# 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.

### BLOCK DIAGRAM



### OUTLINE OF THE 8-BIT CMOS MPU

The 8-bit MPU chip (LH5801) uses CMOS static technol-

ogy. 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 contro
- 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 BH CMOS MPU Block Diagram

### **MPU SIGNALS**

•D—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, BRO, is issued.



MPU Internal Registers and Elip-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.
- 3FO, 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 Ilip-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
- + : 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:

 $\Sigma$ 7,  $\Sigma$ 6,  $\Sigma$ 5,  $\Sigma$ 4,  $\Sigma$ 3,  $\Sigma$ 2,  $\Sigma$ 1,  $\Sigma$ 0, with carry of C7, C6, C5, C4, C3, C2, C1, C0. The input conditions for each of flags shall be as described below:



- 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 IRO 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

D

Memory Map: 000 3FFF Module ROM - 16K 400 47FF User RAM 2K 400 47FF User RAM 2K 400 5FF Module ROM - 16K 400 5FF Duplicate of Annota key definitions 475 47FF Program Wenders 1 Momers 475 47FF Duplicate of 7600 - 78FF 760 77FF Duplicate of 7600 - 78FF 760 77FF Duplicate of 7600 - 78FF 767 766 56 770 77FF Duplicate of 7600 - 78FF 768 768 768 76 768 768 768 76 768 768 768 76 769 77FF Duplicate of 7600 - 78FF 760 77FF Duplicate of 7600 - 78FF 760 77FF Duplicate of 7600 - 78FF 760 77FF Duplicate of 7600 - 78FF 770 77FF Duplicate of 7800 - 78FF 770 77FF Duplicate of 7800 - 7800 Memory Map:

	79F.B	79EF		
	79F0	Printer	r Text/Graphic mode	
	79F2	Printer	r ROTATE value	
	79F3	Printer	r pen color r CSIZE	
	7400	7407 Num	ieric Data Butter or String pointer	
	7A10 -	7A17 Nume 7B4F Struct	eric Data Bulter or String pointer a Bulter	
	7B60 ·	7B67 Tape	out Synchronization header	
	7868 -	Tape ou 7878 Tape	ut file mode	
	7879 -	7B84 Tape	out hearder (available to user)	
	7885	7886 Tape 7888 Tape	e out # bytes in BASIC He -1	
	7891	7BA0 Tape	a in file name	
	7BA1 - 7BAC	78AB Tape 78AD Tape	e in user header he in # bytes in BASIC life -1	
	7BAE	7BAF Tape	e in end header	
7000.	7660	Dunlicat	te of 7800 · 7BFF	
8000 -	BFFF	Expansi	ion ROM - 16K	
	A519	Chan	ge printer pen color	
	A769 A781	Send	er motor out I ASCII character to printer (no LF)	
	ARDD	Move	e pen	
	A9F1	Send	tine feed (LF) to printer	
	AAD9	Pen	Up/Down	
	ABCB	Swite	ch printer from graphic to text mode ch printer from text to graphic mode	
	BBD6	Write	e tape synchronization header	
	BBF5 BCF8	Finali Read	lization of tape 1/U control d tape synchronization header/search for filename	
	BD3C	Read	d/Write file to tape	
	BDCC	Send	d a character to lape	
	BFII	Turn	tape drive on	
C000	BF43	Turn I System	Program BOM - 16K	
	D0D2	Mag	initude Comparison for Numeric Values	
	DOF9	Magr	initude Comparison for Character Strings	
	D2EA	Find	address of variable	
	D925	String	g concatenation	
	D9CF	STR	\$	
	D9D7		2 # Y1 = 60H 1 EN # YL = 64H	
	D9F3	RIGH	HTS, LEFTS, MIDS	
	E243	Keyb	hoard Scan - wait for character	
	E42C	Keyb	board Scan - no wat	
	EBCA	Disp	play contents of display buffer	
	ED3B	Outp	put n characters to LCD beginning at cursor = 0	
	FD40		out one char to LCD and increment cursor position by one	
	5067	Outr	out one character to I CD	
	ED57 ED95	Outr Outr Conv	put one character to LCD ivert two bytes of ASCII code (0-9,A-F) into one byte of hex d	lata
	ED57 ED95 EDEF	Oulp Con Oulp	put one character to LCD vert two bytes of ASCII code (0.9,A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$	lata
	ED57 ED55 ED55 EDEF EFB6 EFBA	Oulp Oulp Conv Outr X - Y X +	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$	lata
	ED57 ED95 EDEF EFB6 EFBA F008	Outp Outp Com Outp X - Y X + NOP	put one character to LCD wert two bytes of ASCII code (0-9,A-F) into one byte of hex d put one graphic column to current cursor position Y → X •Flag 2 Y → X	lata
	ED57 ED95 EDEF EFB6 EFBA F00B F01A F084	Outp Outp Com Outp Com X - 1 X + NOP X - 1 X / Y	put one character to LCD wert two bytes of ASCII code (0-9,A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$	lata
	ED57 ED95 EDEF EFB6 EFBA F008 F01A F084 F0E9 E161	Outp Outp Comp Outp X - 1 X + I/OP X - 1 X / Y SOR I N X	put one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ ( $X \rightarrow X$ ( $X \rightarrow X$ )	lata
	ED57 ED95 ED95 ED86 EF86 EF86 F084 F084 F084 F089 F161 F165	Outp Comp Contr X - 1 X	put one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ ( $\rightarrow X$ ( $x \rightarrow X$ ( $x \rightarrow X$ )	lata
	ED57 ED95 EDEF EFB6 EFB8 F008 F01A F084 F084 F069 F161 F165 F1CB	Outp Comp Comp X - 1 X -	put one character to LCD wert two bytes of ASCII code (0.9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ ( $\rightarrow X$ 1X $\rightarrow X$ ( $\rightarrow X$ ( $\rightarrow X$ 2X $\rightarrow X$ 2X $\rightarrow X$	lata
	ED57 ED95 EDEF EFB6 EFBA F008 F01A F084 F084 F084 F084 F084 F084 F084 F084	Outp Comp Contr X - + I/OP X - Y SOR LOG EXP 10 - COS	but one character to LCD wert two bytes of ASCII code (0-9.4.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ (Flag 2 $Y \rightarrow X$ ( $\rightarrow x$ ( $\rightarrow x$ ( $\rightarrow x$ ( $\rightarrow x$ ( $\rightarrow x$ ) ( $\rightarrow x$ ))( $\rightarrow x$ )( $\rightarrow$	lata
	ED57 ED95 ED95 EDEF EFB6 EFBA F00B F01A F029 F161 F165 F1CB F1CB F1C4 F391 F392 F342	Outp Outp Contr X -+ VO X - Y SOR SOR LOG EXP 10 - COS TAN SIN	but one character to LCD wert two bytes of ASCII code (0.9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ ( $\rightarrow X$ ) ( $\rightarrow X$	lata
	ED57 ED57 ED55 EDEF EFB6 F01A F084 F01A F084 F01A F165 F1CB F1CB F1C4 F39E F3A2 F392 F492	Outp Outp Contr X + VO X Y X Y SOR LOG EXP EXP EXP COS TAN SIN S	but one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ ( $\rightarrow X$ ) ( $\rightarrow X$	lata
	ED57 ED95 EDEF EFB6 EFB6 F01A F00B F01A F029 F161 F104 F391 F392 F392 F392 F492 F49A	OUID OUID COM OUID X - 1 X + P X - Y X - X X - Z X - X X - Z X - Z X - Z X - X X - Z X - X X - Z X - Z X - X X - X	but one character to LCD wert two bytes of ASCII code (0-9,A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $/ \rightarrow X$ $/ \rightarrow X$ $/ \rightarrow X$ $/ \rightarrow X$ $/ x \rightarrow X$	lata
	ED57 ED57 ED56 EFBA F08 F01A F084 F084 F084 F084 F084 F084 F084 F084	Outp Outp Convic X - 1 X - 4 X - 4 X - 7 X	but one character to LCD wert two bytes of ASCII code (0-9,A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ ·Flag 2 $Y \rightarrow X$ · $Y \rightarrow X$ · $X \rightarrow X$	lata
	ED57 ED57 ED56 EFBA F00B F01A F089 F161 F165 F1CB F104 F392 F496 F496 F496 F497 F498 F498 F531 F564 F591	Outp Outp Com Com Com Com Com Com Com Com Com Com	but one character to LCD vert two bytes of ASCII code (0-9,A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ ·Flag 2 $Y \rightarrow X$ · $X = X$ $X \rightarrow X$ X = X $X \rightarrow X$ $X \rightarrow$	lata
	ED57 ED55 ED55 ED55 ED55 ED55 ED55 ED55	Outp Outp Com Outp Com V - 1 V X + VOP V X - V V X - V	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	lata
	ED57 ED55 ED55 ED55 ED55 ED55 ED55 ED55	Outp Outp Com Outp Com V - 1 V X + VOP V X + V X	but one character to LCD wert two bytes of ASCII code (0.9.4.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	jata
FF00 ·	ED57 ED57 ED55 E055 E055 E055 E055 F001A F005 F001A F005 F001A F005 F001A F005 F001A F005 F001A F005 F105 F105 F105 F105 F105 F105 F105	Outp Outp Com Outr Com Outr X - Y X - Y X - Y X - Y SOR EXP EXP SOR EXP EXP COS COS COS COS COS COS COS COS COS COS	but one character to LCD wert two bytes of ASCII code (0.9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	iata
FF00 FFF8	ED57 ED55 ED55 ED55 ED55 ED55 ED55 ED55	Outp Outp Comp Comp X - Y VOP X - Y SOR X - Y SOR X SOR X SOR X SOR X SOR X SOR X SOR	but one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	lata
FF00 FFF8 FFFA	ED57 ED57 ED57 ED57 ED57 ED57 ED57 ED57	Outp Outp Com Com Com Com Com Com Com Com Com Com	but one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	lata
FF00 FFF8 FFFA FFFC	ED57 ED57 ED57 ED57 ED57 ED57 ED57 ED57	Outp Outp Com Com Com Com Com Com Com Com Com Com	but one character to LCD wert two bytes of ASCII code (0-9,A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	1913
FF00 FFF8 FFFA FFFC FFFE	ED57 ED95 ED95 ED95 ED95 EF95 F014 F014 F014 F014 F014 F014 F014 F014	Outp Outp Outp Com Com Com Com Com Com Com Com Com Com	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	1913
FF00 FFF8 FFFA FFFC FFFE	ED57 ED95 ED95 ED95 ED95 EF95 F084 F084 F084 F084 F084 F084 F084 F084	Outp Outp Comp Comp Comp Comp Comp Sor Sor Sor Sor Sor Sor Sor Sor Sor Sor	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	iələ
FF00 FFF8 FFF7 FFFC FFFC	ED575 ED575 ED575 ED575 ED575 EFBA F004 F004 F004 F004 F004 F004 F004 F00	Outp Outp Com Com Com Com Com Com Sor Com Com Com Com Com Com Com Com Com Com	but one character to LCD wert two bytes of ASCII code (0.9.4.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FFF00 FFFF8 FFFAC FFFE	ED575 ED575 ED575 ED575 ED575 EF384 F084 F084 F084 F084 F084 F084 F084 F101 F105 F105 F105 F105 F105 F105 F105	Outp Outp Outp Com Outr Com Outr X - Y X - Y X - Y SOR IN X - Y SOR IN X - Y SOR IN X - Y SOR COS COS ACS ACS ACS ACS ACS ACS ACS ACS ACS AC	but one character to LCD wert two bytes of ASCII code (0.9.4.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	lata
FF00 FFF8 FFFA FFFE	ED57 ED57 ED57 ED57 EF06 FF07 FF07 FF07 FF07 FF07 FF07 FF07 F	Outp Outp Comp Comp Comp X - Y VOP X - Y SOR X	but one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FF00 FFFF8 FFFFE FFFE	ED57 ED57 ED57 ED57 ED57 EFBA F084 F084 F084 F084 F084 F084 F084 F161 F165 F161 F165 F161 F165 F161 F165 F167 F391 F392 F392 F392 F392 F392 F392 F392 F392	Outp Outp Comp Comp Comp Comp X - V VOP X - V SOR EXP COS TAN SIN : ACS COS TAN SIN : ACS SIN :	but one character to LCD wert two bytes of ASCII code (0-9.A-F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FF00 FFF8 FFFA FFFC FFFE	ED575 ED575 ED575 ED575 ED575 EF154 F064 F0757 F165 F104 F1655 F104 F1655 F104 F1655 F104 F1657 F104 F1657 F104 F107 F107 F107 F107 F107 F107 F107 F107	Outp Outp Comp Comp Comp X - 1 VOP X - 1 X - 1 X1 X - 1 X - 1	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $X \rightarrow X$	ata
FF00 FFF8 FFFA FFFC FFFE	ED575 ED575 ED575 ED575 ED575 EFBA F00E F10E F10E F10E F10E F10E F10E F10E	Outp Outp Comp Comp Comp X - 1 VOP X - 1 X - 1 X1 X - 1 X - 1	put one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FF60 FFF8 FFFA FFFC FFFE	ED575 ED575 ED575 ED575 ED575 ED575 EF586 FF04 FF04 FF04 FF04 FF04 FF04 FF04 FF0	Outp Outp Com Com Com Com Com Com Com Com Com Com	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FFF00 FFFF8 FFFFC FFFFC FFFFC	ED57 ED57 ED57 ED57 ED57 EFBA F084 F084 F084 F084 F084 F084 F084 F084	Outp Outp Outp Com Outp Com Com X · Y X · Y Sor X · Sor Sor Sor Sor Sor Sor Sor Sor Sor Sor	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FFF8 FFFF8 FFFFE	ED57 ED57 ED57 ED57 ED57 ED57 EFBA F084 F084 F084 F084 F084 F084 F084 F084	Outp Outp Outp Com Com Com Son X - Y NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	
FFFF8 FFFFA FFFFE	ED57 ED57 ED57 ED57 ED57 EFBA F084 F084 F084 F084 F084 F084 F084 F084	Outp Outp Outp Comp Comp Comp X - Y NOP X - Y SOR SOR LOG EXP COS TAN SIN : COS TAN SIN : COS SIN : COSI	but one character to LCD wert two bytes of ASCII code (0.9.A.F) into one byte of hex d put one graphic column to current cursor position $Y \rightarrow X$ Flag 2 $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $Y \rightarrow X$ $X \rightarrow X$	

F

# 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.

Hex

Mnemonic	Symbolic Operation	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)	$A + (U) + C \rightarrow A$	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)	$A + \#(Y) + C \rightarrow A$	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	<b>B3</b> i	2	7	
ADI (X),i	$(X) + i \rightarrow (X)$	4Fi	2	13	
ADI (Y),i	(Y) + i - (Y)	5 <b>F</b> i	2	13	
ADI (U).i	(U) + i - (U)	6F i	2	13	
ADI (ab),i	(ab) + - (ab)	EFabi	4	19	
ADI #(X).i	#(X) + - #(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)	FDEFabi	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.

...

Her

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	$UL + A \rightarrow UL$	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	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)	$A \wedge \#(ab) \rightarrow A$	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	$(X) \land i \rightarrow (X)$	49 i	2	13
ANI (Y),i	$(\Upsilon) \land i \rightarrow (\Upsilon)$	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) ^ + - #(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 binarycoded-decimal (BCD) system and the result is stored in the accumulator. Change may take place in C, H, Z, or V.

intentionic Symbolic Opera			
$\begin{array}{cccc} DCA (X) & A + (X) + C - A \\ DCA (Y) & A + (Y) + C - A \\ DCA (U) & A + (U) + C - A \\ DCA \#(X) & A + \#(X) + C - A \end{array}$	A 8C	1	15
	A 9C	1	15
	A AC	1	15
	A FD 8C	2	19

		55.00	•	
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	Op-Code	Byte	Cycle
DCS (X)	$A - (X) - \overline{C} \rightarrow A$	0C	1	13
DCS (Y)	$A - (Y) - \overline{C} - A$	1C	1	13
DCS (U)	A - (U) - C - A	2C	1	13
DCS #(X)	A – #(X) – C – A	FD 0C	2	17
DCS #(Y)	$A - \#(Y) - \overline{C} - A$	FD 1C	2	17
DCS #(U)	A – #(U) – 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 16bit R is decremented.

Symbolic Operation	Op-Code	Byte	Cycle
$A - 1 \rightarrow A$	DF	1	5
XL-1 - XL	42	1	5
YL-1 - YL	52	1	5
UL-1-UL	62	1	5
XH-1-XH	FD 42	2	9
YH-1 - YH	FD 52	2	9
UH-1-UH	FD 62	2	9
X – 1 → X	46	1	5
Y-1 - Y	56	1	5
U-1 → U	66	1	5
	Symbolic Operation A - 1 - A XL - 1 - XL YL - 1 - YL UL - 1 - UL XH - 1 - XH YH - 1 - YH UH - 1 - UH X - 1 - X Y - 1 - Y U - 1 - U	Symbolic Operation         Op-Code $A-1 \rightarrow A$ DF $XL-1 \rightarrow XL$ 42 $YL-1 \rightarrow YL$ 52 $UL-1 \rightarrow UL$ 62 $XH-1 \rightarrow XH$ FD 42 $YH-1 \rightarrow YH$ FD 52 $UH-1 \rightarrow UH$ FD 62 $X-1 \rightarrow X$ 46 $Y-1 \rightarrow Y$ 56 $U-1 \rightarrow U$ 66	Symbolic OperationOp-CodeByte $A-1 \rightarrow A$ DF1 $XL-1 \rightarrow XL$ 421 $YL-1 \rightarrow YL$ 521 $UL-1 \rightarrow UL$ 621 $XH-1 \rightarrow XH$ FD 422 $YH-1 \rightarrow YH$ FD 522 $UH-1 \rightarrow UH$ FD 622 $X-1 \rightarrow X$ 461 $Y-1 \rightarrow Y$ 561 $U-1 \rightarrow U$ 661

**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
EALi	A ⊕ i → A	BD i	2	7

 $\textbf{Comment} - \oplus \cdot \textbf{represents the XOR operation}$ 

**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	ADab	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	$X + 1 \rightarrow X$	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	28	1	7	
ORA (ab)	A v (ab) - A	ABab	3	13	
ORA #(X)	A ∨ #(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.

Symbolic Operation	Hex Op-Code	Byte	Cycle	
Avi - A	BB i	2	7	
(X) v i - (X)	4B i	2	13	
$() v i \rightarrow ()$	5B i	2	13	
$(U) \vee i \rightarrow (U)$	6B i	2	13	
(ab) v i - (ab)	EBabi	4	19	
#(X) v i #(X)	FD 4B i	3	17	
#(Y) v i → #(Y)	FD 5B i	3	17	
#(U) v i #(U)	,FD 6B i	3	17	
#(ab) v i → #(ab)	FD EB a bi	5	23	
	Symbolic Operation A $v i - A$ (X) $v i - (X)$ (Y) $v i - (Y)$ (U) $v i - (U)$ (ab) $v i - (ab)$ #(X) $v i - #(X)$ #(Y) $v i - #(Y)$ #(U) $v i - #(U)$ #(ab) $v i - #(ab)$	Hex           Symbolic Operation         Op-Code           A $\vee i \rightarrow A$ BB i           (X) $\vee i \rightarrow (X)$ 4B i           (Y) $\vee i \rightarrow (Y)$ 5B i           (U) $\vee i \rightarrow (U)$ 6B i           (ab) $\vee i \rightarrow (X)$ FD 4B i           #(X) $\vee i \rightarrow #(X)$ FD 5B i           #(Y) $\vee i \rightarrow #(Y)$ FD 6B i           #(ab) $\vee i \rightarrow #(ab)$ FD EB a b i	Symbolic OperationOp-CodeByteA vi $\rightarrow$ ABB i2(X) vi $\rightarrow$ (X)4B i2(Y) vi $\rightarrow$ (Y)5B i2(U) vi $\rightarrow$ (U)6B i2(ab) vi $\rightarrow$ (db)EB a b i#(X) vi $\rightarrow$ #(X)FD 4B i#(Y) vi $\rightarrow$ #(Y)FD 5B i#(U) vi $\rightarrow$ #(U)FD 6B i#(ab) vi $\rightarrow$ #(ab)FD EB a b i	

**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-C-A	00	1	6
SBC YL	A – YL – C – A	10	1	6
SBC UL	A-UL-C-A	20	1	6
SBC XH	A−XH−C→A	80	1	6
SBC YH	A-YH-C→ A	90	1	6
SBC UH	A-UH-C-A	AO	1	6
SBC (X)	A (X) - C A	01	1	7
SBC (Y)	A - (Y) - C - A	11	1	7
SBC (U)	A – (U) – C – A	21	1	7
SBC (ab)	A - (ab) - C - A	Alab	3	13
SBC #(X)	A – #(X) – C – A	FD 01	2	11
SBC #(Y)	A – #(Y) – C – A	FD 11	2	11
SBC #(U)	A – #(U) – C – A	FD 21	2	11
SBC #(ab)	A – #(ab) – C → A	FD A1 a b	4	17
			<u> </u>	

**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-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.

Here

Hex

Here

Mnemonic Symbolic Operation		Op-Code	Byte	Cycle	
BILA.i	A^i - Z	BF i	2	7	
BII (X),i	$(X) \wedge i - Z$	4D i	2	10	
BII (Y).i	$(\dot{n}) \wedge i - Z$	5D i	2	10	
BII (U),i	(Ú) ^ i — Z	6D i	2	10	
Bll (ab).i	(ab) ^ i - Z	EDabi	4	16	
BIL #(X).i	¥(X) ^ i → Z	FD 4D i	3	14	
BII #M.i	#mí^i→ Z	FD 5D i	3	14	
BIL #(U).i	#(Ú) ^ 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	ation			

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	Op-Code	Byte	Cycle	
BIT (X)	$A \wedge (x) \rightarrow Z$		1	7	
BIT	$A \land \dot{M} \rightarrow Z$	1F	1	7	
BITIO	A ^ (U) - Z	2F	1	7	
BIT (ab)	A ^ (ab) - Z	AFab	.3	13	
BIT #00	$A \wedge \#(X) \rightarrow Z$	FD OF	2	11	
BIT #(Y)	A ^ #() - Z	FD 1F	2	11	
BIT #(1)	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	c Symbolic Operation			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
CPAM	A-M				17	1	7
CPA (U)	A – (Ú)				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 #(Ú)	A – #(U)				FD 27	2	11
CPA #(ab)	A - #(ab)				FD A7 a b	4	17
Comment-	- If	С	Z	V	н		
	A) op	1	0	•	•		
	A=op	1	1	•	•		
A(op		0	0	•	•		

 ${\sf V}$  and  ${\sf H}$  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	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 CPI UL,i CPI XH,i CPI YH,i CPI UH,i COmment	YL −i UL −i XH −i YH −i UH −i (op) > i	C 1	Z 0	5 6 4 5 6 V •	Ei Ei Ci Ci H·	2 2 2 2 2	7 7 7 7 7
	(op) = i (op) ( i	1 0	1 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	Op-Code	Byte	Cycle
ATT Comment—T -	A → T Status Register	FD EC	2	9

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

Mnemonic	Symbolic Operation	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 M	(Ý) - A	15	1	6
LDA (Ú)	(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.

		Hev		
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle
LDE X	$(X) \rightarrow A, X - 1 \rightarrow X$	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 Opera	ition	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		AAij	3	12	
IDX-The	content of the	reaister	R. stack	pointer	S. o	r

**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.

L.

Mnemonic	Symbolic Operation	Hex Op-Code	Byte	Cycle
LDX 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	$(X) \rightarrow A, X+1 \rightarrow X$	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	$XL \rightarrow (S)$			
	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	$A \rightarrow (X), X - 1 \rightarrow X$	43	1	6
SDE Y	$A \rightarrow (Y), Y - 1 \rightarrow Y$	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	$A \rightarrow (X), X + 1 \rightarrow X$	41	1	6
SIN Y	$A \rightarrow (Y), Y+1 \rightarrow Y$	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	$A \rightarrow XL$	0A	1	5
STA YL	$A \rightarrow YL$	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)	$A \rightarrow (X)$	OE	1	6
STA (Y)	$A \rightarrow (Y)$	1E	1	6
STA (U)	A → (U)	2E	1	6
STA (ab)	A - (ab)	AEab	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	Op-Code	Byte	Cycle
STX X	$\times \rightarrow \times$	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	Х Р	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	Т — А	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	Sym	bolic	Oper	ation	Hex Op-Code	Byte	Cycle
AEX	7	4	3	0	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	Op-Code	Byte	Cycle
CIN	A - (X). X + 1 — X	F7	t	7
DRL-Perfor	ms digit-to-digit fo	rward rotation	betwee	en the

DHL — 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	Op-Code	Byte	Cycle
DRL (X) DRL #(X)		D7 FD D7	1 2	12 16
	A (X) or #(X)			

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	Op-Code	Byte	Cycle
DRR (X)		D3	1	12
DRR #(X)		FD D3	2	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	Op-Code	Byte	Cycle
ROL	-C - 7 - 0	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	Op-Code	Byte	Cycle
ROR		D1	1	9
	A			

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

SHL 
$$\begin{bmatrix} C \\ A \end{bmatrix} \leftarrow \begin{bmatrix} 7 \\ 0 \end{bmatrix} \leftarrow 0 \quad D9 \quad 1 \quad 6$$

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

SHR 
$$0 - \frac{7}{A} - C$$
 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	Op-Code	Byte	Cycle
TIN	$(X) \rightarrow (Y).$ X+1 $\rightarrow X, Y+1 \rightarrow Y$	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	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	nex 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	Op-Code	Byte	Cycle
ITA	IN – A	FD BA	2	9

### NOP-No operation

Mnemonic NOP	Symbolic Operation	Hex Op-Code 38	<b>Byte</b> 1	<b>Cycle</b> 5
OFF-Reset	s the BF flip-flop. It cau	uses no chi	ange	in the
<b>Mnemonic</b> OFF	Symbolic Operation 0 → BF	Hex Op-Code FD 4C	<b>Byte</b> 2	<b>Cycle</b> 8
RDP-Reset	ts display flip-flop.			
<b>Mnemonic</b> RDP	<b>Symbolic Operation</b> 0 — Display	Hex Op-Code FD C0	<b>Byte</b> 2	Cycle 8
REC-Reset	s the carry flag C off. If flags.	t causes no	o cha	nge in
Mnemonic REC	Symbolic Operation 0 – C	<b>Hex Op-Code</b> F9	Byte 1	Cycle 4
RIE-Resets chang	the Interrupt Enable (IE) e in other flags.	flip-flop off.	Itcau	sesr
Mnemonic RIE	Symbolic Operation	Hex Op-Code FD BE	<b>Byte</b> 2	<b>Cycle</b> 8
<b>RPU</b> —Resets the general purpose flip-flop PU off. It causes no change in other flags.				
<b>Mnemonic</b> RPU	Symbolic Operation 0 - PU	Hex Op-Code E3	Byte 1	Cycle 4
RPV-Resel	ts the general flip-flop ge in other flags.	PV off. It	caus	es no
<b>Mnemonic</b> RPV	Symbolic Operation 0 PV	Hex Op-Code B8	Byte 1	<b>Cycle</b> 4
SDP-Sets of	display flip-flop.			
Mnemonic SDP	<b>Symbolic Operation</b> 1 — Display	Hex Op-Code FD C1	Byte 2	Cycle 8
SEC-Sets ti flags.	he carry flag C on. It cau	ises no cha	nge ir	other
Mnemonic SEC	<b>Symbolic Operation</b> 1 — C	<b>Hex</b> Op-Code FB	Byte 1	Cycle 4
SIE—Sets th change	e Interrupt Enable (IE) fl e in other flags.	ip-flop on. I	lt cau	ses no
<b>Mnemonic</b> SIE	Symbolic Operation 1 → IE	Hex Op-Code FD 81	Byte 2	<b>Cycle</b> 8
SPU-Sets to chang	he general purpose flip-f ge in other flags.	lop PU on.	lt cau	ses no
Mnemonic SPU	Symbolic Operation	Hex Op-Code E1	Byte 1	<b>Cycle</b> 4
SPV—Sets th	he general purpose flip-f	lop PV on. I	lt cau	ses no
Mnemonic SPV	Symbolic Operation	Hex Op-Code A8	Byte 1	Cycle 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	8Ė i	2	8
BCH-i	P – iP	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.

		Hex		<b>C</b>
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle
BCR+i	if C = 0, P + i - P	81 i	2	8-11
BCR – i	if C = 0, P − iP	91 i	2	8-11
Comment-I	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 $C = 1$ , $P + i \rightarrow P$	83 i	2	8-11
BCS-i	if $C = 1$ , $P - i \rightarrow P$	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.

		Hex		
Mnemonic	Symbolic Operation	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.

		Hex				
Mnemonic	Symbolic Operation	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	Op-Code	Byte	Cycle
BVR+i	if V = 0, P + i→P	8D i	2	8-11
BVR – i	if $V = 0$ , $P - i \rightarrow P$	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.

		Hex		
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle
BVS+i	if $V = 1$ , $P + i \rightarrow P$	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.

		Hex		
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle
BZR+i	if $Z = 0$ , $P + i \rightarrow P$	89 i	2	8-11
BZR – i	if $Z = 0$ , $P - i \rightarrow P$	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.

Maamania	Symbolic Operation	Hex On Code	<b>D</b> urba	Cycle
Minemonic	Symbolic Operation	Op-Code	Dyle	Cycie
BZS+i	if $Z = 1$ , $P + i \rightarrow P$	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	Op-Code	Byte	Cycle
JMP i,j	i→PH, j→PL	BAij	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	
<b>Comment</b> —if borrow = 1, no jump; if borrow = 0, $P - i \rightarrow P$					

### 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.

		Hex			
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle	
SJP	PL-(S), PH-(S-1),	BĖij	3	19	
	S-2-S. i-PH. i-PL	•			

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 $C = 0$ .	C1 i	2	8.21
	$PH \rightarrow (S - 1), PL \rightarrow (S)$			
	(FFab)→PH, (FFab + 1)→	PL		
	S-2-S			
Commont	C 1 no jump ob How	diate in i		

**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	if $C = 1$ ,	C3 i	2	8.21
	PH(S1), PL(S)			
	(FFab)-PH, (FFab + 1)-	PL		
	S-2→S			
Comment-if	C=0, no jump, ab = Hex o	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.

		Hex		
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle
VEJ (ab)	PL-(S), S-1-S			
VEJ (CO)	PH→(S), S – 1→S	CO	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 (DO)	DO	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 (EO)	EO	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 (FO)	FO	1	17
VEJ (F2)	F2	1	17
VEJ (F4)	F4	1	17
VEJ (E6)	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 fla 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	Op-Code	Byte	Cycle
VHRI	⊪t H = 0, PH-→(S - 1), PL-→(S)	C5 i	2	8-21
	(FFab)→PH. (FFab+1)→PL S-2→S			
	$PH \rightarrow (S - 1), PL \rightarrow (S)$ (FFab) - PH, (FFab + 1) - PL S - 2 - S	001	E	U

**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.

		nex		
Mnemonic	Symbolic Operation	Op-Code	Byte	Cycle
VHS i	if H = 1,	CŻi	2	8.21
	PH(S-1), PL(S)			
	(FFab)-PH, (FFab + 1)-PL			
	S-2-S			
Comment-if	H=0, no jump; ab = Hex c	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	Op-Code	Byte	Cycle
VMJ i	PL(S), S - 1S PH-→(S), S - 1S	CD i	2	20
	(FFab)-PH			
	(FFab + 1)→PL			
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	Bvte	Cvcle
VVS i	if V = 1.	CF i	2	8-21
	$PH \rightarrow (S - 1), PL \rightarrow (S)$			
	(FFab)-PH, (FFab+1)-	PL		
	S-2-S			
Comment i	IV-0 no iumo: ab + Hex c	diaite in i		

**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,	C9 i	2	8-21
	PH→(S – 1), PL→(S)			
	(FFab)-PH, (FFab+1)-1	PL		
	S-2-S			

**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,	CB i	2	8-21
	PH-(S-1), PL-(S)			
	(FFab)-PH, (FFab+1)-	PL		
	S-2-S			
Comment-if	Z=0, no jump; ab = Hex d	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,	'8Å	1	14
	(S+2) - PL,			
	(S+3)-T,			
	S+3-S			

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)PI	9A	1	11
	(3+2,→1 C, S+2→S			

### 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.



.....

# 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)	$A + #(X) + C \rightarrow A$	FD 03	2	
ADC #(Y)	A+ #(Y) + C → A	FD 13	2	
ADC (ab)	A + (ab) + C - A	A3 a b	3	
ADC (U)	$A + (U) + C \rightarrow A$	23	1	
ADC (X)	$A + (X) + C \rightarrow A$	03	1	
ADC M	$A + (\gamma) + C \rightarrow A$	13	1	
ADC UH	$A + UH + C \rightarrow A$	A2	1	
ADC UL	A + UL + C → A	22	1	
ADC XH	$A + XH + C \rightarrow A$	82	1	
ADC XL	$A + XL + C \rightarrow A$	02	1	
ADC YH	A + YH + C → A	92	1	
ADC YL	$A + YL + C \rightarrow A$	12	1	
ADI #(ab).	#(ab) + i → #(ab)	FDEFabi	5	
ADI #(U),1	#(U) + i → #(U)	FD 6F i	3	

Mnemonic	Symbolic Operation	Op-Code	Byte
	$\#(X) + i \rightarrow \#(X)$	ED 4E	З
	$\frac{\pi(\lambda)}{2} + i \rightarrow \frac{\pi(\lambda)}{2}$	ED SE	3
	$\pi(1) \rightarrow \pi(1)$	FEab	4
ADI (ab),i	$(ab) \neq 1 \rightarrow (ab)$		
ADI (U),i	$(0) + 1 \rightarrow (0)$		
ADI (X),i	$(\mathbf{X}) + \mathbf{i} \rightarrow (\mathbf{X})$	451	2
ADI (Y),i	$(Y) + i \rightarrow (Y)$	5F 1	2
ADI A,i	$A + i + C \rightarrow A$	B3 i	2
ADR U	UL + A → UL	FD EA	2
ADR X	XL + A → XL	FD CA	2
ADR Y	$YL + A \rightarrow YL$	FD DA	2
	A	<b>.</b>	
AEX	7 4 3 0	F1	1
AMO	A → Timer (0-7)	FD CE	2
	$0 \rightarrow \text{Timer}(8)$		•
AM1	A → Timer (0-7)	FD DE	2
	1 → Timer (8)		
	$A \wedge #(ab) \rightarrow A$	FD A9 a D	4
	$\Delta \wedge \#(1) \rightarrow \Delta$	FD 29	2
	$A \wedge \#(X) \rightarrow A$	FD 09	2
	$\Delta \wedge \#(X) \rightarrow \Delta$	FD 19	2
	$A \land \pi(1) \rightarrow A$	49 a b	2
AND (ab)	$A \land (ab) \rightarrow A$	20	1
AND (U)	$A \wedge (0) \rightarrow A$	29	1
AND (X)	$A \land (x) \rightarrow A$	10	
AND (Y)	$A \land (f) \rightarrow A$	19	'
ANI #(ab),i	#(ab) ∧ i → #(ab)	FD E9 a D 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).	(ab) ∧ i → (ab)	E9 a b i	4
ANE (U),	(U) ∧ i → (U)	69 i	2
ANI (X).i	$(x) \land i \rightarrow (x)$	49 i	2
ANI (Y) i	$\dot{M} \wedge i \rightarrow \dot{M}$	59 i	2
ANI A,i	$A \land i \rightarrow A$	B9 i	2
ATP	A → Data Bus	FD CC	2
			_
ATT	$A \rightarrow T$	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 $C = 0, P - i \rightarrow P$	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

		Hay				Нач	
Mnemonic	Symbolic Operation	Op-Code	Byte	Mnemonic	Symbolic Operation	Op-Code	Bvte <sup>1</sup>
BHR + 1	$if H=0, P+i \rightarrow P$	85 i	2	DCA (U)	$A + (U) + C \rightarrow A$	AC	1
BHR -	if $H = 0$ , $P - i \rightarrow P$	95 i	2	DCA (X)	$A + (X) + C \rightarrow A$	8C	1
	••••		-	DCA (Y)	$A + (Y) + C \rightarrow A$	9C	1
BHS + 1	if $H = 1, P + i \rightarrow P$	87 i	2				•
BHS - I	if $H = 1, P = i \rightarrow P$	97 i	2	DCS #(U)	A – #(U) – C → A	FD 2C	2
				DCS #(X)	$A - #(X) - \overline{C} \rightarrow A$	FD OC	2
Bil #(ab).i	#(ab) ∧ i → Z	FDEDabi	5	DCS #(Y)	$A - \#(Y) - \overline{C} \rightarrow A$	FD 1C	2
BII #(U),i	#(U) ∧ i → Z	FD 6D	3	DCS (U)	$A = (U) = \overline{C} \rightarrow A$	2C	1
Bll #(X).i	$\#(X) \land i \rightarrow Z$	FD 4D i	3	DCS (X)	$A - (X) - \overline{C} \rightarrow A$	0C	1
Bll #(Y).;	$\#(Y) \land i \rightarrow Z$	FD 5D i	3	DCS (Y)	$A - (Y) - \overline{C} \rightarrow A$	1C	1
Bll (ab).	(ab) $\land \downarrow \rightarrow Z$	EDabi	4	•••			
BII (U).i	$(U) \land I \rightarrow Z$	6D I	2	DEC A	$A - 1 \rightarrow A$	DF	1
BII (X).	$(X) \land i \rightarrow Z$	4D i	2	DEC U	U – 1 → U	66	1
BII (Y).i	$(Y) \land i \rightarrow Z$	5D i	2	DEC UH	UH – 1 → UH	FD 62	2
BILA.	$A \land : \rightarrow Z$	BF i	2	DEC UL	UL – 1 → UL	62	1
				DEC X	$X = 1 \rightarrow X$	46	1
BIT #(ab)	A ∧ #(ab) → Z	FD AF a b	4	DEC XH	XH – 1 → XH	FD 42	2
BIT #(U)	$A \wedge \#(U) \rightarrow Z$	FD 2F	2	DEC XL	XL – 1 → XL	42	1
BIT #(X)	$A \wedge \#(X) \rightarrow Z$	FD OF	2	DEC Y	$Y = 1 \rightarrow Y$	56	1
BIT #(Y)	$A \wedge \#(Y) \rightarrow Z$	FD 1F	2	DEC YH	YH – 1 → YH	FD 52	2
BIT (ab)	A ∧ (ab) → Z	AFab	3	DEC YL	YL – 1 → YL	52	1
BIT (U)	$A \land (U) \rightarrow Z$	2F	1				
BIT (X)	$A \land (X) \rightarrow Z$	0F	1				
BIT (Y)	$A \land (Y) \rightarrow Z$	1F	1				2
							2
BVR +	if V = 0, P + i → P	8D i	2			07	1
BVR - I	if V = 0, P = i → P	9D i	2				
					A (X) Or #(X)		
BVS +	if V = 1, P + i → P	8F i	2				
BVS - 4	if $V = 1$ , $P = \pm \rightarrow P$	9F i	2				
				DRR #(X)		FD D3	2
BZR + 1	if Z = 0, P + i → P	89 i	2	DHH (X)	╘┰┵┰┙╴╘┰┵┰╛	D3	1
BZR -	if Z = 0, P − + → P	99 i	2		•		
					A (X) or #(X)		
BZS+ I	if Z = 1, P + i → P	8B i	2				
BZS – 1	if Z = 1, P → + → P	9B i	2	EAL	$A \oplus I \rightarrow A$	BD I	2
				500 #4.13			
CDV	0 → Divider	FD 8E	2	EOH #(ab)	$A \oplus \#(aD) \rightarrow A$		4
				EOH #(U)	$A \oplus \#(U) \rightarrow A$		2
CIN	$A - (X) \cdot X + 1 \rightarrow X$	F7	1		$A \oplus \#(A) \rightarrow A$		2
004 44-1-1		5D 47 - 1			$A \oplus \#(f) \rightarrow A$		2
CPA #(ab)	A = #(aD)	FU A/ a D	4	EOR (ab)	$A \oplus (ab) \rightarrow A$		3
	A = #(U)	FU 27	2		$A \oplus (0) \rightarrow A$	20	1
	$A - \pi(X)$	FD 07	2		$A \ominus (X) \rightarrow A$	10	1
	A - #(T)	FU 17	2	EOR (T)	$A \Theta(I) \rightarrow A$	10	1
	A = (ab)	A/ a D	3	шт			~
	A = (0)	27	1			FUBI	2
	A = (A)	17	1		$\wedge + 1 \rightarrow \wedge$	00	
	A = (1)	17	1			64	•
		26	1			50 60	2
		20	1			60	2
		00	1			44	
			4			44 ED 40	1
CPA IN		90	•			FD 40	2
CFA TE		10	1		$X \rightarrow 1 \rightarrow X$	40	1
	<b>A</b> .	07 :	2			54	1
CPI A.I			2			FD 50	2
			2			50	1
			2	ITA			2
			د ۲	11A			6
		4C 1	4			BAIL	3
		SE i	2	Sivir 1,j	i · rn,j≕r⊑		3
UPI TE,i			2		#(ab) → A	ED A5 a b	4
	A + #(1) + C - A		2		#(U) → A	FD 25	2
	$A + \#(X) + C \rightarrow A$		2		$\#(X) \rightarrow A$	FD 05	2
	$\Delta + \#(Y) + C \rightarrow \Delta$	FD 9C	2		$\#(Y) \rightarrow A$	FD 15	2
		. 5 50	-		····		-

22 TRS-80 Microcomputer News, September 1983

		Hex				Hex	
Mnemonic	Symbolic Operation	Op-Code	Byte	Mnemonic	Symbolic Operation	Op-Code	Byte
LDA (ab)	(ab) → A	A5 a b	3	PSH A	$A \rightarrow (S), S - 1 \rightarrow S$	FD C8	2
LDA (U)	$(U) \rightarrow A$	25	1	PSH U	UL → (S).		
LDA (X)	$(X) \rightarrow A$	05	1	22111	$UH \rightarrow (S-1),  S-2 \rightarrow S$	FD A8	2
	$(Y) \rightarrow A$	15	1	PSH X	$XL \rightarrow (S)$		~
		A4	1		$XH \rightarrow (S-1), S-2 \rightarrow S$	FD 88	2
	$UL \rightarrow A$	24	1	PSH T	$TL \rightarrow (S),$		2
	$XH \rightarrow A$	84	1		$(3-1), 3-2 \rightarrow 3$	FU 96	2
	$X \sqcup \rightarrow A$	04 9 <i>4</i>	1	RDP	0 → Display	ED CO	2
	$Y \rightarrow \Delta$	14	1		D' PUN' OF T		۲
		14		REC	0 - CANKY	F9	1
LDE U	$(U) \rightarrow A, U-1 \rightarrow U$	67	1				-
LDE X	$(X) \rightarrow A, X-1 \rightarrow X$	47	1	RIE	0 - IE MATRICE	FD BE	2
LDE Y	$(Y) \rightarrow A, Y = 1 \rightarrow Y$	57	1		じたられていた		
LDI A,i	i → A	B5 i	2	BOI		DB	1
LDI S.i.j	i → SH, j → SL	AA ij	3			00	•
LDI UH,i	i → UH	68 i	2		A		
LDI UL.i	i → UL	6A i	2				
LDI XH,i	i → XH	48 i	2				
LDI XL.i	i → XL	4A i	2	ROR		D1	1
LDI YH.i	i → YH	58 i	2				
LDI YL,i	i → YL	5A i	2				
				RPU	0 → PU	E3	1
LDX P	$P \rightarrow X$	FD 58	2				
LDX S	S → X	FD 48	2	RPV	0 → PV	B8	1
LDX U	U → X	FD 28	2				
LDX X	$X \rightarrow X$	FD 08	2	RTI	(S + 1) → PH,	8A	1
LDX Y	Y → X	FD 18	2		(S + 2) → PL.		
					(S + 3) → T,		
LIN U	(U) → A, U+1 → U	65	1		S + 3 → S		
LIN X	$(X) \rightarrow A, X+1 \rightarrow X$	45	1				
LIN Y	$(Y) \rightarrow A, Y+1 \rightarrow Y$	55	1	RTN	(S + 1) → PH.	9A	1
					(S + 2) → PL,		
LOP UL.i	UL – 1 → UL	88 i	2		S + 2 → S		
	If borrow = 0, $P - i$	→ P					
				SBC #(ab)	A - #(ab) - C → A	FD A1 a b	4
NOP		38	1	SBC #(U)	A – #(U) – C→ A	FD 21	2
				SBC #(X)	A – #(X) – C→ A	FD 01	2
OFF	0 → BF	FD 4C	2	SBC #(Y)	A - #(Y) - C → A	FD 11	2
				SBC (ab)	A - (ab) - C→ A	Alab	3
ORA #(ab)	A v #(ab) → A	FD AB a b	4	SBC (U)	$A = (U) = \overline{C} \rightarrow A$	21	1
ORA #(U)	$A \vee \#(U) \rightarrow A$	FD 2B	2	SBC (X)	$A = (X) = \overline{C} \rightarrow A$	01	1
OBA #(X)	$A \vee \#(X) \rightarrow A$	FD 0B	2	SBC (Y)	$A - (Y) - \overline{C} \rightarrow A$	11	1
	$A \vee \#(Y) \rightarrow A$	FD 1B	2	SBC UH	$A = UH = \overline{C} \rightarrow A$	AO	1
OBA (ab)	$A \vee (ab) \rightarrow A$	ABab	3	SBC UI	$A = U = \overline{C} \rightarrow A$	20	1
	$A \vee (II) \rightarrow A$	2B	1	SBC YH	$A = XH = \overline{C} \rightarrow A$	80	1
	$A \vee (X) \rightarrow A$	08	1	SBC XI	$A = XI = \overline{C} \rightarrow A$	00	1
	$A \vee (X) \rightarrow A$	1B	1	SBC YH	$A = YH = \overline{C} \rightarrow A$	90	1
				SBC YI	$A = YI = \overline{C} \rightarrow A$	10	1
OPL #(ab) i	#(ab) v i → #(ab)	FD FB a b i	5				
	$\#(ab) \lor I \rightarrow \#(ab)$		3	SBLA, i	A – i – C → A	B1 :	2
	$\#(\mathbf{U}) \vee \mathbf{I} \rightarrow \#(\mathbf{U})$		3				
	$\#(X) \lor I \rightarrow \#(X)$	ED 58 (	3	SDE U	A → (U), U – 1 → U	63	1
	$\pi(t) \lor t \rightarrow \pi(t)$	FD JD I	4	SDE X	$A \rightarrow (X)$ . $X - 1 \rightarrow X$	43	1
	$(ab) \lor i \to (ab)$		7	SDE Y	A → (Y), Y – 1 → Y	53	1
	$(0) \vee (- \rightarrow (0))$		2				•
	$(\mathbf{X}) \vee \mathbf{I} \rightarrow (\mathbf{X})$	4D I 5D :	2	SDP	1 → Display	FD C1	2
	$(\mathbf{r}) \vee \mathbf{I} \rightarrow (\mathbf{r})$		2			<b>FD</b>	
ORI A,I	$A \vee I \rightarrow A$		2	SEC	$1 \rightarrow C / N R^{-1}$	FB	1
			2				
	$(S+1) \rightarrow \square H$		-	SHL		D9	1
FUF U			2				
	$(S+1) \rightarrow YH$		-		2		
FUP X	$(0+1) \rightarrow (0+1) \rightarrow (0+1$		2				
	(3+2) → ∧L,3+2 → 3 (2+1) → V□		2	SHR	0-→7 0 → C	D5	1
FUP I	$(3+1) \rightarrow 1\Pi,$ $(2+2) \rightarrow \forall 1 + 2 \rightarrow 2$		2				
	(3+2) - IL,3+2 - 3		-				

TRS-80 Microcomputer News. September 1983 23

		Hex				Hex	
Mnemonic	Symbolic Operation	Op-Code	Byte	Mnemonic	Symbolic Operation	Op-Code	Byte
SIE	1 → IE	FD 81	2	VEJ (DE)		DE	1
				VEJ (E0)		EO	1
SIN U	A → (U), U+1 → U	61	1	VEJ (E2)		E2	1
SIN X	$A \rightarrow (X), X+1 \rightarrow X$	41	1	VEJ (E4)		E4	1
SIN Y	$A \rightarrow (Y), Y+1 \rightarrow Y$	51	1	VEJ (E6)		E6	1
				VEJ (E8)		E8	1
SJP	$PL \rightarrow (S),  PH \rightarrow (S-1),$	BEij	3	VEJ (EA)		EA	1
	$S = 2 \rightarrow S, i \rightarrow PH$			VEJ (EC)		EC	1
	i → PL			VEJ (EE)		FF	1
				VEJ (FO)		FO	1
SPU	1 → PU	E1	1	VE.I (E2)		F2	1
				VE. (F4)		FA	1
SPV	1 → PV	A8	1	VEJ (F6)		F6	1
	A → #(ab)	FD AF a b	4			~	•
STA #(1)	$A \rightarrow \#(1)$	ED 2E	2	VHR	if H = 0,	C5 (	2
	$A \rightarrow \#(O)$		2		$PH \rightarrow (S-1), PL \rightarrow (S)$		
SIA #(X)	$A \rightarrow \#(X)$		2		(FFab) → PH		
SIA #(Y)	$A \rightarrow \#(Y)$	FU IE	2		(FFab+1) → PL		
STA (ab)	$A \rightarrow (ab)$	AEab	3		$S - 2 \rightarrow S$		
STA (U)	A → (U)	2E	1				
STA (X)	$A \rightarrow (X)$	0E	1	VHS i	if H = 1,	C7 i	2
STA (Y)	$A \rightarrow (\Upsilon)$	1E	1		$PH \rightarrow (S - 1), PL \rightarrow (S)$		
STA UH	A → UH	28	1		(FFab) → PH		
STA UL	A → UL	2A	1		$(FFab + 1) \rightarrow P1$		
STA XH	A → XH	08	1		$S = 2 \rightarrow S$		
STA XL	A → XL	0A	1		° 2 °		
STA YH	A → YH	18	1		$PI \rightarrow (S) S = 1 \rightarrow S$	CDi	2
STA YL	A → YL	1A	1		$PH \rightarrow (S), S-1 \rightarrow S$		2
STY D	Y P	ED SE	2		(Frab) - PH		
STA F	X - S	FD 4E	2		$(Frad + 1) \rightarrow PL$		
			2				
			2	VVS i	it V = 1,	CFI	2
SIXX	$X \rightarrow X$		2		PH → (S – 1), PL → (S)		
SIXY	$X \rightarrow Y$	FU SA	2		(FFab) → PH		
					(FFab+1) → PL		
TIN	$(x) \rightarrow (Y),$	F5	1		S – 2 → S		
	$X + 1 \rightarrow X, Y + 1 \rightarrow Y$						
		<b>FD 4 4</b>	•	VZR i	if Z = 0,	C9 i	2
TTA	I→A	FU AA	2		PH → (S – 1), PL →(S)		
		<b>•</b> • •	-		(FFab) →PH		
VCRi	if C = 0,	C1 i	2		(FFab+1) →PL		
	$PH \to (S-1),  PL \to (S)$				S – 2 →S		
	(FFab) → PH						
	(FFab+1) → PL			VZS i	if Z = 1.	CB i	2
	S – 2 →S				$PH \rightarrow (S-1), PL \rightarrow (S)$		
					(FFab) →PH		
VCS	if C = 1,	C3 i	2		$(FEab + 1) \rightarrow Pl$		
	$PH \rightarrow (S-1), PL \rightarrow (S)$				$S = 2 \rightarrow S$		
	(FFab) → PH				5 2 5		
	(FFab+1) → PL					1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1	
	$S = 2 \rightarrow S$			1 Automation			
				TIRS-BO	and the second sec		
VELICON	$PI \rightarrow (S) S = 1 \rightarrow S$	CO	1				
VEJ (C2)	$PH \to (S) \ S = 1 \to S$	C2	1		un an an an an		
	(EEab) → PH	C4	1				
	$(FEab + 1) \rightarrow Pl$	С6 С6	1				
		C9	1				
		CA	1				- 1
		00	1				
VEJ (CC)			1				1
			1			1. T. S. M. L.	
VEJ (DO)		00	1				
VEJ (D2)		02		NUMERIC O	P-CODE LIST		
VEJ (D4)		U4	1	The follo	wing list presents the P	C-2 machine land	uade
VEJ (D6)		06	1	instructions	numerically and include	s the hey and de	cimal
VEJ (D8)		08	1		numerically and include	buog which are mi	eeina
VEJ (DA)		DA	1	values for the	boup po wellet an and	that we are aware	of
VEJ (DC)		DC	1	nom the list	have no valid op-code	inal we are aware	01.

Hex Value	Decimal Value	Opcode	Hex Value	Decimal Value	Opcode	Hex Value	Decimal Value	Opcode
00 01 02	00 01 02	SBC XL SBC (X) ADC XL	5D i 5E i 5F i	93   94   95	BIL (Y),i CPL YL,i ADI (Y),i	C0 C1 C2	192 193 194	VEJ (C0) VCR i VEJ (C2)
03 104 105	03 04 05		60 61	96 97	NC UL	C3   C55 - C55	195 i 196	VCS VEJ (C4) VHR i
06 07 08 00	06 07 08 00	STA XH	62 63 64	98 199 100	SDF U		199 i 200 201	VEJ (C6) VHS 1 VEJ (C8) VZP
03 0A 0B	10 11	STA XL ORA (X)	65 66 67	102 103		CA CB	202 203	VEJ (CA) VZS
OD OE OF	13 14 16	EOR (X) - STA (X) - BIT (X) -	69 i 6A i 6B	105 1 106 107 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	ANI (U):U LDI ULT ORL(U):	CD IA- CE CE CE CE CE CE CE CE CE CE CE CE CE	2051: 05 206 19071: 05	VMJ1 VEJ (CE) VVS
10	16 17	SBC YL SBC (Y)	6C 6D 6E	108 i 109 i 110 i	CPI UH	D0 D1	208	VEJ (D0) ROR
13 14 5 F C C S	16 19 920		8012 - MINT 84		SBC XHIR	D2 D3	210 211	VEJ (D2) DRR (X) VEH D4)
16 17 18	22 23 24	CPA YL ZCPA-MIE STA YH	82 831 84	130		D6 D7 D8	216	VE1 (D6) DRL (X) VEJ (D8)
19 1A 1B	25 26 27	AND (Y) STA YL ORA (Y)	85   86 87	133   134 135	BHR + CPA XH BHS +	D9 DA DB	217 218 219	SHL VEJ (DA) ROL
1D 1E	29 30	EOR (Y) = STA (Y) =	88 89 8A	137 i 138	BZR + 1 RTI	OD DE	221 222 222	INC A' VEJ (DE)
20 21	32 33	SBC UL SBC (U)	8C 8D 8E	140 141 142	DCA (X) BVR + + BCH + +	E0 E1	224 225	VEJ (E0) SPU
22 23 24	34 35 <b>36</b>		8F i	143 i 144 miles set of	BVS + 1	E2 E3 0E4	226 227 7228	VEJ (E2) RPU VEJ (E4)
25 26 27 1 2. 6	37 38 39	CPA UL ; CPA (U) ; CPA (U) ;	91 92 93 i -	145   146 147	ADC YH BCS-L	E6 E8 E9 a.b.(	230 232 233 a 5.16	VEJ (E6) VEJ (E8) ANL (ab), J
29 2A 2B	41 42 43	AND (U) STA UL ORA (U)	94 95 i 96	148 149 150	BHR	EBabi EC EDabi	234 235 a b i 236 237 a b i	VEJ (EA) ORI (ab).) VEJ (EC) Bll (ab).)
20 20 2E	44 45 46	EOR (U) STA (U)	7991 194 981	1531 154 1651	BZR = 1 RTN BZS = 1	EF ADS	280 - David State 280 - David State	
38	56	NOP	9D   9E	157 158-1	BVR - 1 BCH - 1	F0 F1 F2	240 241 242	VEJ (F0) AEX VEJ (F2)
40 41 42	64 65 66	INC XC SIN X DEC XL	9F1	460	SBC UH	F6	244 245 246	VEJ (F4) VEJ (F6) CIN
44 44 45	68 68 69	INC X LIN X	A2 A3 a b A2	162 163.a.e.	ADC UH	F9 / 52 F8	249 251	SEC
40 47 48	71 721 731		A5 a b A6 A7 a b	165 a b 166 167 a b	LDA (ab) CPA UH CPA (ab)	FD 01 0 2 FD 03 FD 05	253 03 253 03 253 055, 112	SBC #(X) ADG #(X) LDA #(X)
4A1 4B1 442-3 4C	74 1 75 1. 55 1. 76 1	CPI XL13 CPI XH1	A9 a b	168 169 a b 170 i i -+	AND (ab)	FD 07 FD 08 FD 09	253 07 253 08 253 09 253 10	LDX X AND #(X)
40 i 4E i 4F i	77   78   79	BII (X). CPI XL.1 ADI (X).1	AC AD a b AE a b	172 173 a b 174 a b	DCA (U) EOR (ab) STA (ab)	FD 0A FD 0B FD 0C FD 0C	253 10 253 11 253 12 253 13	
50 51 52	80 81 82	INC YL SIN Y DEC YL	AF a b	175 a b	BIT (ab)	FD OF FD OF	-253 14 -253 15	, STA #(X) BIT #(X)
- <b>53</b> -5-5-2-5- 54 55	83	INC Y	83 85 87	1791 1811 1831	ADIA: LOIA: CPLA	FD 11 FD 13 FD 15	253 17 253 19 253 21 253 23	SBG #(Y) ADC #(Y) LDA #(Y)
56 57 581 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 -	86 87 881 1		B9 i BA i j BB i	184 185 i 186 i i	ANLAJ JMP i.j ORLAJ	FD 18 FD 19 FD 10	253 24 253 25 253 25 253 26	AND #(Y) POP Y
SA I SB I: 5C I	901 911 921	LDI.YLI LORL(M:L) CPI.YHJ	BD3 BE i j BF J	189 1 190 1 191 - 1	SJP BILAL	FD 1B FD 1C FD 1D	253 274 253 28 253 28 253 29	EOR #(Y)



### 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:

```
200 WAIT Ø
210 CLS
220 GOURSOR 3
       REM SHIFT PRINTING RIGHT SLIGHTLY
230 PRINT "Microcomputer News"
240 FOR I=0 TO 155
     REM GRAPHIC COLUMNS
250 GCURSOR I
     : REM SET GRAPHIC CURSOR
260 A=POINT I
     : REM STORE COLUMN VALUE
270 B=0
     : REM NEW COLUMN - ALL POINTS OFF
280 FOR J=6 TO Ø STEP -1
    : REM EXAMINE DOTS
290 C=INT(A/2°J)
     : POINT ON OR OFF (1 OR Ø)
300 IF C=0 LET B=B+2^J
     : REM TURN ON LF OFF
310 A=A-C*2^J
     : REM GET READY FOR NEXT POINT
320 NEXT J
     : REM DO NEXT DOT
330 GPRINT B:
     : REM PRINT REVERSED COLUMN
```

#### 340 NEXT I : REM DO NEXT COLUMN 350 GOTO 350

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:

```
13 WAIT Ø

20 CLS

30 GCURSOR 3

40 PRINT "TRS-80 PC-2"

50 POKE 18409, 72, 118, 74, Ø, 5, 189, 255, 65, 78,

78, 153, 8

60 POKE 18421, 76, 119, 139, 6, 72, 119, 74, Ø, 158,

18, 154

83 CALL 18409 COPPO

96 NEXT 1
```

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:

- 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
<b>48</b> 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	Op-Code	Byte	
LDI XH.i	i → XH	<b>48</b> 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	8ZR – 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 re 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 inguage. 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.

C8	PSH	A
88	PSH	X
98	PSH	Y
<b>A</b> 8	PSH	U
	C8 88 98 A8	C8         PSH           88         PSH           98         PSH           A8         PSH

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,	øøн
4 <b>A</b> 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 lused 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	1 A	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	1.4	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:

```
10 WAIT Ø
20 DATA &FD, &C8, &FD, &88
30 DATA &FD, &98, &FD, &A8
40 DATA &B5, &00, &4A, &75
50 DATA &48, &78, &0E
60 DATA &BE, &E2, &43
70 DATA &BE, &ED, &4D
80 DATA &FD, &2A, &FD, &1A
90 DATA &FD, &0A, &FD, &8A
100 DATA 69A
11Ø M=16999
120 FOR I=1 TO 30
130 READ A
140 POKE M+I, A
150 NEXT I
160 M=M+1
170 PRINT "
                  READY"
180 CALL M
190 COTO 180
```

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

# 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, OFEH

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 = VMJ 746
- 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  $\langle$  = ACC  $\langle$  = 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 right
  - most end of the displayed character string on the LCD.
    - 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  $\langle$  = XL  $\langle$  = 1AH).
- 3. Exit Conditions:
- 4. Flags: Carry = 0 The character string has been displayed in 25 or fewer columns.
  - 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
- 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 the 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 9.9999999999×10E99 = x = 9.999999999×10E99.
- 7A00H contains the exponent (negative exponents are expressed as complements: 03H = ×10E3, 1FH = ×10E31, and FFH = ×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 = 0.987654321208H 80H 54H 32H 00H 00H 00H 00H =  $-5.432 \times 10$

44 TRS-80 Microcomputer News, February 1984

Integer Values:

- An integer value may fall into the range -32768  $\langle = \times \langle = 32767 \rangle$
- 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.

# JASSETTE 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
- 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
- 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).</li>
- 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
  - TRS-80 Microcomputer News, February 1984 45

- 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  $\langle = X \rangle \langle = FFFFH$ ) that the file is to be loaded into.
  - b. The U register contains the number of bytes minus one (0000H  $\zeta$  = U  $\zeta$  = 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:
  - 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) minua 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 (SHI) 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 #FO0BH, 0FDH (FDH E9H F0H 0BH FDH)
- 4. Flags: Carry 0 = Accumulator has key code 1 = BREAK key, Accumulator = 0EH
- Key Code Table

$\sim$											
	0		1	2	3	4	l –	5	6	3	7
			S	PACE	Ξ0	C		Ρ			р
(	(SHIFT	) F	-1	!	1	A	۱.	Q	é	à l	q
	(SML)	F	-2	"	2	E	3	R	t	5	r
		F	-3	#	3	C	)	S	C	2	S
		F	F <b>4</b>	\$	4	. C	)	Т	C	t	t
		F	F5	%	5	E	Ξ	U	e	Э	u
		F	F6	&	6	F	=	V	ļ	f	V
	•				7	0	à	W	ç	3	W
	←	(	CL	(	8	H	1	Х	ł	ĥ	X
	4	R	CL	)	9	. 1	1	Υ		i	У
	Ļ	0	CA	*	:	i.	}	Ζ		j	Z
	Ť	(D	EF)	+	;	ŀ	<	rad		ĸ	
			NS	,	<	: L	_			1	
E	ENTER	A D	EL	•	-	. N	A 🛛	π	r	n	
E	BREAI	K			>	· • •	1	۸	I	n	
	OFF	M	ODE	1	?	<b>(</b>	)		(	0	
E	BREAI OFF	к М(	ODE	i	> ?			۸	 (	n D	

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. GNLY SCD 1

# Two Variable Numeric Functions

Addition	X + Y→X	EFBAH	240
Subtraction	X - Y→X	EFB6H	,
Multiplication	X * Y→X	F01AH	126
Division	X / Y→X	F084H	89
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)	<b>∢(7891H)</b> –8	same	<(7891H)–16
(7892H)	(7890H)+8	same	(7890H) + 16
(7894H)	10H	10H	10H
7A00H-	Y	Y	Z
7A07H			
(7890H)-	X\$	X\$	X\$
(7890H)+7			
(7890H)+8-			Y
(7890H) + 15			
YL	02H	7AH	7BH
3. Exit Conditio	ons:		

- 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: