), S−1 → S (FFab) → PH (FFab+1) → PL	C0	1
VEJ (C2)		C2	1
VEJ (C4)		C4	1
VEJ (C6)		C6	1
VEJ (C8)		C8	1
VEJ (CA)		CA	1
VEJ (CC)		CC	1
VEJ (CE)		CE	1
VEJ (D0)		D0	1
VEJ (D2)		D2	1
VEJ (D4)		D4	1
VEJ (D6)		D6	1
VEJ (D8)		D8	1
VEJ (DA)		DA	1
VEJ (DC)		DC	1
VEJ (DE)		DE	1
VEJ (E0)		E0	1
VEJ (E2)		E2	1
VEJ (E4)		E4	1
VEJ (E6)		E6	1
VEJ (E8)		E8	1
VEJ (EA)		EA	1
VEJ (EC)		EC	1
VEJ (EE)		EE	1
VEJ (F0)		F0	1
VEJ (F2)		F2	1
VEJ (F4)		F4	1
VEJ (F6)		F6	1
VHR i	if H=0, PH → (S−1), PL → (S) (FFab) → PH (FFab+1) → PL S − 2 → S	C5 i	2
VHS i	if H=1, PH→ (S−1), PL → (S) (FFab) → PH (FFab+1) → PL S − 2 → S	C7 i	2
VMJ i	PL → (S), S−1 → S PH → (S), S−1 → S (FFab) → PH (FFab+1) → PL	CD i	2
VVS i	if V=1, PH → (S−1), PL → (S) (FFab) → PH (FFab+1) → PL S − 2 → S	CF i	2
VZR i	if Z=0, PH → (S−1), PL→(S) (FFab) →PH (FFab+1) →PL S − 2 →S	C9 i	2
VZS i	if Z=1, PH → (S−1), PL→(S) (FFab) →PH (FFab+1) →PL S − 2 →S	CB i	2



NUMERIC OP-CODE LIST

The following list presents the PC-2 machine language instructions numerically and includes the hex and decimal values for the op-codes. Numeric values which are missing from the list have no valid op-code that we are aware of.

Hex Value	Decimal Value	Opcode
00	00	SBC XL
01	01	SBC (X)
02	02	ADC XL
03	03	ADC (X)
04	04	LDA XL
05	05	LDA (X)
06	06	CPA XL
07	07	CPA (X)
08	08	STA XH
09	09	AND (X)
0A	10	STA XL
0B	11	ORA (X)
0C	12	DCS (X)
0D	13	EOR (X)
0E	14	STA (X)
0F	15	BIT (X)
10	16	SBC YL
11	17	SBC (Y)
12	18	ADC YL
13	19	ADC (Y)
14	20	LDA YL
15	21	LDA (Y)
16	22	CPA YL
17	23	CPA (Y)
18	24	STA YH
19	25	AND (Y)
1A	26	STA YL
1B	27	ORA (Y)
1C	28	DCS (Y)
1D	29	EOR (Y)
1E	30	STA (Y)
1F	31	BIT (Y)
20	32	SBC UL
21	33	SBC (U)
22	34	ADC UL
23	35	ADC (U)
24	36	LDA UL
25	37	LDA (U)
26	38	CPA UL
27	39	CPA (U)
28	40	STA UH
29	41	AND (U)
2A	42	STA UL
2B	43	ORA (U)
2C	44	DCS (U)
2D	45	EOR (U)
2E	46	STA (U)
2F	47	BIT (U)
38	56	NOP
40	64	INC XL
41	65	SIN X
42	66	DEC XL
43	67	SDE X
44	68	INC X
45	69	LIN X
46	70	DEC X
47	71	LDE X
48 i	72 i	LDI XH,i
49 i	73 i	ANI (X),i
4A i	74 i	LDI XL,i
4B i	75 i	ORI (X),i
4C i	76 i	CPI XH,i
4D i	77 i	BII (X),i
4E i	78 i	CPI XL,i
4F i	79 i	ADI (X),i
50	80	INC YL
51	81	SIN Y
52	82	DEC YL
53	83	SDE Y
54	84	INC Y
55	85	LIN Y
56	86	DEC Y
57	87	LDE Y
58 i	88 i	LDI YH,i
59 i	89 i	ANI (Y),i
5A i	90 i	LDI YL,i
5B i	91 i	ORI (Y),i
5C i	92 i	CPI YH,i
5D i	93 i	BII (Y),i
5E i	94 i	CPI YL,i
5F i	95 i	ADI (Y),i
60	96	INC UL
61	97	SIN U
62	98	DEC UL
63	99	SDE U
64	100	INC U
65	101	LIN U
66	102	DEC U
67	103	LDE U
68 i	104 i	LDI UH,i
69 i	105 i	ANI (U),i
6A i	106 i	LDI UL,i
6B i	107 i	ORI (U),i
6C i	108 i	CPI UH,i
6D i	109 i	BII (U),i
6E i	110 i	CPI UL,i
6F i	111 i	ADI (U),i
80	128	SBC XH
81 i	129 i	BCR+ i
82	130	ADC XH
83 i	131 i	BCS+ i
84	132	LDA XH
85 i	133 i	BHR+ i
86	134	CPA XH
87 i	135 i	BHS+ i
88 i	136 i	LOP UL,i
89 i	137 i	BZR+ i
8A	138	RTI
8B i	139 i	BZS+ i
8C	140	DCA (X)
8D i	141 i	BVR+ i
8E i	142 i	BCH+ i
8F i	143 i	BVS+ i
90	144	SBC YH
91 i	145 i	BCR−i
92	146	ADC YH
93 i	147 i	BCS−i
94	148	LDA YH
95 i	149 i	BHR−i
96	150	CPA YH
97 i	151 i	BHS−i
99 i	153 i	BZR−i
9A	154	RTN
9B i	155 i	BZS−i
9C	156	DCA (Y)
9D i	157 i	BVR−i
9E i	158 i	BCH−i
9F i	159 i	BVS−i
A0	160	SBC UH
A1 a b	161 a b	SBC (ab)
A2	162	ADC UH
A3 a b	163 a b	ADC (ab)
A4	164	LDA UH
A5 a b	165 a b	LDA (ab)
A6	166	CPA UH
A7 a b	167 a b	CPA (ab)
A8	168	SPV
A9 a b	169 a b	AND (ab)
AA i j	170 i j	LDI S,i,j
AB a b	171 a b	ORA (ab)
AC	172	DCA (U)
AD a b	173 a b	EOR (ab)
AE a b	174 a b	STA (ab)
AF a b	175 a b	BIT (ab)
B1 i	177 i	SBI i
B3 i	179 i	ADI A,i
B5 i	181 i	LDI A,i
B7 i	183 i	CPI A,i
B8	184	RPV
B9 i	185 i	ANI A,i
BA i j	186 i j	JMP i,j
BB i	187 i	ORI A,i
BD i	189 i	EAI i
BE i j	190 i j	SJP
BF i	191 i	BII A,i
C0	192	VEJ (C0)
C1 i	193 i	VCR i
C2	194	VEJ (C2)
C3 i	195 i	VCS i
C4	196	VEJ (C4)
C5 i	197 i	VHR i
C6	198	VEJ (C6)
C7 i	199 i	VHS i
C8	200	VEJ (C8)
C9 i	201 i	VZR i
CA	202	VEJ (CA)
CB i	203 i	VZS i
CC	204	VEJ (CC)
CD i	205 i	VMJ i
CE	206	VEJ (CE)
CF i	207 i	VVS i
D0	208	VEJ (D0)
D1	209	ROR
D2	210	VEJ (D2)
D3	211	DRR (X)
D4	212	VEJ (D4)
D5	213	SHR
D6	214	VEJ (D6)
D7	215	DRL (X)
D8	216	VEJ (D8)
D9	217	SHL
DA	218	VEJ (DA)
DB	219	ROL
DC	220	VEJ (DC)
DD	221	INC A
DE	222	VEJ (DE)
DF	223	DEC A
E0	224	VEJ (E0)
E1	225	SPU
E2	226	VEJ (E2)
E3	227	RPU
E4	228	VEJ (E4)
E6	230	VEJ (E6)
E8	232	VEJ (E8)
E9 a b i	233 a b i	ANI (ab),i
EA	234	VEJ (EA)
EB a b i	235 a b i	ORI (ab),i
EC	236	VEJ (EC)
ED a b i	237 a b i	BII (ab),i
EE	238	VEJ (EE)
EF a b i	239 a b i	ADI (ab),i
F0	240	VEJ (F0)
F1	241	AEX
F2	242	VEJ (F2)
F4	244	VEJ (F4)
F5	245	TIN
F6	246	VEJ (F6)
F7	247	CIN
F9	249	REC
FB	251	SEC
FD 01	253 01	SBC #(X)
FD 03	253 03	ADC #(X)
FD 05	253 05	LDA #(X)
FD 07	253 07	CPA #(X)
FD 08	253 08	LDX X
FD 09	253 09	AND #(X)
FD 0A	253 10	POP X
FD 0B	253 11	ORA #(X)
FD 0C	253 12	DCS #(X)
FD 0D	253 13	EOR #(X)
FD 0E	253 14	STA #(X)
FD 0F	253 15	BIT #(X)
FD 11	253 17	SBC #(Y)
FD 13	253 19	ADC #(Y)
FD 15	253 21	LDA #(Y)
FD 17	253 23	CPA #(Y)
FD 18	253 24	LDX Y
FD 19	253 25	AND #(Y)
FD 1A	253 26	POP Y
FD 1B	253 27	ORA #(Y)
FD 1C	253 28	DCS #(Y)
FD 1D	253 29	EOR #(Y)

Hex Value	Decimal Value	Opcode
FD 1E	253 30	STA #(Y)
FD 1F	253 31	BIT #(Y)
FD 21	253 33	SBC #(U)
FD 23	253 35	ADC #(U)
FD 25	253 37	LDA #(U)
FD 27	253 39	CPA #(U)
FD 28	253 40	LDX U
FD 29	253 41	AND #(U)
FD 2A	253 42	POP U
FD 2B	253 43	ORA #(U)
FD 2C	253 44	DCS #(U)
FD 2D	253 45	EOR #(U)
FD 2E	253 46	STA #(U)
FD 2F	253 47	BIT #(U)
FD 40	253 64	INC XH
FD 42	253 66	DEC XH
FD 48	253 72	LDX S
FD 49 i	253 73 i	ANI #(X),i
FD 4A	253 74	STX X
FD 4B i	253 75 i	ORI #(X),i
FD 4C	253 76	OFF
FD 4D i	253 77 i	BII #(X),i
FD 4E	253 78	STX S
FD 4F i	253 79 i	ADI #(X),i
FD 50	253 80	INC YH
FD 52	253 82	DEC YH
FD 58	253 88	LDX P
FD 59 i	253 89 i	ANI #(Y),i
FD 5A	253 90	STX Y
FD 5B i	253 91 i	ORI #(Y),i
FD 5D i	253 93 i	BII #(Y),i
FD 5E	253 94	STX P
FD 5F i	253 95 i	ADI #(Y),i
FD 60	253 96	INC UH
FD 62	253 98	DEC UH
FD 69 i	253 105 i	ANI #(U),i
FD 6A	253 106	STX U
FD 6B i	253 107 i	ORI #(U),i
FD 6D i	253 109 i	BII #(U),i
FD 6F i	253 111 i	ADI #(U),i
FD 81	253 129	SIE
FD 88	253 136	PSH X
FD 8A	253 138	POP A
FD 8C	253 140	DCA #(X)
FD 8E	253 142	CDV
FD 98	253 152	PSH Y
FD 9C	253 156	DCA #(Y)
FD A1 a b	253 161 a b	SBC #(ab)
FD A3 a b	253 163 a b	ADC #(ab)
FD A5 a b	253 165 a b	LDA #(ab)
FD A7 a b	253 167 a b	CPA #(ab)
FD A8	253 168	PSH U
FD A9 a b	253 169 a b	AND #(ab)
FD AA	253 170	TTA
FD AB a b	253 171 a b	ORA #(ab)
FD AC	253 172	DCA #(U)
FD AD a b	253 173 a b	EOR #(ab)
FD AE a b	253 174 a b	STA #(ab)
FD AF a b	253 175 a b	BIT #(ab)
FD B1	253 177	HLT
FD BA	253 186	ITA
FD BE	253 190	RIE
FD C0	253 192	RDP
FD C1	253 193	SDP
FD C8	253 200	PSH A
FD CA	253 202	ADR X
FD CC	253 204	ATP
FD CE	253 206	AM0
FD D3	253 211	DRR #(X)
FD D7	253 215	DRL #(X)
FD DA	253 218	ADR Y
FD DE	253 222	AM1
FD E9 a b i	253 233 a b i	ANI #(ab),i
FD EA	253 234	ADR U
FD EB a b i	253 235 a b i	ORI #(ab),i
FD EC	253 236	ATT
FD ED a b i	253 237 a b i	BII #(ab),i
FD EF a b i	253 239 a b i	ADI #(ab),i

HOW DO I USE ALL THIS?

The primary advantage of machine language over BASIC is speed. Your PC-2 has a very complete BASIC so there really isn't a lot of reason to program in machine language unless you are looking for a speed advantage. Let's look at a couple of programs which will demonstrate how fast machine language is compared to BASIC.

What we will do is write a BASIC program which will reverse each graphic point on the PC-2's LCD display. Any point which is black (on) will be turned white (off) and any point which is off will be turned on. We will then show you a similar program in machine language. This should let you compare the speeds of the two languages.

First the BASIC program:

```
2ØØ WAIT Ø
21Ø CLS
22Ø GCURSOR 3
      : REM SHIFT PRINTING RIGHT SLIGHTLY
23Ø PRINT "Microcomputer News"
24Ø FOR I=Ø TO 155
      : REM GRAPHIC COLUMNS
25Ø GCURSOR I
      : REM SET GRAPHIC CURSOR
26Ø A=POINT I
      : REM STORE COLUMN VALUE
27Ø B=Ø
      : REM NEW COLUMN - ALL POINTS OFF
28Ø FOR J=6 TO Ø STEP -1
      : REM EXAMINE DOTS
29Ø C=INT(A/2^J)
      : POINT ON OR OFF (1 OR Ø)
3ØØ IF C=Ø LET B=B+2^J
      : REM TURN ON IF OFF
31Ø A=A-C*2^J
      : REM GET READY FOR NEXT POINT
32Ø NEXT J
      : REM DO NEXT DOT
33Ø GPRINT B;
      : REM PRINT REVERSED COLUMN
34Ø NEXT I
      : REM DO NEXT COLUMN
35Ø GOTO 35Ø
```

To use the program, enter it into your PC-2. Change line 230 to print what ever you wish on the LCD. When you run the program, the LCD will be reversed one column at a time from left to right.

Lets look at a machine language program to do the same thing:

```
1Ø WAIT Ø
2Ø CLS
3Ø GCURSOR 3
4Ø PRINT "TRS-8Ø PC-2"
5Ø POKE 184Ø9, 72, 118, 74, Ø, 5, 189, 255, 65, 78,
     78, 153, 8
6Ø POKE 18421, 76, 119, 139, 6, 72, 119, 74, Ø, 158,
     18, 154
8Ø CALL 184Ø9
9Ø NEXT I
```

Looks kind of like a BASIC program doesn't it?

With the PC-2, you will normally use BASIC as a "vehicle" for getting the machine language routine into the computer and then executing it.

Lines 10-40 of this second program look a lot like the first four lines of our first program, and they do the same things—housekeeping and getting something on the LCD so the program can reverse it.

Lines 50 and 60 contain the actual machine code for our program. POKE is a PC-2 command which tells the computer to "poke" values into memory. The first value following POKE (18409 and 18421) tells the computer where in memory to start poking and the remaining values are the values to be POKEd into successive memory locations.

The CALL statement in line 80 tells the PC-2 to "jump" to the memory location specified (18409) and begin executing the program it finds there. If you have the computer jump to a memory location and the location does not begin a valid program, your PC-2 may freeze or perform in an unpredictable manner.

The GOTO 100 statement in line 100 "freezes" the LCD and lets you see the result of the reversal.

If you have entered and RUN the second program, you should have noticed that your message was printed on the display and then, almost instantly, the LCD was reversed. Quite a bit faster than BASIC's many seconds to reverse the screen.

This second program was copied from pages 62 and 63 of your PC-2 Owner's Manual. Add lines 70 and 90 from those pages to see multiple reversals. I numbered the first program in so that both programs can be in memory at the same time for comparisons of their speed.

DISASSEMBLY

You may be curious about how the machine code in lines 50 and 60 are able to reverse the display. To find out, we need to "disassemble" the machine code. The term "disassemble" means to take the hexadecimal (hex) or decimal values which represent a machine code program and to translate those values into more recognizable assembly language operation codes (op-codes.) Once you have the op-codes you will be better able to understand the logic that makes the program work.

Here is how I went about disassembling the machine code from lines 50 and 60:

1. Find the first value which represents an instruction to the computer. This is the value 72 in line 50. We know that this is a decimal value because a hex value (on the PC-2) is preceded by an '&'.
2. Locate the value 72 in the numeric op-code list. Remember that the decimal values are in the second column. The listing looks like this:

Hex Value	Decimal Value	Op-Code
48 i	72 i	LDI XH,i

The Op-code is LDI XH, i.

3. The 'i' in the op-code tells us that this instruction requires another value to be complete.
4. A quick check in the alphabetic listing gives this listing for LDI XH,i:

Mnemonic	Symbolic Operation	Hex Op-Code	Byte
LDI XH,i	i → XH	48 i	2

Mnemonic is just another word for op-code. The symbolic operation tells us that the value 'i' is stored into 'XH' (the high 8-bits of the 16-bit X register). We already knew the Hex Op-Code. The 'Byte' information tells us that this instruction requires two bytes (two values.)
Since this command requires a second value, we go back to line 50 in the BASIC program and get the next value (118).

5. I now have two values (72 118) which represent an instruction to the computer. The instruction translates as: Load the high portion of the X register with the decimal value 118.
6. I would now go back to line 50, get the next available value (74) and continue with steps 2-5 until I had used all of the available values in lines 50 and 60.

The result of the disassembly is:

Decimal Values	Hex Codes	Op-Code Translation
72 118	48 76	LDI XH, 76H
74 0	4A 00	LDI XL, 00H
5	05	LDA (X)
189 255	BD FF	EAI FFH
65	41	SIN X
78 78	4E 4E	CPI XL, 4EH
153 8	99 08	BZR−08H
76 119	4C 77	CPI XH, 77H
139 6	8B 06	BZS+ 06H
72 119	48 77	LDI XH, 77H
74 0	4A 00	LDI XL, 00H
158 18	9E 12	BCH−12H
154	9A	RTN

You should have noticed that I included the hex equivalents of the decimal values as I went along, and noticed that I used the hex values in my disassembled list (with an 'H' after those values for clarity.) The reason for doing this is that it will make comparisons with the PC-2 memory map a little easier. Also, most assembly language listings you read will use hex, so now is the time to start getting used to hex codes (if you aren't already.)

The simplest way of getting the hex codes is to get them from the numerical listing of op-codes that was presented earlier in this article.

Great, you say, but what do I do with all of this stuff? We will look at each line of the listing and see if we can make sense of it. To help the process, I am going to give each line a number (starting with 100 and incrementing by 10) to make referring to the lines a little easier.

Line	Decimal	Hex	Op-Code
100	72 118	48 76	LDI XH, 76H
110	74 0	4A 00	LDI XL, 00H
120	5	05	LDA (X)
130	189 255	BD FF	EAI FFH
140	65	41	SIN X
150	78 78	4E 4E	CPI XL, 4EH
160	153 8	99 08	BZR−08H
170	76 119	4C 77	CPI XH, 77H
180	139 6	8B 06	BZS+ 06H
190	72 119	48 77	LDI XH, 77H
200	74 0	4A 00	LDI XL, 00H
210	158 18	9E 12	BCH−12H
220	154	9A	RTN

Lines 100 and 110 load the X register with the hex value 7600.

Line 120 then tells the computer to load the A register with the value stored in the memory location that the X register is pointing to (7600). A quick glance at the PC-2 memory map (March MCN, pg. 26) shows us that the memory locations beginning at 7600H and continuing to 764DH are part of the PC-2's LCD display. What the computer has done is to look at the first byte of LCD memory (which corresponds to the first column of dots in the main LCD display area) and then place a copy of the value in that location into the MPU's A register.

Line 130 tells the computer to take the value in the A register and exclusive OR (XOR) it with the immediate value FFH. The bit pattern for FFH is: 1111 1111.

The exclusive OR operation compares each bit of the display value (stored in A) with a one bit from the FFH (a solid black, all on, column). If both bits are ones the computer stores a zero (0). If one bit is a one and the other is a zero, the computer stores a one. The net result is that after the EAI (XOR) operation, the A register contains a reversed copy of the original display byte.

Line 140 contains the one byte instruction SIN X. This single instruction tells the computer to take the value which is currently in the A register (our reversed column image) and store that value in the memory location pointed to by the X register.

If you remember (the computer does), this is currently the first byte of LCD RAM. Once the value from A has been stored, the computer will add one to the value currently in the X register.

Let's pause a moment and see what has happened. With only eight bytes of memory we have told the computer where the first column of LCD memory is (7600H), we have made a copy of that column, reversed the copy, stored the result back into the first column of LCD memory (7600H) and we have incremented our counter (the X register) so that it now points to the second column of the LCD. No wonder machine language is so fast!

Line 150 tells the computer to compare the lower 8-bits of the X register with the value 4EH. The computer will set its 'flags' based on whether the value in XL is 4EH or not.

Recall that the X register is pointing to LCD memory. A glance back to the PC-2 memory map shows us that if X contains 764EH, it is pointing just past the end (764DH) of LCD display sections 1 and 3.

Line 160 instructs the computer to examine the flags which were set by the CPI instruction in line 150. If the Z flag is zero ($Z=0$), meaning that XL did NOT contain the value 4EH, then the computer is instructed to count backwards eight bytes and continue executing the program from that point. If $Z=1$ the computer will continue to the instruction in line 170.

To count back eight bytes the way the computer will do it, we have to understand that the program counter (which is what will be reduced by eight) is already pointing to the first byte of the instruction in line 170. Count back eight from that point. You should have stopped on the 05H in line 120. The computer would continue executing instructions beginning with line 120.

What the programmer did was to create a loop. The purpose of the loop is to have the computer move one byte at a time through the memory of LCD chips 1 and 3 (7600H - 764DH) reversing each byte in memory as the computer comes to them.

Line 170 tells the computer that if the value in XL was 4EH (from the test and compare in lines 150 and 160), then test the value in XH (the upper 8-bits of X) to see if a 77H is present. The first time the computer executes line 170 the value in XH will be a 76H (put there in line 100.)

Line 180 tells the computer to move its program counter forward six bytes if the value in XH WAS a 77H. Remembering that the program counter is currently pointing to the first byte in line 190, adding six would move the pointer forward to the single byte in line 220.

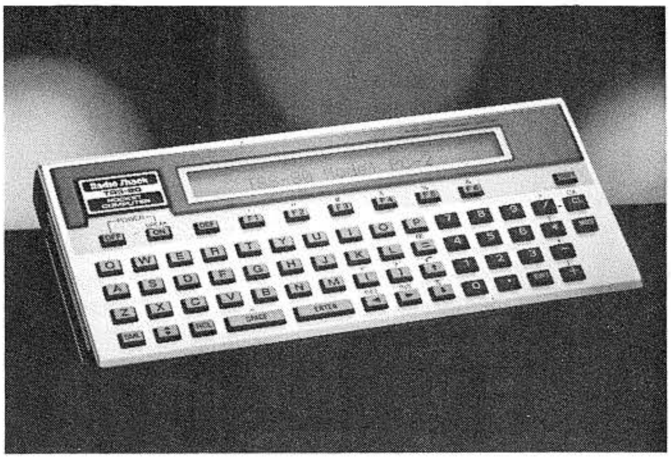
Line 190 is executed only if the value XH was not a 77H.

Line 200 will put a 00H into XL. A quick glance at the memory map shows us that 7700H if the first byte of LCD display memory for chips 2 and 4.

Line 210 tells the computer to subtract 12H (18 decimal) from its current program counter value. Since the program counter would be pointing at the 9AH in line 220, moving back 18 decimal would make the program counter point to line 120 again.

We already know that this will cause the computer to move through this new section of LCD memory (starting at 7700H this time) until the value in XL reaches 4EH. When XL reaches 4EH (this would be the second time), the computer would find 77H in XH (line 170) and the program counter would be moved forward to point at line 220 (line 180).

Line 220 is very important in any program which began by BASIC executing a CALL command. If you will look back to the BASIC program which loaded the machine code into memory, you will find the CALL command in line 80. The purpose of the RTN instruction in line 220 of our machine language program is to return control of the computer to BASIC and the program which contained the CALL command. If you forget to do this, you may have to push the ALL RESET button on the back of the PC-2 to regain control of the computer.



PC-2 Assembly Language–Part 5

By Bruce Elliott

This is the fifth in a series of articles which describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map, and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of the MPU, and does not imply that all of these things can be done with a PC-2.

The information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but cannot reply to questions outside the scope of these articles. Further, published copies of TRS-80 Microcomputer News are the only source of this information, and we will not be maintaining back issues. Parts One, Two, Three and Four of this series were published in the March, April, May, and September 1983 issues, respectively.

The first three articles described the MPU used in the PC-2, including information on the MPU's structure and its machine language. We also gave you details on the PC-2 memory map and the locations of ROM routines which are available. In the fourth article we presented two lists to make finding a particular machine language instruction easier. We also provided some information on how you might begin to use the information we have published. In this fifth article we want to present information on how to create your own machine language routines, and begin describing how to use the PC-2 ROM calls which are available.

CREATING YOUR OWN PROGRAMS

Last month we looked at an existing machine language program and described a procedure (disassembly) for determining how the program did what it was supposed to do. This month I want to define a program and then describe the procedure for creating a workable program that fits the definition. To make things simple, the program we are going to design will do only one thing—display on the LCD the key you press on the keyboard. I know that this program may sound silly. After all, doesn't the PC-2 automatically display the key you press? The answer is no, it doesn't. Try using the INKEY$ command. With INKEY$, if you want the character displayed you must display it yourself.

What we are really doing is designing a program which will accept characters from the PC-2 keyboard and display them on the LCD. This program should show you how to do three important things in assembly language: first, how to get information from the keyboard into the computer; second, how to take information that is in the computer and display it on the LCD; and third, how to use the PC-2's ROM subroutines.

In Part 1 of this series (March, 1983, pg. 26) we published a PC-2 memory map. It is in this section of PC-2 memory that we find ROM subroutines.

WHY DO ROM SUBROUTINES EXIST?

In general, any computer consists of similar basic parts. To function, a computer must have a processing unit, input and output functions, working memory to store temporary results, and some sort of control mechanism or program.

In the PC-2, the processing unit is the MPU which we have been describing in this series. The input function is handled primarily by the keyboard, and the output function is handled primarily by the LCD. The working memory is RAM (Random Access Memory), and the control mechanism is in the form of programs stored in ROM (Read Only Memory).

In order to make the PC-2 behave so that you can use it, the manufacturer wrote an operating system to control the various functions of the computer. Part of this operating system is instructions which control the keyboard, the LCD, and BASIC. This is where ROM subroutines come from. To function properly, the PC-2 has to have a routine which looks at the keyboard and stores any key which may be pressed. Likewise, there has to be a routine somewhere which takes a character and displays it on the LCD. The PC-2 memory map tells us where some of these routines are located, and we will use this information to create our machine language program.

IS THIS INFORMATION AVAILABLE ON OTHER COMPUTERS?

Radio Shack has received permission from the original manufacturer of the PC-2 to disclose the information which we are presenting in this series of articles. The information is fixed, and we do not expect it to change.

If you happen to own a different TRS-80 you may have tried to get similar information for that computer and you were told "I am sorry, but we cannot provide you with that information." Why? Well, there are two major reasons. The first and largest reason is that most computers are evolving products. As a computer evolves, the contents of its operating systems also change. If we give you information about where a particular routine is located in the first version of a program or operating system, you are going to expect that information to be true in the second version of that program or operating system also. With few exceptions, every change of a machine language program such as an operating system means a relocation of ALL of the contents of that program.

Because the contents of programs are subject to change with each revision, what Radio Shack typically does is to publish certain "published entry points." These published entry points won't normally change, even if the rest of the program does change. Other than the published entry points, Radio Shack, in general, will not provide you with other information about the contents of the program. Using only published entry points protects your software from becoming obsolete as soon as Radio Shack issues a new version of the program.

The second major reason for not providing the information is that Radio Shack often does not have permission from the copyright holder to release the information. As an example, Microsoft BASIC on any of our machines is owned by Microsoft. Since Microsoft owns the code, they have the right to tell us what we can and cannot publish.

BACK TO THE PC-2

The stated function of our machine language program is to accept keyboard entries and display the pressed key on the LCD.

A quick glance at the memory map for System Program ROM shows two keyboard scan routines and two routines which output single characters to the LCD.

E243H Keyboard Scan—Wait for Character
E42CH Keyboard Scan—No Wait
ED4DH Output one character to LCD and increment cursor position by one
ED57H Output one character to LCD

(Remember that the H after the address, as in E243H, indicates that the number is in Hexadecimal notation and not decimal.)

E243H

My information on the E243H Keyboard scan routine tells me that the PC-2 will wait for a key to be pressed. Once a key has been pressed, the key's code will be placed in the MPU Accumulator. If a key is not pressed within about seven minutes, the PC-2 will be turned off automatically. Once power-down has occurred, pressing the ON key will return the computer to the keyboard scan routine.

E42CH

The information on the E42CH routine states that if a key has been pressed, the key code will be in the accumulator. If a key has not been pressed the accumulator will contain 00H.

ED4DH

To output a character using ED4DH, the ASCII code of the character to be displayed is placed in the accumulator and the routine is executed. The character will be placed at the current cursor position, and then the cursor position will be updated.

The current cursor position is stored in memory location 7875H. According to our information, if the old cursor position (before the call to ED4DH) was less than 96H the new cursor position (stored in 7875H) will be the old position plus 6H. If the old cursor position was 96H or greater, the new position will be 00H.

ED57H

To display a character using the ROM routine at ED57H, place the ASCII value of the character to be displayed into the accumulator and execute the ED57H routine. The character will be displayed at the current cursor location and the cursor position will not be updated.

LET'S WRITE THE PROGRAM

I try to program conservatively when I use machine language. What I mean by this is that I try to disturb as few things as I can. So, the first part of my program will "save the MPU registers." What I mean by this is that I will save a copy of the various registers so I can restore the MPU when I am finished with my program. This is done by using the appropriate push (PSH) instructions to "push" the register values onto the stack.

```
FD C8       PSH A
FD 88       PSH X
FD 98       PSH Y
FD A8       PSH U
```

Now that I have saved a copy of the registers, I want to set the PC-2's cursor position to the left side of the LCD. This would make the cursor position (stored in 7578H) zero (0).

```
B5 ØØ       LDI A, ØØH
4A 75       LDI XL, 75H
48 78       LDI XH, 78H
ØE          STA (X)
```

Notice that I used three LoaD Immediate (LDI) instructions. The first LDI puts the cursor position (00H) into the MPU's Accumulator (A register.) The next two LDIs load the X register with the address which stores cursor position (7578H). The fourth instruction (STA) tells the MPU to put the value currently in the A register into the memory location which is currently in the X register.

Now that the cursor is where I want it, it is time to get a keystroke from the keyboard. Since the only thing I want to do is to get a keystroke, I choose to use the routine which waits for a key to be pressed before returning. A ROM routine is executed by using the Subroutine JumP (SJP) command.

```
BE E2 43    SJP E243H
```

We learned earlier that once a key is pressed, the PC-2 stores the ASCII value of the key in the A register. Both display routines I am considering require the ASCII value of the character I want displayed to be in the A register. Since the keyboard scan routine already put the ASCII value in the A register, all I need to do is use a subroutine jump to the proper display routine.

```
BE ED 4D    SJP ED4DH
```

I chose to display each character in cursor position 0, so I used the display routine at ED4DH.

The purpose of this program was to get a character from the keyboard and to display it on the LCD. My program has done that, so I restore the registers by POPping their values (in reverse order) off the stack.

```
FD 2A       POP U
FD 1A       POP Y
FD ØA       POP X
FD 8A       POP A
```

There is one final task which any machine language program which is called from BASIC (as this one will be) must perform and that is to return control of the PC-2 to BASIC. This is accomplished by executing a return command.

```
9A          RTN
```

Here is the completed machine language program along with various comments so I can remember what is happening.

```
FD C8       PSH A       'Save Registers
FD 88       PSH X
FD 98       PSH Y
FD A8       PSH U
B5 ØØ       LDI A, ØØH    'Cursor Position
4A 75       LDI XL, 75H   'Cursor Storage
48 78       LDI XH, 78H   ' Location
ØE          STA (X)       'Store Cursor
BE E2 43    SJP E243H     'Read Keyboard
BE ED 4D    SJP ED4DH     'Display Character
FD 2A       POP U         'Restore Registers
FD 1A       POP Y
FD ØA       POP X
FD 8A       POP A
9A          RTN           'Return to BASIC
```

TURN IT INTO A BASIC PROGRAM

Now that I have the machine code for my program, I need a way to get the program into the PC-2 and executed. A very straight forward way to do this in the PC-2 is to put the machine language program into a BASIC program shell like the following:

```
1Ø WAIT Ø
2Ø DATA &FD, &C8, &FD, &88
3Ø DATA &FD, &98, &FD, &A8
4Ø DATA &B5, &ØØ, &4A, &75
5Ø DATA &48, &78, &ØE
6Ø DATA &BE, &E2, &43
7Ø DATA &BE, &ED, &4D
8Ø DATA &FD, &2A, &FD, &1A
9Ø DATA &FD, &ØA, &FD, &8A
1ØØ DATA &9A
11Ø M=16999
12Ø FOR I=1 TO 3Ø
13Ø READ A
14Ø POKE M+I, A
15Ø NEXT I
16Ø M=M+1
17Ø PRINT "        READY"
18Ø CALL M
19Ø GOTO 18Ø
```

Line 10 simply sets the PC-2 PRINT command delay time to 0.

Lines 20-100 contain DATA statements into which I have placed the hexadecimal values for my machine language program. Notice the use of a leading '&' to indicate that the values are in Hex.

Line 110 contains the address (minus one) where I will begin storing the machine language program in memory.

Lines 120-150 POKE the machine language routine into PC-2 RAM memory. Line 160 updates the memory pointer from line 110 so that it contains the actual starting address of my routine (17000 decimal).

Line 170 tells me that the machine language program has been put into memory and will begin executing with the next instruction.

Line 180 tells BASIC to turn control of the PC-2 over to the machine language program which begins at location M (my memory pointer). The PC-2 will set the cursor position to zero, wait for a key to be pressed on the keyboard, display the proper character and return to BASIC.

Line 190 tells BASIC to go back to line 180 and execute the machine language program again.

THAT IS ALL THERE IS TO IT!

If you have followed this series of articles all the way through, you now have enough information about the PC-2 and how it operates to begin writing your own programs in machine language.

Next month we plan on giving you some additional information about the various ROM subroutines which are available to you in the PC-2.

A CLOSING GIFT

Operation codes (op-codes, mnemonics) are short names which programmers give to machine language commands to make them more readable, and more rememberable. We have given you several lists with op-codes and have provided some detail on what the commands do. At least one person has asked "How am I supposed to pronounce those funny looking things?"

Below is a listing of the various PC-2 op-codes and a recommended "name" or pronunciation for each.

Op-Code	Suggested Name	Op-Code	Suggested Name	Op-Code	Suggested Name
ADC	Add with Carry	PSH	Push	HLT	Halt
ADI	Add Immediate	POP	Pop	OFF	OFF
DCA	Decimal Add	ATT	Accumulator to T Register	JMP	Jump
ADR	Add Register	TTA	T Register to Accumulator	BCH	Branch
SBC	Subtract with Carry	TIN	Transfer and Increment	BCS	Branch Carry Set
SBI	Subtract Immediate	CIN	Compare and Increment	BCR	Branch Carry Reset
DCS	Decimal Subtract	ROL	Rotate Left	BHS	Branch Half Carry Set
AND	AND Accumulator	ROR	Rotate Right	BHR	Branch Half Carry Reset
ANI	AND Immediate	SHL	Shift Left	BZS	Branch Zero Set
ORA	OR Accumulator	SHR	Shift Right	BZR	Branch Zero Reset
ORI	OR Immediate	DRL	Decimal Rotate Left	BVS	Branch Overflow Set
EOR	Exclusive OR Accumulator	DRR	Decimal Rotate Right	BVR	Branch Overflow Reset
EAI	Exclusive OR Accumulator Immediate	AEX	Accumulator Nibble Exchange	LOP	Loop on Positive
		SEC	Set Carry	SJP	Subroutine Jump
INC	Increment	REC	Reset Carry	VEJ	Vector Jump
DEC	Decrement	CDV	Clear Divider	VMJ	Vector Unconditional
CPA	Compare Accumulator	ATP	Accumulator to Port	VCS	Vector Carry Set
CPI	Compare Immediate	ITA	Port Input to Accumulator	VCR	Vector Carry Reset
BIT	Bit	SPU	Set PU	VHS	Vector Half Carry Set
BII	Bit Immediate	RPU	Reset PU	VHR	Vector Half Carry Reset
LDA	Load Accumulator	RDP	Resets display flip-flop	VZS	Vector Zero Set
LDE	Load and Decrement	SDP	Sets display flip-flop	VZR	Vector Zero Reset
LIN	Load and Increment	SPV	Set PV	VVS	Vector Overflow Set
LDI	Load Immediate	RPV	Reset PV	VVR	Vector Overflow Reset
LDX	Load X	SIE	Set Interrupt Enable	RTN	Return from Subroutine
STA	Store Accumulator	RIE	Reset Interrupt Enable	RTI	Return from Interrupt
SDE	Store and Decrement	AM0	Accumulator to Timer, Bit 9 = 0	ME0	Memory Enable 0
SIN	Store and Increment	AM1	Accumulator to Timer, Bit 9 = 1	ME1	Memory Enable 1
STX	Store X	NOP	No Operation		

PC-2 Assembly Language–Part 6

By Bruce Elliott

This is the sixth in a series of articles which describe the MPU (microprocessor unit) used in the Radio Shack PC-2 pocket computer. It is our intention to include specific information about the 8-bit CMOS microprocessor, the machine code used by the microprocessor, as well as information about the PC-2 memory map, and certain ROM calls which are available. Please realize that much of what we are talking about refers to the overall capabilities of the MPU, and does not imply that all of these things can be done with a PC-2.

The information provided in these articles is the only information which is available. We will try to clarify any ambiguities which occur in the articles, but can not reply to questions outside the scope of these articles. Further, published copies of *TRS-80 Microcomputer News* are the only source of this information, and we will not be maintaining back issues.

In this article we want to present information on some of the PC-2 ROM calls which are available.

When you are going to use a ROM call, there are four items which you want to be concerned with:

1. Entry Address
2. Entry Conditions
3. Exit Conditions
4. Flags

The Entry Address is the address you use in the CALL statement from BASIC or a SJP call from machine language.

The Entry Conditions are conditions you must fulfill if the routine is to function properly. Normally, entry conditions specify where information must be and what information you must put in the MPU registers for the routine to function properly.

The Exit Conditions tell you where you will find the result of the operation (if there is a result) or provide you with other information about how things will change as a result of using a particular ROM call.

If a ROM call makes particular changes to any of the machine's flags, this information will be noted so you can properly interpret the results you get.

A CAUTION

I have not had time to test the information which is provided below on ROM calls. The information provided is as accurate as I could make it from the materials I am working with. Test any ROM call for proper operation BEFORE you use it in a program. Remember that the 'H' following a numeral indicates hexadecimal notation.

CURSOR INFORMATION

The PC-2 cursor pointer is located at 7875H. This location is used by the PC-2 to keep track of where the cursor should be. If you are working exclusively in machine language, updating 7875H is all that is needed for cursor location.

If you are working from BASIC, and wish to update the cursor location directly using POKEs or CALLs, you must also set bit 0 of location 7874H. Setting this bit from machine language can be accomplished by:

```
    ORI 7874H, 01H
```

This operation is done automatically when you use the CURSOR or GCURSOR BASIC commands.

If you execute a ROM call which resets the cursor pointer and are going to return to BASIC, you must set bit 0 of location 7874H as described above.

If you wish to reset the cursor from machine language, you can use the following code:

```
    ANI 7874H, 0FEH
    ANI 7875H, 00H
```

To increment the cursor pointer, use the following:

If you are displaying characters:

(7875H) = (7875H) + 06H

If you are displaying graphics:

(7875H) = (7875H) + 01H

Note: (7875H) must be between 00H and 9BH.

SYSTEM CALLS FOR THE LCD DISPLAY

Output one character to the LCD

1. System call address: ED57H
2. Entry Conditions:
   a. The ASCII character code for the character to be displayed must be in the ACC (Accumulator) before making the call.
   b. The location where the character will be placed is determined by the content of the cursor pointer.
3. Exit Conditions: The cursor pointer does not change.
4. Flags: Carry = 0 The cursor stays between 00H and 95H on call.
   = 1 The cursor stays in 96H on the call.

Output one character to the LCD and increment the cursor position by one character (6H).

1. System call address: ED4DH
2. Entry Conditions: The ASCII character code for the character to be displayed must be in the ACC (Accumulator) before making the call.
3. Exit Conditions: If the cursor position before the call was in the range 00H to 95H, then the new cursor position equals the old position plus 6H. If the cursor position before the call was 96H or larger, then the new cursor position is set equal to zero.
4. Flags:

Outputting n characters to the LCD.

1. System call address: ED00H
2. Entry Conditions:
   a. The 16 bit starting address for the string to be displayed is placed in the U register (0000H ⟨ = U ⟨ = FFFFH).

b. The length of the character string is placed in the Accumulator (01H ⟨ = ACC ⟨ = 1AH).

c. The cursor pointer indicates where on the LCD the computer is to begin displaying the string.

3. Exit Conditions: The cursor pointer is updated.
4. Flags: Carry = 0 The cursor position is set to the rightmost end of the displayed character string on the LCD.

= 1 The specified character string ended in the 26th LCD column, or the string was too long to be displayed within 26 columns. The cursor will be steady, indicating the last character displayed.

The number of characters specified in the accumulator is output from consecutive addresses beginning with the address specified in the U register. The characters will be placed on the LCD beginning with the position indicated by the cursor pointer. The cursor pointer can be set from machine language, or by using the BASIC CURSOR or GCURSOR commands. If the information to be displayed exceeds the 156th dot on the LCD, the excess information will not be displayed.

Outputting n characters to the LCD beginning from character position 1.

1. System call address: ED3BH
2. Entry Conditions:
   a. The 16 bit beginning address location of the string to be displayed is stored in the U register (0000H ⟨ = U ⟨ = FFFFH).
   b. An 8 bit number indicating the length of the character string is stored in XL (The lower half of the X register. 01H ⟨ = XL ⟨ = 1AH).
3. Exit Conditions:
4. Flags: Carry = 0 The character string has been displayed in 25 or fewer columns.

= 1 The character string reached or exceeded the 26th column.

Transferring 1 byte of data (1 dot column of graphic information) to the current cursor position.

1. System call address: EDEFH
2. Entry Conditions: The byte representing the graphic pattern to be displayed is placed in the accumulator.
3. Exit Conditions:
   a. The data is transferred to the current cursor position, which does not change.
   b. The contents of ACC and the X and U registers may change.
   c. The content of the Y register will not change.
4. Flags:

DATA CONVERSIONS

Converting two bytes of ASCII code (0 - 9, A - F only) into a one byte hexadecimal value.

1. System call address: ED95H
2. Entry Conditions: The X register should contain the address of the first of two consecutive bytes in memory which contain the ASCII characters.
3. Exit Conditions:
   a. The X register will be incremented by 2
   b. The U and Y registers will be unchanged
   c. The ACC will contain the converted hex value.
4. Flags:

DISPLAY THROUGH A BUFFER

Data can be placed into an 80-byte buffer (7BB0H - 7BFFH) and then displayed as needed by specifying the proper cursor address in the buffer.

1. System call address: E8CAH
2. Entry Conditions:
   a. Any character string which is placed in the buffer must have a 0DH code as the last character. This means that the longest allowable character string is 79 characters plus the 0DH end code.
   b. The Y register holds the cursor pointer for the buffer. The documentation does not specify what value goes into Y. Since Y is 16 bits long, I presume that you would use the actual memory address within the buffer.
   c. Address 7880H contains a parameter which determines how the contents of the buffer are to be displayed:

      If the binary content of 7880H is 0100 0000, then the character string stored in the buffer is output to the LCD using the content of the Y register as the cursor pointer.

      Note: If the number of characters in the buffer is 26 or less, then all of the characters are displayed on the LCD starting from the left side of the LCD. The cursor pointer (7875H) has no effect on this operation. If the number of characters in the buffer is greater than 26, the character in the address specified by the Y register and the PRECEDING 25 characters are displayed on the LCD starting at the left side of the LCD.

      If the binary content of 7880H is 0000 0000, then the cursor pointer in the Y register is ignored and he first 26 characters stored in the buffer are output to the LCD.

      If the binary content of 7880H is 0010 0000, then numeric data stored in memory addresses 7A00H - 7A07H are output to the LCD.

      Note: See below for a discussion of the 7A00H - 7A07H buffer.
3. Exit Conditions:
4. Flags:

The 7A00H - 7A07H Buffer

The PC-2 documentation describes three possible sets of data for the 7A00H buffer:

Decimal Values:

A decimal value may fall into the range 9.999999999 x 10E99 = x = 9.999999999 x 10E99.

7A00H contains the exponent (negative exponents are expressed as complements: 03H = x 10E3, 1FH = x 10E31, and FFH = x 10E-1)

7A01H contains the sign of the mantissa (00H = +, 80H = −)

7A02H - 7A06H contains the mantissa.

7A07H contains 00H.

Examples

```
7A00H     7A07H
00H 00H 00H 00H 00H 00H 00H 00H = 0.0
00H 00H 12H 34H 50H 00H 00H 00H = 1.2345
FEH 00H 98H 76H 54H 32H 12H 00H = 0.9876543212
08H 80H 54H 32H 00H 00H 00H 00H = -5.432 x 10
```

Integer Values:
An integer value may fall into the range -32768 ⟨ = × ⟨ = 32767.
7A00H—7A03H - Don't Care
7A04H—B2H
7A05H—7A06H Binary number in complements (e.g. 00H 00H = 0, FFH FBH = -5, 7FH FFH = 32767)
7A07H—Don't Care
Character Strings:
7A00H—7A03H—Don't Care
7A04H—D0H
7A05H—Upper two bytes of string address in memory
7A06H—Lower two bytes of string address in memory (string address can be in the range 0000H - FFFFH)
7A07H—Length of the string (range 01H - 50H)

Note: This last set of conditions (for strings) seems to imply that a string buffer can be anyplace in memory, rather than being restricted to 7BB0H - 7BFFH. Test this before relying on it.

CASSETTE I/O AND CONTROL

During tape I/O activities, the paper feed action of the printer is inhibited.

Turn Tape Drive On

1. System call address: BF11H
2. Entry Conditions: Memory address 7879H is used to specify certain conditions:
   Bit 7: 0 = CMT input port closes; select 0 for CMT input.
   1 = CMT input port opens; select 1 for CMT input.
   Bit 4: 0 = Remote 0
   1 = Remote 1
3. Exit Conditions:
4. Flags:

Turn Tape Drive Off

1. System call address: BF43H
2. Entry Conditions:
3. Exit Conditions: Remote drive 0 is turned off unconditionally. Remote drive 1 is turned off or on depending on bit 7 of an unspecified address (probably 7879H). If bit 7 is 0 the drive is OFF, and if bit 7 is a 1 then, the drive is ON. This bit can be set using the BASIC commands RMT ON and RMT OFF.
4. Flags:

Construct Tape Synchronization Header

The header, a 40-byte data set, consists of the synchronization header, a file name, file mode, and other data. This header is created inside the computer (addresses 7B60H - 7B87H) and output to tape.

1. System call address: BBD6H
2. Entry Conditions: The file mode (00 = Machine Object, 01 = Program, 02 = Reserve, 04 = Data) must be placed in the accumulator.
3. Exit Conditions:
   a. An 8 byte synchronization header will be in 7B60H - 7B67H
   b. File mode will be in 7B68H
   c. 00H characters will be placed in locations 7B69H - 7B87H.
4. Flags:

A program file name (16 or fewer characters) can be placed in memory locations 7B69H - 7B78H, if you wish. Address locations 7B79H - 7B87H may be used for your own purposes.

Output Tape Synchronization Header

1. System call address: BCE8H
2. Entry Conditions:
   a. Bit seven of address 7879H must be zero and bit four will be a zero for remote 0 and a one for remote 1.
   b. Whether the PC-2 will beep or not during cassette I/O is controlled by the BASIC commands BEEP ON and BEEP OFF, or by setting bit zero of 786BH.
3. Exit Conditions:
4. Flags:

Send a Character to Tape

1. System call address: BDCCH
2. Entry Conditions: Character to be output is placed in the Accumulator. The call to write the synchronization header must be used before outputting data using this system call.
3. Exit Conditions:
4. Flags:

Write a tape file

Files can be written by specifying the start address of the data and the number of bytes to be output.

1. System call address: BD3CH
2. Entry Conditions:
   a. The X register should contain the start address (0000H ⟨ = X ⟨ = FFFFH) for the file to be written.
   b. The U register should contain the number of bytes to be written minus one (0000H ⟨ = U ⟨ = FFFFH).
3. Exit Conditions: Check sum data is output at the rate of 2 bytes for each 80 bytes written. The number of check sum bytes is not included in the U register number of bytes to be output.
4. Flags: CARRY = 0 if Output ended normally
   = 1 if BREAK key was pressed

Read Tape Synchronization Header

Before the header can be read from tape, you must construct a header using the BBD6H call. This will specify the file type. If you are searching for a particular file, you may place the file name in address locations 7B69H - 7B78H. If you specify a file name, the tape will be searched for a matching name. If you do not specify a file name (file name = all 00H characters) then file names will be ignored during input.

1. System call address: BCE8H
2. Entry Conditions:
   a. build a header with file type
   b. specify a file name if you wish.
   c. Set 7879H: Bit Seven = 1

Bit Four = 0 for Remote 0
= 1 for Remote 1

3. Exit Conditions:
   a. 7B91H - 7BA0H will contain the 16 character file name (padded with 00H characters if file name was less than 16 characters)
   b. 7BA1H - 7BAFH will contain whatever was in 7B79H - 7B87H when the file was written to tape.
4. Flags: Carry = 0 Reading finished
   = 1 BREAK key pressed

Read a Character from Tape

1. System call address: BDF0H
2. Entry Conditions:
3. Exit Conditions: The data value read from the tape is placed in the accumulator.
4. Flags: Carry = 0 Byte read properly
   = 1 BREAK key was pressed

Read a file from tape

1. System call address: BD3CH
2. Entry Conditions:
   a. The X register contains the first memory address (0000H < = X < = FFFFH) that the file is to be loaded into.
   b. The U register contains the number of bytes minus one (0000H < = U < = FFFFH) to be read from tape.
   c. Address 7879H bit seven contains zero
      bit six = 0 for data read
      = 1 for data verify
3. Exit Conditions:
   a. Check sum information is automatically checked during tape input.
   b. The X register contains the address of the last data byte plus one.
4. Flags: Carry = 0 if loading ended normally
   = 1 abnormal end, check H and V flags
   H = 1 if C = 1 then BREAK key pressed
   = 0 check V flag
   V = 1 if C = 1 and H = 0 then data in memory and the data from the tape did not verify properly.
   = 0 if C = 1 and H = 0 then a check sum error occurred.

Finishing Tape I/O Activities

When you are finished using tape I/O you should inform the system.
1. System call address: BBF5H
2. Entry Conditions: Bit seven of 7879H should be a zero to terminate data output or a one to terminate data input.
3. Exit Conditions:
   a. The serial port is reset
   b. Printer Paper Feed is enabled
   c. Cassette motor drives are turned off.
4. Flags:

BASIC Program Tapes

The PC-2 creates and reads tapes for BASIC program files using the file read and write routines described here. Before the synchronization header is written to tape, the PC-2 stores the length of the program (in bytes) minus one in locations 7B85H and 7B86H. This information is then recorded as part of the synchronization information for later use in reading the file. When the header information is read back during a synchronization header read, the length information is in 7BACH and 7BADH.

KEYBOARD INPUT CALLS

Scan Keyboard, wait for a key to be pressed

1. System call address: E243H
2. Entry Conditions:
3. Exit Conditions:
   a. Key code is in the accumulator
   b. SHIFT, DEF, and SML do not cause this routine to return.
   c. Auto power off will occur after about seven minutes if no key is pressed.
   d. If the BREAK key is entered, execute the following: ANI #F00BH, 0FDH (FDH E9H F0H 0BH FDH)
4. Flags: Carry 0 = Accumulator has key code
   1 = BREAK key, Accumulator = 0EH

Key Code Table

	0	1	2	3	4	5	6	7
0			SPACE	0	@	P		p
1	(SHIFT)	F1	!	1	A	Q	a	q
2	(SML)	F2	"	2	B	R	b	r
3		F3	#	3	C	S	c	s
4		F4	$	4	D	T	d	t
5		F5	%	5	E	U	e	u
6		F6	&	6	F	V	f	v
7				7	G	W	g	w
8	←	CL	(	8	H	X	h	x
9	⇕	RCL	)	9	I	Y	i	y
A	↓	CA	*	:	J	Z	j	z
B	↑	(DEF)	+	;	K	rad	k	
C	→	INS	,	<	L		l	
D	ENTER	DEL	-	=	M	π	m	
E	BREAK		.	>	N	∧	n	
F	OFF	MODE	/	?	O		o	

Scan keyboard and Return

1. System call address: E42CH
2. Entry Conditions:
3. Exit Conditions:
   a. If no key was pressed, accumulator = 00H
   b. If a key was pressed, Key code is in accumulator
4. Flags:

NUMERIC FUNCTION CALLS

From the documentation, it appears that numeric functions are called with the X register pointing to 7A00H - 7A07H and the Y register pointing to 7A10H - 7A17H if Y is needed. Results appear to always be stored in 7A00H - 7A07H. Numeric data is stored in these memory areas as previously described.

Two Variable Numeric Functions

Addition	X + Y→X	EFBAH
Subtraction	X - Y→X	EFB6H
Multiplication	X * Y→X	F01AH
Division	X / Y→X	F084H
Exponentiation	X∧Y→X	F89CH

Single Variable Numeric Function

Square Root	SQR X→X	F0E9H
Logarithm	LN X→X	F161H
	LOG X→X	F165H
Exponentials	EXP X→X	F1CBH
	10∧X→X	F1D4H
Sine	SIN X→X	F3A2H
Cosine	COS X→X	F391H
Tangent	TAN X→X	F39EH
Arcsine	ASN X→X	F49AH
Arccosine	ACS X→X	F492H
Arctangent	ATN X→X	F496H
	DEG X→X	F531H
	DMS X→X	F564H
Absolute Value	ABS X→X	F597H
Signum Function	SGN X→X	F59DH
Integer Function	INT X→X	F5BEH

OPERATIONS WITH STRINGS

ASC and LEN Subroutines

1. System call address: D9DDH
2. Entry Conditions:
a. Character string information is stored in 7A04H - 7A07H as previously described.
b. YL = 60H for ASC
= 64H for LEN
3. Exit Conditions:
a. The result is in 7A00H - 7A07H
b. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

CHR$ Subroutine

1. System call address: D9B1H
2. Entry Conditions:
a. Integers from 0 - 255 are placed into 7A07H.
b. 7894H = 10H
3. Exit Conditions:
a. If UH = 0 then a proper exit occurred, otherwise UH contains the error code.
b. 7B10H contains the ASCII code
c. 7A04H - 7A06H contain C1H 7BH 10H
d. If the ASCII code was 00H then 7A07H contains 00H otherwise, 7A07H contains 01H.
4. Flags:

VAL Subroutine

1. System call address: D9D7H
2. Entry Conditions: string information is in 7A00H - 7A07H.
3. Exit Conditions:
a. The result is in 7A00H - 7A07H
b. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

STR$ Subroutine

1. System call address: D9CFH
2. Entry Conditions:
a. numeric value to be converted is in 7A00H - 7A07H
b. 7894H = 10H
3. Exit conditions:
a. The string pointer is in 7A00H - 7A07H
b. The actual character string is stored at 7B10H and following.
c. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

RIGHT$(X$,Y), LEFT$(X$,Y), and MID$(X$,Y,Z) Subroutines

1. System call address: D9F3H
2. Entry Conditions:

	RIGHT$	LEFT$	MID$
(7890H)	((7891H)-8	same	((7891H)-16
(7892H)	(7890H)+8	same	(7890H)+16
(7894H)	10H	10H	10H
7A00H–7A07H	Y	Y	Z
(7890H)–(7890H)+7	X$	X$	X$
(7890H)+8-(7890H)+15	--	--	Y
YL	02H	7AH	7BH

3. Exit Conditions:
a. The string pointer is in 7A00H - 7A07H
b. The actual character string is stored at 7B10H and following.
c. UH contains the error code (00H is a normal finish) if an error occurred.
4. Flags:

Note: (7890H) and (7891H) cannot be overwritten or changed. If these are changed, the routine will not function properly.

String Concatenation

1. System call address: D925H
2. Entry Conditions:
a. 7894H = 10H
b. Information on the first character string is stored in 7A00H - 7A07H
c. Information of the second character string is stored in 7A10H - 7A17H in the same format as previously described.
3. Exit Conditions:
a. Information on the new character string is placed in 7A00H - 7A07H.
b. Actual concatenated string is put in 7B10H and following memory locations.
c. If an error occurs, UH contains the error code.
4. Flags: