Vitrax Plus[tm] Controller

Document # 960627-OPP

OPERATION and PROGRAMMING MANUAL

HARDWARE
SOFTWARE

CONTROL TECHNOLOGY
P.O. Box 4605
Mountain View CA 94040

Voice: 650 962-0298
Fax: 650 962-0298

TABLE OF CONTENTS PAGE

VITRAX-PLUS Controller OVERVIEW

The Vitrax-PLUS Controller is a compact high performance digital/analog controller designed for either stand alone operation or as an intelligent embedded sub-system. It contains features most often required by commercial and industrial control applications on a *single* 4.5" x 6.5" board designed to accept optional modules that add expanded functions without adding size or complexity. Easy to use, program and interface. A complete high performance low cost controller emphasizing:

Industrial & commercial control
Equipment & sensor monitoring
Automated measurement & test
Data acquisition applications.

The power of the **Vitrax-PLUS Controller** is based on the 64180 CMOS microprocessor with its integrated peripheral circuitry and the optimized BASIC interpreter. A full-featured BASIC-in-ROM, complete with floating-point arithmetic and interrupt processor, makes the **Vitrax-PLUS** easy to program even for complex applications.

To handle a variety of applications, the **Vitrax-PLUS** System contains 64K bytes of RAM/EPROM space, a built-in EPROM programmer, 24 digital I/O lines, two RS-232 programmable serial channels and a parallel printer port. Optional modules fit directly on the single board controller: a 10-bit CMOS A/D converter with eight data channels, a real-time calendar clock with timekeeping capability from 1/10th second to 10 years, and a LCD module port with contrast control that accepts 1, 2, and 4-line modules with up to 80 characters.

SPECIFICATIONS

MICROPROCESSOR: Hitachi HD64180, CMOS design, 6.144 MHz, 8-bit data bus, 16(+3) bit address bus; accumulator, flag register and 6+6 general purpose registers, 6 special registers for stack pointer, program counter, interrupt vector, fetch counter and index addressing (2); 47 I/O registers.

DIGITAL I/O LINES: 8255A: 24 digital lines configurable as two 8-bit and two 4-bit ports; inputs or outputs with latched or strobed data. Edge connection: 2 x 25 pin header on 0.1" centers with separate ground lines for each signal line.

CONSOLE SERIAL PORT: RS-232 port utilizing Transmit data, Receive Data, Ground signals. Baud rates of 600, 1200, 2400, 4800, 9600 and 19.2K. Automatic baud rate setting. DB-9 male connector. Internal -5 volt supplied from U10.

SECONDARY SERIAL PORT: Supports Tx, Rx, CTS, RTS, DTR, DSR/DCD, and Ground signals. Baud rates to 38.4K. Internal -5 volt supplied from U10. Termination for DB-9 male connector. Support in BASIC for port configuration, data buffer and manager.

FOUR EXTERNAL INTERRUPTS Support in BASIC for interrupt service routines

MEMORY: 16K-byte ROM with BASIC Interpreter and System Control Monitor. EPROM space of 16K bytes (27128 devices) for user application code. One socket, jumperable, for static RAM memory for 8K or 32K-bytes SRAM devices.

PRINTER PORT: Epson/Centronics interface with 7 data lines, strobe (STB), Acknowledge (ACK), Busy (BSY) and interleaved ground lines. Connection provided as two rows of 13 pins on 0.1" centers. Ribbon cable pinout. Support in ROM with Cntl-P/LIST and LPRINT commands.

EPROM PROGRAMMER: Programs 27128 / 27128A EPROMs directly from RAM memory using fast algorithm with byte verify. EPROM and LOAD commands in software. Requires Vpp Programming voltage via edge connector.

FULL FEATURED BASIC in ROM:

80 Commands	Floating Point Arithmetic, double-dimension arrays
Structured interrupt handling	RS-232 port configuration with data buffer & manager
Logical operators	On-Board EPROM Programming Commands
String Manipulation	Link to Machine Code with ROM-based Monitor

SYSTEM MONITOR in ROM: Machine code debugging with HEX/ASCII Display & Edit. Breakpoint execution with CPU register/flag displays. Memory diagnostics, timer configuration and hex/decimal translations

AUTO-START OPERATION from application EPROM when power is applied.

A/D CONVERTER OPTION: 8-channels, 10-bit resolution. Settling time of 100 us with ±1 count in 1024 resolution. 0-5vdc operation. Input channels interface through a 2 x 10 header pin connector with alternate ground connections. P/C Board designed with grounded guard ring for noise immunity.

LCD MODULE PORT OPTION: Supports 1,2 & 4-line displays; Up to 80 Characters; Contrast control; 2x7 pinout with ribbon cable interface

REAL-TIME CALENDAR CLOCK OPTION: 10 year calendar: plus day-of-week, hours, minutes, seconds & 0.1 second clock. Calibration function and frequency trimmer cap

PRINTED CIRCUIT BOARD: 4.5" x 6.5" x .062"; double-sided, plated through holes, solder mask, silk-screened component legend. All connections along board edge. Options fit directly on the single board controller without the need for card cages or backplanes

POWER: +5 vdc regulated at 200 ma. Negative voltage generated on-board for RS-232 operation and LCD module contrast.

** FUNCTIONAL CHECKOUT **

This chapter describes how to confirm the operation of the **Vitrax Plus** Controller.

RAM MEMORY: Two densities of static RAM are supported: 8K x 8 and 32K x 8. One jumper at W4 is required to configure socket Z8. Refer to the Jumper Table below and Memory Maps in the Appendix.

JUMPER SETTINGS: The p/c board contains jumpers for memory, EPROM programming and serial transmission. Initially, jumpers W4 and W8 must be set.

SUMMARY TABLE OF JUMPER SETTINGS

Function	Jumper Number	Jumper Position
EPROM Change	W1	No jumper except when changing EPROMs in U7
Memory Density	W4	Lower horizontal position for 8Kx8 (HM6264)
		Upper horizontal position for 32Kx8 (HM62256)
RS-232 #2 (DCE mode)	W8	Both jumpers horizontal
RS-232 #2 (DTE mode)	W8	Both jumpers vertical
RS-232 #2 no DCD0	W9	Default trace solder side or lower horiz jumper
RS-232 #2 DCD	W9	Right vertical jumper (must cut W9 trace on solder side)

POWER CONNECTION: Main power is applied to J14 at the top of board (+ to the left as marked). Power leads are fastened to a 2-circuit screw-terminal connector. Input must be +5 vdc fully regulated. Use an industrial power supply for stability and high regulation even though current requirement is low.

SERIAL I/O (RS-232) CONNECTIONS: The serial I/O port at J3 provides a console connection to program the **Vitrax-PLUS** Controller using a computer with a serial port or a CRT terminal. A second serial I/O port with full RS-232 protocol exists at J4. The signals at these connectors are arranged in DCE configuration and terminate at DB-9 male connectors.

CONSOLE CABLE: Assemble and connect a three-wire cable between a programming console and connector J3 (RS-232 Port #1) of the Controller. Most terminals require a male DB-25 connector while many personal computer require a female cable connector. The Rx and Tx signals at oppsite ends of the cable are connected to each other - and ground is connected to ground. See the following table.

CABLE CONNECTIONS FOR PROGRAMMING CONSOLE

Signal	SYSTEM CONSOLE (DTE)		Vitrax CONTROLLER (DCE)	
	DB-25M Pin #	DB-9M Pin #	DB-9M @J3 Pin #	Signal
GND	7	5	5	Gnd
Tx	2	3	3	Rx
Rx	3	2	2	Tx

CONSOLE SETTINGS: Configure the system console to the protocol shown in the following table. Depending on the type of console, settings may be made in the software of a communication program (e.g. ProComm) - or from the keyboard or by dip switches or jumpers within the CRT terminal

FUNCTION	SETTING
Transmission	Full Duplex
Data Word	7-bit
Stop Bits	One

Parity	Even
Auto LF/CR	Off
Baud Rate	9600 or 19.2K*
Mode	On-Line (FDX)
Scroll	On
Capital Ltrs	On**

* Use 9600 or 19.2K for speed; 600, 1200, 2400 and 4800 available. Use 9600 in AutoStart Mode

** **Use CAPITAL LETTERS only; No lower case allowed**

*** Reset may be required for console and controller if parameters are changed during power on.

BAUD RATES: Three factors make it relatively easy to configure the serial ports: The CPU contains the baud rate generators; programmable I/O registers are available to set the baud rates; and the baud rate for the programming console is automatically sensed on power-up when the <Enter> key is pressed three times. The configuration software for serial channel #1 (console port at J3) exists in the BASIC ROM. The baud rate setting for serial port #2 (at connector J4) can be set with **OUT** commands to internal registers 00, 02 and 04 (Channel 0) - or with the extended capability contained in the BASIC ROM. One of its major routines contains a set of memory addresses to hold transmission parameters to configure the second channel. Other routines, CALLable from BASIC, configure the port and create a data buffer and data manager for incoming data.

INITIALIZATION: Connect the **Vitrax Plus** Controller to a programming console with a serial cable attached to connector J3. Attach a 5.0 volt DC regulated power supply to the appropriate connections at J14. Observe polarity. Apply power to the programming console. Turn on **CAPS LOCK (Shift Lock).** Switch on the Controller power supply. Press the <Enter> key up to four times. Vitrax BASIC should issue a sign-on message, ***READY*** and a "**>**" prompt character. The power-on sequence initializes the Interpreter. The Interpreter is also reinitialized by typing **NEW** <Enter>, by momentarily pressing reset switch SW-1 or by reapplying power to the board. Check that the Controller is functional by typing the following program:

```
>10 PRINT "THIS IS A TEST"
>20 FOR I=1 TO 5
>30 PRINT "TEST ",I
>40 NEXT I
>50 STOP
>RUN     <Enter>
```

The program should display:

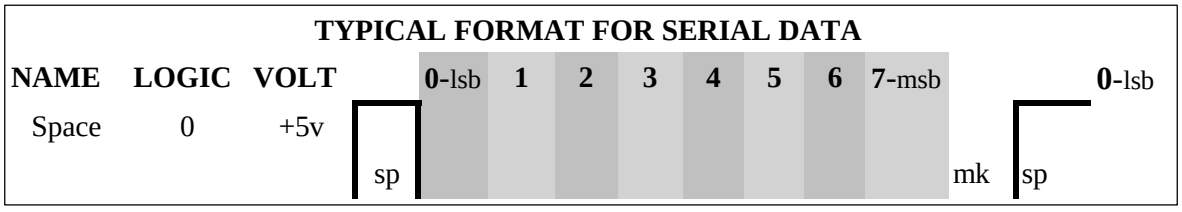
THIS IS A TEST
TEST 1
TEST 2
TEST 3
TEST 4
TEST 5

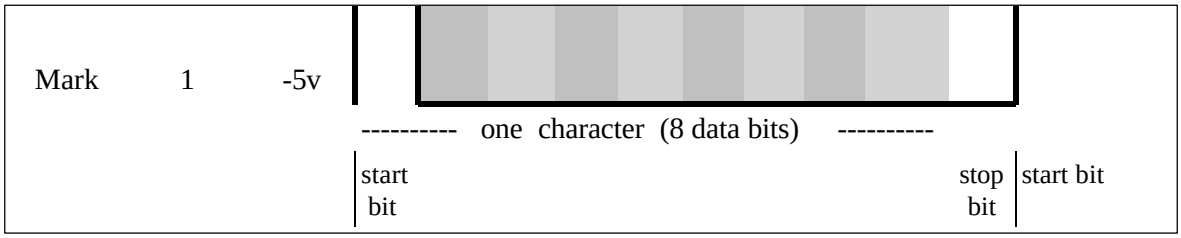
RS-232 INTERFACE: If the unit does not respond with a BASIC prompt sign (>):

* Check console settings: ***correct COM port, even parity, 7 data bits, one stop bit***.

* Some CRT terminals require CTS (pin 5) and DTR (pin 6) signals, to be pulled up to +5 volts. Look for switches in the terminal - or connect pin 20 to pins 5 (CTS), 6(DSR) and possibly 8(DCD/RLSD) or add 5K pull-up resistors to pins 5, 6 (and 8).

* Connect a scope to J3-3 (Tx). Push reset button SW1 to initialize the system. A high-going start pulse, width equal to 1/baud rate, should appear followed by a space and then by a bit pattern of ASCII characters. At 9600 baud, the start pulse should be about 100 us. See the following diagram:





One start bit, 7 or 8-bits of data, no parity, one stop bit
LOGIC 1 = more negative voltage; LOGIC 0 = more positive voltage

DIGITAL INPUT/OUTPUT LINES: The Input/Output lines are controlled by an 8255A device (U15). The **OUT** commands (below) show the code needed to produce a latched high signal on a specific pin. Use a voltmeter, logic probe or LED to check the pin's logic state after each command is entered. Use a '0' as the data byte to generate a low signal. More information is contained in the Digital I/O Section.

> NEW	Initializes system
> OUT 195,128	U15 to output
> OUT 192,16	Set J7 pin 2 high
> OUT 192,0	Set J7 pin 2 low
> OUT 193,8	Set J7 pin 50 high
> OUT 193,0	Set J7 pin 50 low

PRINTER CONNECTOR AND CABLE: Refer to the printer cable assembly diagram (Appendix). The Select In/ signal for most printers with a Centronics-type interface is asserted low by an switch inside the printer. When the switch is not present, pin 36 in the Centronics connector can be tied to ground by the printer cable via pin 30 on the Centronics connector and pin 14 on the Controller Board.

PRINTER CONNECTION: Solder a 2 x 13 pin header to the J6 printer port. Attach the cable. If the printer has a SLCT IN switch, move it to the 'selected' position. Reset the Controller. Type **Cntl-P** <Enter>. The printer should echo characters that are typed on the keyboard or are produced with **LIST**. Printing occurs when the <Enter> key is pressed. Printout is also activated by the **LPRINT** command from within a BASIC program. The **Cntl-P** command is toggled off when RUN is invoked.

** VITRAX BASIC PROGRAMMING **

This chapter describes the operation and use of Vitrax BASIC. The first section provides an overview of the language with general comments. The next section summarizes the commands, statements, functions and operators with the required format and syntax. Error messages are also explained.

Many programming examples are included in each command group - and these examples are available on MS DOS compatible diskette.

GENERAL COMMENTS

OPERATING MODES: Vitrax BASIC typically operates from a **RUN** command executing commands, statements & functions according to line numbers. BASIC can also operate in command (direct) mode without line numbers.

Commands and statements are typed at the prompt "**>**" and are executed when <Enter> is pressed. Variables can be **PRINT**ed and memory can be **PEEK**ed.

VARIABLES: Variables consist of a letter or a letter plus a single digit (e.g. A, D0, Q7). String variables add a '$' sign (e.g B8$). Single and double-dimension arrays use the form X(n) and X(n,m) for array elements. They require initial configuration with the **DIM** statement. Up to 286 non-string variables and up to 286 string variables can be used. Each defined variable occupying six bytes of memory. String variables use one byte per character plus six bytes for overhead.

NUMBERS: All numbers are handled in floating point decimal & exponential format with 6-digit precision. Numerical limits range from 2E-20 and 5E+18. The BASIC floating point algorithm is accurate to ± 1 digit in the sixth significant digit. For example, the square root of 16 may be returned as the value 3.99999. Also, BASIC returns numbers in exponential format that are less than 0.01.

COMMAND FORMAT: Most commands can be truncated to the initial three letters. Examples are **PRInt, LISt, ENAble, EPRom, RETurn**. (Exceptions are **INPUT** and **INPUT LINE**). The exclamation point, **!** , can be substituted for **PRINT** and the apostrophe, **'** , for **REMARK.** Spaces are optional. These features save memory space and speed up program execution.

CNTL-C from the console interrupts and halts program execution.

I/O ACCESS: I/O Registers within the CPU and external I/O devices (Digital I/O lines, Real time clock, A/D converter and LCD module port) are written with **OUT I/O addr, data byte** and read with **PRI INP(I/O addr)** or var=**INP(I/O addr)**.

AUTO-START MODE: When using the EPROM auto-start mode from a power-on condition, the console baud rate must be set to 9600. It can be reprogrammed with **OUT** statements to I/O Register 03, bits 0-2. Remember to preserve other bits.

MULTIPLE STATEMENTS PER LINE: Commands can be grouped on a ***single*** line if separated by colons provided the line does not exceed 72 characters. This feature helps conserve memory. Example:

```
200 PRINT "INPUT A POSITIVE NUMBER";:INPUT X:IF X <= 0 GO TO 200:B=55
250 PRINT "X =";X,"B =";B
```

If X is negative or zero, the user is prompted to enter a positive number and the program returns to line 200 for new input. When a positive number is correctly entered, the program proceeds to the next numbered line after line 200.

Care must be exercised in use of multiple statements per line. The above example shows that if the condition of the IF statement is false, control is passed to the next program line. Anything else on such a multiple statement line will be ignored. Because of the IF statement, the variable B is never set to 55.

PROTECTED USER MEMORY SPACE: Most applications can be programmed entirely in BASIC using its ability to transparently allocate memory space. Typically, the BASIC interpreter dynamically allocates RAM memory as a program is developed and debugged. As BASIC code is being written, the lines of code grow upward from the bottom of RAM while variable, string and array space fill downward from the top of RAM. Theoretically, the in-between memory space could be considered for CALLs and POKEd data. BASIC, however, contains a compaction routine that compresses fragmented memory for more efficient use. Code **POKE**d into the intervening space could eventually be moved, changed or destroyed.

Experienced programmers may wish to **POKE CALL**able machine code routines, tables, parameters, data or drivers into memory to enhance capability and performance. The initialization routine provides protected user memory space accessible only by **PEEK**s, **POKE**s and **CALL**s. It isolates BASIC program code, variables, strings and arrays from the protected RAM space above BFFFH. Memory from C000H to FF00H is available for protected routines and data. A single 32K x 8 RAM in socket Z8 will supply 16K bytes for BASIC code and nearly 16K bytes of protected space. Refer to the memory maps and the initialization section for additional information .

SUMMARY: COMMANDS, FUNCTIONS, STATEMENTS, OPERATORS

(Grouped by Function - Listed Alphabetically)

FUNCTION	COMMANDS, STATEMENTS, FUNCTIONS, OPERATORS
Arithmetic	ABS,EXP,INT,LN,LOG,RANDOMIZE,RND,SGN,SQR,+,-,*,/,^
Branching	CALL,GOSUB,GOTO,IF...THEN
Comparison	IF...THEN,>,<,=,<=,>=,><,&(AND),@(OR),%(XOR)
EPROM Programming	EPROM,LOAD
Execution	CONT,CNTL-C,RUN,STOP,XEQ
Format/Configuration	DIM,NEW,NEW*
Input	DATA,INP,INPUT,INPUT LINE,LET,PEEK,READ,RESTORE
Interrupts	DIS,ENA,OI (ON INTERRUPT)
Looping	FOR...NEXT,
Miscellaneous	FRE,RANDOMIZE,RND
Output	OUT,POKE
Print and List	CNTL-P,LIST, LPRINT,PRINT,!,REM,'
String Handling	ASC,CHR$,INPUT,INPUT LINE,INSTR,LEFT$,LEN,MID$, NUM$,STRING$,VAL,+
Trigonometric	ATN,SIN,COS,TAN
Logical Operators	&(AND), @(OR), %(XOR)

COMMANDS - (Alphabetical Listing)

The following table lists the commands in the VITRAX BASIC language along with a brief description. A reference page number is shown for each command group. Refer to the following table for notation explanation. The next section in this chapter contains expanded definitions & examples.

COMMAND	DESCRIPTION
ARITHMETIC:(page 10)	
ABS var	Absolute value of **var**
EXP var	Calculate the value of **e** to power **var**
INT var	Integer value of **var**
RANDOMIZE	Re-seed random number generator
RND var	Generate random number between 0 and **var** (default = 1)
SQR var	Square root of **var**
LOG var	Calculate logarithm to base 10 of **var**
LN var	Calculate natural logarithm of **var**
SGN var	Sign of **var** (-1, 0 or +1)
+, -, *, /,^	Add, subtract, multiply, divide, exponentiation
DIRECT COMMANDS:(page 12)	
CONT	Continue execution after STOP or break.
CNTL-C	Stop execution of the current program
CNTL-P	Toggle Printer On/Off
EPROM	Program content of RAM text area to EPROM
FRE	Returns bytes remaining in RAM not including variables

LIST	List program to console (& printer if Cntl-P is set)
LOAD	Load contents of EPROM in U7 to RAM text area
NEW	Initialize BASIC; clears variables, pointers
NEW*	Initialize BASIC; retains variable values
RUN	Execute the current program.
XEQ	Execute the current program without zeroing variables

INPUT/OUT:(page 13)

DATA value, ...	Sequential data for READ command
INP (I/O-addr)	Read data at (I/O-addr)
INPUT var, var...	Input number(s) or string variable(s) - (chnl 1 @ J3)
INPUT LINE $var	Input string variable with embedded spaces (chnl 1 @ J3)
|LET| var = expr	Assign value of expr to variable. LET is optional
PRINT expr or **!**	Output expr at J3 for display; or <crlf> if no expr
LPRINT or **!**	Same as PRINT except output to printer at J6
OUT I/O-addr,data	Output data to I/O addr
PEEK (addr)	Return the decimal value at addr
POKE addr, value	Store value at addr
READ var, var ...	Assign data to variables from a DATA table
RESTORE	Returns the DATA statement pointer to initial value

INTERRUPTS:(page 14)

DIS	Disable Interrupts (mnemonic DI,code F3 hex)
ENAble	Enable Interrupts (mnemonic EI,code FB hex)
OI Intr N line #	Sets up Interrupt service routine for **Intr N** at **line #**

LOGICAL OPERATORS:(page 17)

& {Logical AND}	Logical AND as mask for bit determination within a byte
@ {Logical OR}	Logical OR for bit determination and setting
% {Logical XOR}	Logical XOR function
>,<,=,>=,<=,><	Comparison operators (used with IF)

STRING FUNCTIONS:(page 20)

ASC $var	Returns ASCII decimal value of the first character in **var$**
CHR$ (value)	Returns ASCII character of the equivalent decimal **value**
INPUT $var,$var,..	Input text string(s); no spaces.
INPUT LINE A$	Same as **INPUT $var** but embedded spaces allowed.
INSTR(p,$var1,$var2)	Search for **$var2** in **$var1** begin at pos **p**. Returns start pos.
LEFT$(var$,n)	Return a string from character position 1 to **n** in **A$**
LEN($var)	Return the number of characters in **$var**
MID$($var,p,n)	Return a string of **n** characters start at character pos **p**
NUM$(var)	Convert **var** to a string with a leading and trailing space
RIGHT$(var$,p)	Return a string from character position **p** to end of **$var**
STRING$(n,d)	Return a string of **n** ASCII characters with decimal value = **d**
VAL($var)	Convert text string **$var** to numeric value **var**
+	Concatenate (join) strings

TRIGONOMETRIC:(page 22)

COS expr	Cosine of **expr** (in radians)
SIN expr	Sine of **expr** (in radians)
TAN expr	Tangent of **expr** (in radians)
ATN expr	Arctangent of **expr** (in radians)

OTHER COMMANDS:(page 18)

CALL addr	Execute a machine code routine at address [addr]
DIM var (expr,expr)	Dimension arrays; single or double dimension
FOR...NEXT...STEP	FOR loop and counter with NEXT and optional STEP
GOSUB line #	Call routine at [line #].
GO TO line #	Transfer control to [line #]
IF expr THEN expr..	Rest of line is executed if [expr] is true (non-zero)
NEXT var	Ending expression of FOR loop counter
READ var,var,...	Assign data to variables from DATA statement
REM or **'**	Remark statement (no operation)
RESTORE	Reset DATA statement pointer to initial value
RETURN	Return from subroutine or CALL
STEP	Parameter to increment counter in FOR...NEXT loop
STOP	Stop execution. Return to COMMAND mode. CONT to continue

EXPLANATION OF NOTATION USED IN COMMANDS

[addr]	=	address in direct memory space
[expr]	=	expression, usually numeric; or a constant or variable
[I/O-addr]	=	address in I/O space
[line #]	=	line number in a BASIC program
[str var]	=	string variable or string expression
[value]	=	numerical value, constant, data
[var]	=	variable, number or numeric expression
( )	=	parentheses that must be included as part of a BASIC command
{...}	=	Expanantory information
<...>	=	keystroke from console or terminal
|command|	=	optional|
italics	=	output from execution of a BASIC program

ERROR CODES

BC Blank Check: EPROM not erased	**OV** Arithmetic overflow; number > 5E+18
DA Insufficient data for READ	**SN** Syntax Error
EP EPROM programming error	**ST** Stack (or syntax) execution error
NX FOR ... NEXT mismatch	**UN** Arithmetic underflow; < 2E-20
OF Fatal arithmetic error	**VF** Verify error; EPROM programming

ARITHMETIC OPERATIONS

There are five arithmetic operators: **+** for addition, **-** for subtraction, ***** for multiplication, / for division and **^** for exponentiation. Arithmetic is performed in floating point format with numerical limits of 2E-20 to 5E+18. Attempted division by zero or calling for the square root of a negative number produces a break in program execution and an error code. The usual algebraic rules for order of execution are followed: Exponentiation, followed by multiplication/division,

followed by addition/subtraction. The processing order can be controlled by parentheses. Use them to add clarity and avoid confusion particularly in complicated expressions.

ADD, SUBTRACT, MULTIPLY, DIVIDE, POWER: [ARITH1]

```
10 PRINT "Program to demonstrate the Four Basic Arithmetic Functions and"
15 PRINT   "raising a Base 10 number to a power":PRINT
20 A=1.5:B=3:C=2:D=4:E=9:F=10
30 PRINT A;"+";B;"-";C;"*";D;"/";F;"   =   ";A+B-C*D/F
40 PRINT A;"*";D;"-";C;"+";B;"/";F;"   =   ";A*D-C+B/F
50 PRINT A;"*";D;"-";C;"+";B;"/";F;"^";C;"      =   ";A*D-C+B/F^C
55 PRINT A;"* (";D;"-";C;"+";B;") /";F;"^";C"  =    "A*(D-C+B)/F^C
60 PRINT B;"^3 =";B^3;"       "B;"^0.5 =";B^0.5;"       "C;"^1.5 =";C^1.5
```

```
>RUN
1.5 + 3 - 2 * 4 / 10         =  3.7
1.5 * 4 - 2 + 3 / 10         =  4.3
1.5 * 4 - 2 + 3 / 10 ^ 2     =  4.03
1.5 * ( 4 - 2 + 3 ) / 10 ^ 2  =  .075
3 ^3 = 26.9999        3 ^0.5 = 1.73204        2 ^1.5 = 2.82842
```

In lines 30 and 40, multiplication and division are performed before addition and subtraction. In line 30, C is multiplied by D (2*4=8). The result is divided by F (8/10=0.8). Then A and B are added (1.5+3=4.5) followed by the subtraction of C*D/F (0.8) to yield 3.7. A similar process occurs for line 40. In line 50, exponentiation is the first arithmetic step. Note in line 55 that parentheses change the processing order compared to line 50. The parenthetical expression, which equates to 5, is multiplied by 1.5 (=7.5) and then divided by 10^2 (=100) to produce an answer of 0.075.

EXP: EXPONENTS to Base e [EXP1]

```
10 PRINT "Program to demonstrate EXP (Base 'e' to  a Power) Function":PRINT
20 A=1:B=2:C=2.303:D=5:E=7.25:F=-2.5
30 PRINT EXP(A);EXP(B);EXP(C);EXP(D);EXP(E);EXP(F)
```

```
>RUN
2.71827  7.38905  10.0041  148.412  1408.1  .082085
```

INT: INTEGER Function [INT1]

```
10  PRINT "Program to demonstrate INT Function":PRINT
20  A=24:B=-24:C=24.75:D=-24.75:E=-.00374:F=0
30   PRINT INT(A);INT(B);INT(C);INT(D);INT(E);INT(F)
```

```
>RUN
24 -24  24 -25  0  0
```

The **INT**eger function truncates numbers at the decimal point (no rounding). Note that negative numbers truncate in the negative direction: variable D (-24.75) truncates to -25.

LOG and LN: LOGARITHMS: Base 10 and Base e [LOG1/LN1]

```
10 PRINT "Program to demonstrate LOG (Base 10) Function":PRINT
20 A=1:B=10:C=100:D=0.0001:E=20.5:F=1234
30 PRINT LOG(A);LOG(B);LOG(C);LOG(D);LOG(E);LOG(F)
```

```
>RUN
0  .999998
1.99999 -3.99999  1.31175  3.09131
```

```
10 PRINT "Program to demonstrate LN (Natural Log) Function":PRINT
```

```
20 A=1:B=2.71828:C=100:D=0.0001:E=20.5:F=1234
30 PRINT LN(A);LN(B);LN(C);LN(D);LN(E);LN(F)

>RUN
 0  .999998  4.60517 -9.21034  3.02042  7.11801
```

ABS: ABSOLUTE VALUE Function [ABS1]

```
10 PRINT "Program to demonstrate ABS Function":PRINT
20 A=24:B=-24:C=24.75:D=-24.75:E=-.00374:F=0
30 PRINT ABS(A);ABS(B);ABS(C);ABS(D);ABS(E);ABS(F)

>RUN
 24  24  24.75  24.75  3.73999E-03  0
```

SQR: SQUARE ROOT [SQR1]

```
10 PRINT "Program to demonstrate SQR (Square Root) Function":PRINT
20 A=16:B=81:C=15625:D=4.5:E=0.374:F=0.00215
30 PRINT SQR(A);SQR(B);SQR(C);SQR(D);SQR(E);SQR(F)

>RUN
 3.99999  8.99999  125  2.12132  .611555  .046368
```

RND & RANDOMIZE: RND (n) returns a pseudo-random number of six significant digits from 0 to n inclusive. If n is omitted, the range is o to 1. RANDOMIZE re-seeds the random number generator.

```
>PRI RND(22)        13.6474
>PRI RND            .794064
```

DIRECT (OPERATING SYSTEM) COMMANDS

Direct commands are executed at the system (BASIC) prompt. They are similar to the operating system commands in MS-DOS.

CONT: Continues program execution from a STOP or Cntl-C Command.

CNTL-C: Stops (breaks) execution of the current program

CNTL-P: Toggle Printer On/Off. Most useful for LISTing programs to a printer for hardcopy debugging. RUN toggles CNTL-P to Off.

EPROM: Programs the contents of RAM to EPROM. See section *Programming & Using EPROMs*.

FRE FUNCTION: The FRE function helps keep track of the amount of unused RAM by monitoring the number of bytes of RAM occupied by BASIC code statements. When **FRE** is invoked, it prints the number of bytes (decimal) from the top of BASIC code to the bottom of the stack. While **FRE** does not include the space used by variable and **POKE**d data, it helps estimate RAM usage. As code is generated, **FRE** continues to decrease. **FRE** is updated with each <Enter>. To use this command, type FRE <Enter>. To estimate the total amount of remaining RAM memory, subtract the memory used for variables, strings, arrays and **POKE**d data from the value of **FRE**. Variables use six bytes. Strings require one byte per character plus six bytes of overhead.

LIST line #n,line #m: **LIST**s the current program to the console at connector J4 starting at line number n and ending at line number m. **LIST** without line numbers **LIST**s the entire program. **LIST** followed by a single line number will **LIST** the specified line. Examples are:

LIST	Lists entire program
LIST 40	Lists Line 40
LIST 20,135	Lists program from line 20 to line 135
LIST ,55	Lists program from the beginning to line 55
LIST 85,9999	Lists program from line 85 to end of program

LOAD: **LOAD**s the contents of EPROM (U7) to RAM (U8), usually for editing. It is most useful when a CRT console is used and disk storage is not available.

NEW and NEW* initialize the interpreter. **NEW** clears all variables (while **NEW*** retains the current values), resets stacks, disables interrupts INT1 and INT2, and sets the appropriate pointers. All RAM data is lost. **Use a hardware reset (switch SW1) when the reserved BASIC area is corrupted.**

RUN: Executes the current program. See also **XEQ.**

XEQ: Executes the current program without zeroing variables

INPUT / OUTPUT COMMANDS

DATA value, value ..	Sequential data for READ command
INP (I/O addr)	Read data at <I/O addr>
INPUT var,var,...	Input number(s) and text string(s) - (chnl 1 @ J3)
INPUT A$,B$,...	Input a text string; no spaces - (chnl 1 @ J3).
INPUT LINE A$	Same as INPUT A$ but embedded spaces allowed
(LET) var = expr	Assign value of <expr> to variable
OUT I/O addr, data	Output 'data' to 'I/O addr'
PEEK addr	Return the decimal value at (addr)
POKE addr, data	Store data at addr
PRINT expr or **!**	Output <expr> at J3 for display; or <crlf> if no <expr>
LPRINT or **!**	Same as PRINT except output to printer at J6
READ var, var, ...	Assign data to variables from a DATA table
RESTORE	Returns the DATA statement pointer to initial value

DATA/READ and RESTORE: **READ** and **DATA** commands work in unison to input data sequentially from a **DATA** statement to variables defined in a **READ** statement. This command pair provides a convenient way to furnish constants to a BASIC program. The first **READ** obtains the first valve from the **DATA** statement; the second **READ**, the second **DATA** value - continuing until the final **READ**. An error message occurs if the program contains more **READ**s than **DATA** values. **RESTORE** resets the **DATA** pointer to the initial **DATA** value. Place **DATA** statements before **READ**s.

The following example of **READ/DATA** and **RESTORE** inputs data (the digits in the constant, PI) into two single-dimension arrays, X and Y. Note that the **RESTORE** command (line 110) initializes the **DATA** pointer. Subsequent READs input the same values into array 'Y' in reverse order - to show additional capability: [RD-DATA1]

```
10  REM  Program to Demonstrate READ/DATA and RESTORE Commands
20  DATA 3,1,4,1,5,9,2,6,5,3,5
30  DIM X(11)
40  DIM Y(11)
50  FOR I=1 TO 11
60  READ X(I)
70  NEXT I
80  FOR J=1 TO 11
```

```
 90     PRI X(J);:NEXT J
100     PRINT
110     RESTORE
120     FOR K=1 TO 11
130     READ Y(12-K)
140     NEXT K
150     FOR L=1 TO 11
160     PRI Y(L);:NEXT L
170     STOP

>RUN
 3  1  4  1  5  9  2  6  5  3  5
 5  3  5  6  2  9  5  1  4  1  3
STOP AT LINE 170
```

INP (I/O addr): This function inputs data to a variable from an I/O address including the I/O registers in the CPU. Syntax is Var=INP (I/O-addr). When this command is executed, the named variable contains the binary weight value (decimal format) of ***all*** bits at the designated I/O-addr. To determine the value of a specific bit, mask the variable value (use *&* operator) with binary weighted value of the specific bit. See the Digital I/O Map on page 25 and masking with **&** opertor on page 17.

```
10    OUT 195,155                          'Sets all I/O ports as inputs
20    X = INP (192)                        'Reads value of port A
30    X1 = X & 16                          'mask  to read pin 2 on port A
40    IF X1 = 16 THEN PRINT "Bit 2 is high"
50    IF X1 = 0   THEN PRINT "Bit 2 is low"
```

INPUT: Inputs data from channel 1 at J3 (console channel):

```
10 INPUT A          RUN
20 INPUT B6         ? 45        (If user types 45 at console, BASIC assigns A=45)
                    ? 237.2     (If user types 237.2, B6=237.2 and continues execution)
```

At line 10, BASIC responds with a question mark. When a user types a number, it is assigned to variable A when <Enter> is pressed. In this example, BASIC prompts with another question mark. The user then types in the value for variable B6. Refer to the STRING Section below for rules governing the use of **INPUT** and **INPUT LINE** when defining string variables.

OUT I/O-addr, data: Use this function to output data directly to an I/O port address including the I/O registers in the CPU. Syntax is OUT I/O-addr, data.

PEEK and POKE: Individual RAM memory addresses can be written with **POKE addr, data byte** and both RAM and EPROM memory can be read with **PRINT PEEK (addr)** or **var = PEEK (addr)**.

PRINT/LPRINT: The **PRINT** and **LPRINT** statements output information to the console channel (and the J6 printer port) including the value of variables and expressions, data at specified memory locations, literal strings and string variables. Quoted strings are output as they appear in the **PRINT/LPRINT** statements. Numbers are printed in decimal format. Positive numbers are preceded by a space and negative numbers are preceded by a minus (-) sign. There is a trailing space for all numbers. A semi-colon (;) at the end of a **PRINT/LPRINT** statement suppresses the usual return/line feed.

```
10 PRINT "THIS IS A STRING"          >RUN
20 A = 10                            THIS IS A STRING
30 B = 20                            10 PLUS 20 = 30
40 PRINT "10 PLUS 20 =";A+B
```

EXTERNAL INTERRUPT SIGNALS

INTERRUPT SIGNALS: The Vitrax Plus Controller provides four external interrupts: NMI and INT0, 1 and 2. NMI is an low-going edge triggered Non-Maskable Interrupt. With no way to disable this interrupt, careful judgment should be used before deciding to employ NMI. When invoked, NMI causes the program to vector to a pre-assigned address to execute the user's NMI interrupt service routine. The incoming NMI signal should be conditioned to produce a single clean low-going edge together with some external means to prevent the signal from interrupting itself until the service routine is complete. If interrupt inputs overrun the routine, it is likely to produce a fatal stack overflow error. NMI is best used to detect a serious or catastrophic condition that leads to system shutdown.
INT0, INT1 and INT2 are low-level triggered. Triggering signals to these inputs should be clean and momentary to avoid multiple interrupts. If held low, these interrupts halt the processor.

INTERRUPT SOFTWARE: Vitrax BASIC includes six commands to service four external interrupt signals. Two global commands turn interrupt inputs on (**ENA**able) and off (**DIS**able) except for NMI which, by design, is non-maskable. Additionally, I/O Register ITC (address 34H) operates in conjunction with the global commands to complete the enabling/disabling process for INT0, 1 & 2

ITC Register						**Address: 34 hex 52 dec**		
BIT	7	6	5	4	3	2	1	0
NAME	TRAP	UFO	-	-	-	INT2	INT1	INT0
RESET	0	0	1	1	1	0	0	1
R/W	R/W	R	-	-	-	R/W	R/W	R/W

Use this table for ITC Register code for all enable/disable combinations of INT0, INT1 and INT2. Use **OUT** to set the bit patterns and **INP** to read them.

INT2	INT1	INT0	DATA (ITC REGISTER)
Dis	Dis	Dis	0 or 56
Dis	Dis	Ena	1 57
Dis	Ena	Dis	2 58
Dis	Ena	Ena	3 59
Ena	Dis	Dis	4 60
Ena	Dis	Ena	5 61
Ena	Ena	Dis	6 62
Ena	Ena	Ena	7 63

Examples:
OUT 52, 60 (set INT2 on and INT1, INT0 off)
PRINT INP(52) (Reads the status of INT0,1,2)

At the end of each BASIC command, BASIC checks for interrupt flags from INT0, INT1, INT2 and NMI. When an interrupt signal is sensed, the program vectors to the line number in the associated **OI** (On Interrupt) command. The BASIC code at the vectored line number should begin the interrupt service routine. Implementation follows the familiar GOSUB structure ending with RETURN. The **OI** command format is:

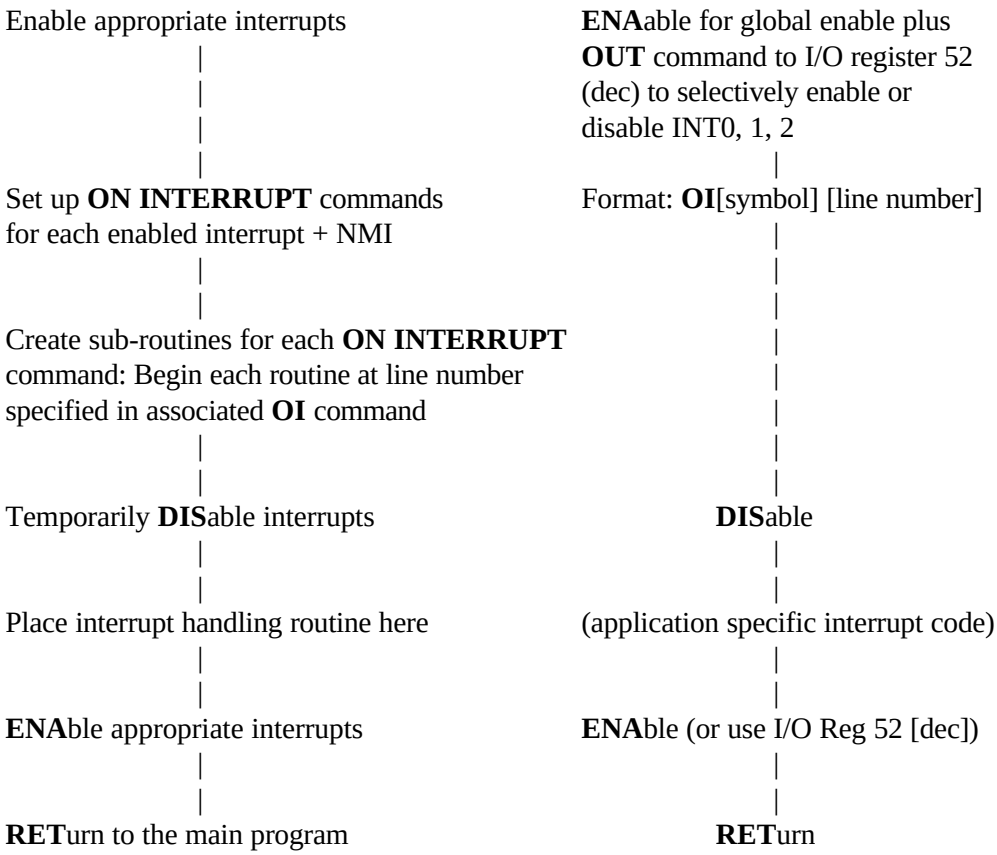
OI interrupt symbol line number Example: **OI2 150**
(On Interrupt 2 --> line 150)

N = NMI
0 = INT0
1 = INT1
2 = INT2

initial line number of the interrupt service routine

|__ On Interrupt Command

Except for NMI, interrupts must be **ENA**bled to be detected. On RESET, INT0 is enabled; Interrupts 1 and 2 are disabled. Interrupt handlers can be written in BASIC as subroutines, beginning with a **DIS**able command to block additional signals while the interrupt is being serviced. Next should be the specific application code to properly service the interrupt. Finally, the routine should conclude with an **ENA**able and **RET**urn command. Since NMI is non-maskable, it is good practice to include an **OIN** command and its subroutine. e.g. **ENA**ble followed by a **RET**urn to allow the program to continue. Here is a general procedure using On Interrupt Commands:



TEST PROGRAM FOR INTERRUPTS: The following program can determine if interrupts are functional. Quickly brush a grounded test lead across an interrupt input pad on the board to emulate a **single** low pulse. Observe if the appropriate message appears on the console as shown in lines 210, 260, 310 and 360.

PROGRAM TO TEST INTERRUPT DETECTION CAPABILITY [INTR1]

```
 10 OIN 200                'Set up GOSUB-type calls for
 20 OI0 250                'each interrupt
 30 OI1 300
 40 OI2 350
 50 OUT 52,63                          'Enable INT0,1,2 via I/O Reg 52
 60 ENA                                'Globally Enable all interrupts
 70 PRI "WAITING FOR INTERRUPTS ....."
 80 GOTO 70                            'Two line main program (loop)
200 DIS                        'GOSUB for NMI; Dis INT0,1,2
210 PRI:PRI"NMI RECEIVED"              'GOSUB for NMI; NMI detected
220 FOR I=1TO2000:NEXT I               'Pause 3 sec for NMI acknowledge
230 ENA                                'Re-enable interrupts
240 RET                                'Return to main program
```

```
250 DIS                          'GOSUB for INT0; Disable INT0,1,2
260 PRI:PRI"INT0 RECEIVED"             'INT0 detected
270 FOR I=1TO2000:NEXT I               'Pause 3 sec for INT0 acknowledge
280 ENA                                'Re-enable interrupts
290 RET                                'Return to main program
300 DIS                          'GOSUB for INT1; Disable INT0,1,2
310 PRI:PRI"INT1 RECEIVED"             'INT1 detected
320 FOR I=1TO2000:NEXT I               'Pause 3 sec for INT1 acknowledge
330 ENA                                'Re-enable interrupts
340 RET                                'Return to main program
350 DIS                          'GOSUB for INT2; Disable INT0,1,2
360 PRI:PRI"INT2 RECEIVED"             'INT2 detected
370 FOR I=1TO2000:NEXT I               'Pause 3 sec for INT2 acknowledge
380 ENA                                'Re-enable interrupts
390 RET                                'Return to main program
```

LOGICAL & COMPARISON OPERATORS

COMPARISON OPERATORS: A set of the relational operators provides comparisons:

=	equal to	**<=**	less than or equal to
>	greater than	**>=**	greater than or equal to
<	less than	**><**	not equal to

LOGICAL OPERATORS: BASIC contains **&** (AND), @ (OR) and **%** (XOR) to change/detect the state of specific bits within a byte, and to provide bit set/reset capability. Format: data byte operator data byte

FUNCTION	OPERATOR
AND	**&**
OR	@
XOR	**%**

The **&** (AND) operator is typically used to determine the state of a bit within a byte. When a byte is ANDed with a mask containing a '1' in a specific bit position, the bit value will appear in the result. Examples of masking and bit manipulation:

& (Bitwise AND) is typically used to determine the state of a bit within a byte. When a byte is ANDed with a mask that contains a '1' in the specific bit position, the bit value appears in the result:

Position	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	EXAMPLE									
Data Byte	-	0	0	1	1	-	-	-	A	=	1	1	1	1	0	0	1	1 = 243
Mask	-	0	1	0	1	-	-	-	B	=	0	1	0	0	1	0	1	1 = 75
Result	-	0	0	0	1	-	-	-	**A & B**	=	0	1	0	0	0	0	1	1 = 67

A '1' is produced if and only if both bits are a "1". **&** can also be used to reset a bit (bit=0) with a '0' in the mask:

@ (Bitwise OR) is typically used to set a bit (bit=1)

Position	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	EXAMPLE									
Data Byte	-	0	0	1	1	-	-	-	A	=	1	1	1	1	0	0	1	1 = 243
Mask	-	0	1	0	1	-	-	-	B	=	0	1	0	0	1	0	1	1 = 75
Result	-	0	1	1	1	-	-	-	**A @ B**	=	1	1	1	1	1	0	1	1 = 251

A '1' is produced if one or both of the bits is a "1"

% (Bitwise XOR): same as **BOR** but excludes the 1 **OR** 1 = 1 to produce 1 **XOR** 1 = 0

OTHER COMMANDS

CALL: Vitrax BASIC permits the use of embedded machine code routines. Control is transferred from BASIC code to a machine language routine by a **CALL** Statement. This feature is useful to incorporate highly efficient code for time critical routines. A **CALL [address]** statement transfers control to the machine language routine at [address]. **CALL** automatically stacks the return address to BASIC and adjusts the stack pointer. Control is transferred by the interpreter through execution of an internal jump instruction. Depending on the code in the machine code routine, pointers could be modifie. In similar fashion, the values in the processor's general registers at the completion of the **CALL**ed routine could be unpredictable. under these conditions, the opening statements in the routine should stack or store data that needs to be restored. Similarly, statements at the end of the **CALL**ed routine should restore needed data, code or pointers before surrendering control to Vitrax BASIC. The final statement in the routine should be a **RET** (RETI or RETN) to return control to the BASIC statement following the **CALL**.

Machine code routines are best placed in user-protected memory space -- protected from BASIC's internal compaction routines. Refer to ***Protected User Memory Space***. earlier in this chapter:

DIMENSIONING ARRAYS: Vitrax BASIC accommodates single and double-dimensioned arrays. Arrays must be dimensioned with the **DIM** statement using the format **DIM** [array variable]([column limit, row limit]). Arrays may be dimensioned directly with numerical values or indirectly though variables. [DIM3 & DIM2]

```
10 A=5                                  >RUN
20 DIM X(A)                             X( 1 )= 2
30 FOR I=1 TO A                         X( 2 )= 4
40 X(I)=I*2                             X( 3 )= 6
50 NEXT I                               X( 4 )= 8
60 FOR J=1 TO A                         X( 5 )= 10
70 PRINT "X(";J;")=";X(J)
80 NEXT J

                                        >RUN
10  A=4:B=3                             X( 1 , 1 )= 1
20  DIM X(A,B)                          X( 1 , 2 )= 2
30  FOR I=1 TO A                        X( 1 , 3 )= 3
40  FOR J=1 TO B
50  X(I,J)=I*J                          X( 2 , 1 )= 2
60  NEXT J                              X( 2 , 2 )= 4
70  NEXT I                              X( 2 , 3 )= 6
100 FOR K=1 TO A
110 FOR L=1 TO B                        X( 3 , 1 )= 3
120 PRINT "X(";K;",";L;")=";X(K,L)      X( 3 , 2 )= 6
130 NEXT L                              X( 3 , 3 )= 9
135 PRINT
140 NEXT K                              X( 4 , 1 )= 4
150 STOP                                X( 4 , 2 )= 8
                                        X( 4 , 3 )= 12
```

Note that variable A indirectly dimensions the single-dimensional array variable X, although a numerical value can be used. The statements in lines 10 and 20 must appear **before** values are assigned to the array elements. Arrays can also be double-dimensioned as shown in the second example with a 4 x 3 array. ***Keep in mind that arrays occupy significant memory space.***

FOR/NEXT (STEP): These statements are useful to perform repetitive operations for a prescribed number of times. **STEP** is optional. If it is not included, a value of +1 is used. Starting and ending values of the **FOR/NEXT** loop are included in the **FOR** statement by a counter variable. The loop is repeated when the **NEXT** statement is executed provided the upper limit of the counter has not been reached. When the limit is reached, the program exits from the loop. BASIC

causes an error break if the variable in the **NEXT** statement is not the same variable used in the **FOR** statement. **FOR/NEXT** loops may be nested, but BASIC will report an error if the nesting level exceeds eight deep. A **FOR** loop is executed at least once, even if the initial value of the counter variable exceeds its bounds.

The following program prints odd integers less than l00.

```
10 N=l00                    :REM UPPER LIMIT
20 FOR I=I TO N STEP 2      :REM START AT I WITH STEP OF 2
30 PRINT I                  :REM PRINT A NUMBER
40 NEXT I                   :REM REPEAT (at Line 20)
```

GOSUB Subroutines: **GOSUB** instructions are useful when an operation is required at more than one place in a program. Rather than write the routine at each location, a **GOSUB** command is used to "call" a subroutine. After the subroutine executes, a **RETURN** instruction (the last instruction in the subroutine) causes the program to resume execution at the line number following the **GOSUB** instruction.

IF/THEN: This instruction allows program control to be modified by a logical test condition. The test condition follows the **IF** clause of the statement. When the test condition is true (non-zero), the **THEN** portion of the statement will be executed. When the test condition is false (zero), the **THEN** portion is ignored and execution continues at the next numbered line of the program.

```
50 IF X > J THEN GO TO l40
```

STOP: Although **STOP** is not required to end a program, it can be useful as a breakpoint for program debugging. When a **STOP** is encountered, a stop message and its line number is printed. BASIC returns to direct mode where **PRINT** statements can be used to determine variable values. Multiple **STOP** statements are allowed. By removing **STOP** statements one by one, a program can be tested in sections until debugging is completed. Program execution may be continued by a direct **CONT** command.

RUN executes the current program. Execution begins at lowest line number and proceeds sequentially by ascending line number except for branching commands, such as **GOTO** and **GOSUB**, that alter the order of execution.

STRING MANIPULATION COMMANDS

STRING ASSIGNMENT: Vitrax BASIC handles both text and numeric strings up to 200 characters long. Text strings can be converted to numeric strings and vice versa. Strings can be analyzed and manipulated with **LEN, MID$, LEFT$, RIGHT$ and INSTR**. Text strings require quotation marks when defined within a program. Numeric format must be used for calculations. (use **VAL** command). Strings can be joined with the "**+**" operator or with a sequence of **"PRINT**;" commands. String commands are listed and cross referenced at the end of this section.

ASC(str):	Returns ASCII decimal value of the first character in (str).
CHR$(x):	Returns an ASCII character by converting x (decimal) to ASCII
INPUT A$:	Inputs text string(s), no embedded spaces allowed.
INPUT LINE A$	Same as INPUT(A$) but embedded spaces allowed
INSTR(x,A$,B$)	Searches for B$ in A$; returns beginning position of B$ (or 0)
LEFT$(A$,n)	Returns a string from character position 1 to n in A$
LEN(A$)	Returns the number of characters in A$
MID$(A$,C,n)	Returns a string of n characters beginning at character position C
NUM$(x)	Converts the number x to a string with a leading and trailing space

RIGHT$(A$,n) Returns a string from character position n to the end of string A$
STRING$(n,d) Returns a strings of n ASCII characters whose decimal value is d
VAL(A$) Converts text string to numeric value
+ To concatenate (join) strings

ASC Function [ASC1]

```
10 Program to test ASC Function
20 A$="ALPHA":B$="BETA"
30 A=ASC(A$):B=ASC(B$)
40 PRINT "1st Character of A$ =";A
50 PRINT "1st Character of B$ =";B
```

>RUN
1st Character of A$ = 65
1st Character of B$ = 66

CHR$ Function [CHR1]

```
10 X$=CHR$(65):Y$=CHR$(66):Z$=CHR$(49)
20 PRINT "CHR$(65) =";X$;"  CHR$(66) =";Y$;"  CHR$(49) = ";Z$
```

>RUN
CHR$(65) =A CHR$(66) =B CHR$(49) = 1

INPUT and INPUT LINE input numbers and text strings from console channel (#1). Text strings can be upper/lower case with letters and digits intermixed. Rules for their use - and examples follow:

	NUMBER VARIABLES	QUOTATION MARKS	EMBEDDED SPACES
INPUT:	Multiple	optional	only with quotation marks
INPUT LINE:	One	treated literally	allowed

COMMAND	CONSOLE INPUT	CONSOLE OUTPUT as A$
INPUT A$	VITRAX PLUS	VITRAXPLUS
	VitraxPlus	VitraxPlus
	"Vitrax Plus"	Vitrax Plus
INPUT LINE A$	VITRAX PLUS	VITRAX PLUS
	Vitrax Plus	Vitrax Plus
	"VITRAX PLUS"	"VITRAX PLUS"

LEN, LEFT$, MID$, RIGHT$, INSTR Commands [STRING1]

```
10 PRINT "String Commands Examples: LEN, LEFT$, MID$, RIGHT$, INSTR"
30 A$="ABCDEFGHIJ":B$="HI"
50 L=LEN(A$)
60 P$=LEFT$(A$,3)
70 Q$=MID$(A$,3,4)
80 R$=RIGHT$(A$,4)
90 S=INSTR(1,A$,B$)
100 PRINT "This is the value of A$      :  ";A$
110 PRINT "Length of A$              : ";L;" characters"
120 PRINT "3 left characters of A$     :  ";P$
130 PRINT "Mid 4 characters begin at #3 :  ";Q$
140 PRINT "Right characters begin at #4 :  ";R$
150 PRINT "B$ ('HI') in A$ begins at    : ";S;"character position"
```

>RUN
Program to Test String Commands: LEN, LEFT$, MID$, RIGHT$, INSTR
This is the value of A$: ABCDEFGHIJ

Length of A$: 10 characters
3 left characters of A$: ABC
Mid 4 characters begin at #3 : CDEF
Right characters begin at #4 : DEFGHIJ
B$ ('HI') in A$ begins at : 8 character position

NUM$ and VAL: Text/Numeric Conversiona [NUMVAL1]

```
10 REM Program to test NUM$ and VAL String Functions:PRINT
20 A=123.4
30 A$=NUM$(A)
40 PRINT "Convert A (";A;") to a numeric string, A$ = ";A$
50 PRINT "Print the length of A$ (";A$;")          =";LEN(A$);" characters"
60 PRINT "Note that A$ contains";LEN(A$);" characters, not";LEN(A$)-2;" which"
70 PRINT "   indicates a leading and trailing zero."
90 PRINT "Using ASC to confirm 1st & last characters are ASCII 32 (spaces):"
100 PRINT "The leading character is an ASCII (decimal)";ASC(LEFT$(A$,1))
110 PRINT "The last character is an ASCII (decimal)";ASC(RIGHT$(A$,1))
120 PRINT:PRINT "Convert A$ to numeric value, B:"
130 B=VAL(A$)
140 PRINT "Print values of B and B x 2 (to prove B is numeric):";B;" and ";B*2
```

>RUN
Convert A (123.4) to a numeric string, A$ = 123.4
Print the length of A$ (123.4) = 7 characters
Note that A$ contains 7 characters, not 5 which
indicates a leading and trailing zero.
Using ASC to confirm 1st & last characters are ASCII 32 (spaces):
The leading character is an ASCII (decimal) 32
The last character is an ASCII (decimal) 32
Convert A$ to numeric value, B:
Print values of B and B x 2 (to prove B is numeric): 123.4 and 246.8

STRING$ Function to Fill a String [STRING4]

```
10 REM Program to test STRING$ Command
20 N2=2:N5=5:N8=8:D1=65:D2=66:D3=49
30 A$=STRING$(N2,D1)
40 B$=STRING$(N5,D2)
50 C$=STRING$(N8,D3)
60 PRINT:PRINT "Print a String of 2 'A's : ";A$
70 PRINT "Print a string of 5 'B's : ";B$
80 PRINT "print a string of 8 '1's : ";C$
```

>RUN
Print a String of 2 'A's : AA
Print a string of 5 'B's : BBBBB
Print a string of 8 '1's : 11111111

CROSS-REFERENCE OF STRING MANIPULATION COMMANDS

FROM	TO	USE
Text string	Numeric string	**VAL(A$)**
Numeric string	Text string	**NUM$**
ASCII Character	Decimal Number	**ASCII**
Decimal value	ASCII Character	**CHR$**
Multiple strings	Single string	**+** or **PRINT;**
Single string	Multiple strings	**LEFT$, MID$, RIGHT$**

TRIGONOMETRIC FUNCTIONS

Four trigonometric functions are available to calculate sine, cosine, tangent and arctangent values. Parameters for these functions must be supplied in radian units, but many applications prefer the use of degrees. To use degrees in these functions, multiply the value in degrees by PI/180. For example:

SIN, COS, TAN and ATN Functions [TRIG2]

```
10 A=30*PI/180:B=1.0                    >RUN
20 PRINT "SINE OF A = ";SIN(A)          SINE OF A = .499999
30 PRINT "COSINE  A = ";COS(A)          COSINE  A = .866025
40 PRINT "ARCTAN  B = ";ATN(B)*180/PI   ARCTAN  B = 44.9999
```

** DIGITAL I/O LINES **

The 24 digital I/O lines provided by an 8255A parallel peripheral interface are programmable as latched inputs or outputs. The lines are grouped as four ports, shown below as A, B, C-upper and C-lower

PORT NAME	NUMBER OF BITS	J7 CONNECTOR PIN NUMBERS							
		bit 7							bit 0
A	8	18	8*	4*	2*	6*	10*	12*	14*
B	8	36	40	44	48	50	46	42	38
C-lower	4	-	-	-	-	34	32	30*	28*
C-upper	4	20	22	24	26	-	-	-	-

***** = lines shared with printer port

CONFIGURING THE 8255A: There are 16 input/output arrangements for the three ports of the 8255A. An **OUT 195,[data byte]** command will configure the direction of the 24 digital lines as shown below:

COMMAND	PORT A	PORT B	PORT C-lower	PORT C-upper
addr,data byte	PA0-7	PB0-7	PC0-3	PC4-7
OUT 195,128	OUT	OUT	OUT	OUT
129	OUT	OUT	IN	OUT
130	OUT	IN	OUT	OUT

131	OUT	IN	IN	OUT
136	OUT	OUT	OUT	IN
137	OUT	OUT	IN	IN
138	OUT	IN	OUT	IN
139	OUT	IN	IN	IN
144	IN	OUT	OUT	OUT
145	IN	OUT	IN	OUT
146	IN	IN	OUT	OUT
147	IN	IN	IN	OUT
152	IN	OUT	OUT	IN
153	IN	OUT	IN	IN
154	IN	IN	OUT	IN
155	IN	IN	IN	IN

PROGRAMMING DIGITAL I/O LINES: Each port (A, B and C) has a single I/O address. with eight I/O lines per port. Each bit in the port corresponds to an individual I/O line. So, it is possible, in binary fashion, to manipulate each line or combination of lines. For example, assume port B is configured as an output. It terminates at edge connector J7, pins 36-50:

J7 CONNECTOR PIN #	50	48	46	44	42	40	38	36	- - -
U15 SIGNAL NAME	PB3	PB4	PB2	PB5	PB1	PB6	PB0	PB7	
BIT VALUE FOR PIN HIGH	8	16	4	32	2	64	1	128	

EXAMPLES (using the above diagram):

To bring all lines to a low (logic '0') state: **OUT 193,0**
To bring all lines to a high (logic '1') state: **OUT 193,255**
To assert I/O pin 38 high (all others low): **OUT 193,1**
To assert I/O pin 50 high (all others low): **OUT 193,8**
To assert both I/O pin 38 & 50 high (all others low): **OUT 193,9**

I/O CONNECTOR J7 PINOUT AND SIGNALS

J7 Pin #	8255A Pin #	8255A Signal	OUT Command Port,Data byte	Printer Signal	P3 Conn Pin #
2	40	PA4 *	OUT 192,16	D5	15
4	39	PA5 *	,32	D6	13
6	1	PA3 *	, 8	D4	17
8	38	PA6 *	,64	D7	11
10	2	PA2 *	, 4	D3	19
12	3	PA1 *	, 2	D2	21

14	4	PA0 *	, 1		D1	23
16	no connect	--				
18	37	PA7	,128			
20	10	PC7	OUT 194,128			
22	11	PC6	,64			
24	12	PC5	,32			
26	13	PC4 *	,16		STB/	25
28	14	PC0 *	, 1		ACK/	7
30	15	PC1 *	, 2		BSY	5
32	16	PC2	, 4			
34	17	PC3	, 8			
36	25	PB7	OUT 193,128			
38	18	PB0	, 1			
40	24	PB6	,64			
42	19	PB1	, 2			
44	23	PB5	,32			
46	20	PB2	, 4			
48	22	PB4	,16			
50	21	PB3	, 8			
odd numbered pins are grounded					Gnd 14-26	
* denotes 8255A pins shared with printer connector						

DIGITAL I/O MAPPING: A logical-to-physical map will result in an organized wiring approach and an easier programming effort. It should begin with the port assignments and pin numbers of the J7 I/O Connector followed by each I/O port. For example, Port A controls I/O pins 2, 4, 6, 8, 10, 12, 14 and 16. Next, the map should include the binary value that is necessary to assert each pin high. For example, for Port A I/O pin 6, the necessary bit value is 8. Finally, for path continuity, signal names and pin numbers at the 8255A device (U15) should be added that correspond to each I/O pin. For example:

DIGITAL I/O MAP

	PORT B								**PORT C**-lwr				**PORT C**-upr				**PORT A**								
	ADDRESS =193								**ADDR=194**				**ADDR=194**				**ADDRESS = 192**								
J7 Pin #	50	48	46	44	42	40	38	36	34	32	30*	28*	26*	24		20	18	16	14*	12*	10*	8*	6*	4*	2*
Bit Value	8	16	4	32	2	64	1	128	8	4	2	1	16	32	64	**	128	N	1	2	4	64	8	32	16
U15 Pin #	21	22	20	23	19	24	18	25	17	16	15	14	13	12	11	10	37	N	4	3	2	38	1	39	40
U15 Name	PB	PB	PB2	PB	PB	PB	PB	PB	PC	PC	PC1	PC	PC	PC	PC	PC	PA	-	PA	PA	PA	PA	PA	PA	PA

* denotes 8255A pins shared with printer connector

** SERIAL I/O CHANNELS **

The Vitrax PLUS Controller provides versatile serial communication capability with two independently programmable full duplex channels. It can communicate directly with a wide variety of DCE and DTE RS-232 devices. Both channels are configured and controlled by I/O registers in the main processor and supported by software in the BASIC ROM.

GENERAL INFORMATION: Transmitting and receiving data involve writing and reading I/O registers and validating status bits. Both operations are double buffered for data integrity. In the receive mode, a data byte from the serial bit stream is collected in a primary I/O register; then a status flag is set and the byte is transferred to a holding register until the controller's data bus is available and the program calls for data. Meanwhile, the next byte can be assembled in the primary register. In the transmit mode, the data byte is transparently serialized and formatted; a status flag is set when the byte is shifted out through a second I/O register to the connected equipment.

Before any transmission can occur, serial parameters must be set including baud rate, data and stop bits, parity and flow control. ViTRAX BASIC simplifies most of the configuration and provides concise commands for receiving and transmitting data.

USING THE PRIMARY (CONSOLE) SERIAL CHANNEL

The main serial channel, #1, at connector J3 serves as the primary console interface. It is designed for a three-wire cable arrangement (Rx, Tx and ground) to connect to a serial port on a personal computer or CRT terminal during program development. After power-on or reset, Vitrax BASIC matches the console's baud rate when <Enter> on the keyboard's terminal is pressed three times. Three configuration parameters are fixed by ROM software: no parity, 7 stop bits and 1 stop bit (N,7,1). During program development, input is received from the console's keyboard in full duplex mode. Output to the programming console also occurs when PRINT and LIST commands are executed.

When program development is complete and application code is in EPROM, this channel can be assigned to other uses. The AutoStart mode, with application code in EPROM (U7), automatically defaults to a 9600, N, 7, 1 initialization. Configuration, status and data flow are contolled by I/O registers CNTLA, CNTLB and STAT at I/O addresses 01, 03, 05. These registers can be used to reprogram the configuration parameters.

Data is be transmitted with an **OUT 7, databyte** or **PRINT** command and received with an **INP 9** function. For your reference, outgoing data is written to the TDR register at 07 and doubly buffered in TSR . Incoming data is similarly handled. It is buffered in RSR and fed to RDR at 09 for processing. Refer to the Supplement Section or the manual described in the Machine Code Section.

USING THE SECONDARY SERIAL CHANNEL

The secondary channel (#0) can be programmed in different ways depending on the connected equipment and the application. This channel uses a register configuration similar to Channel #1 while supporting Rx, Tx, RTS, CTS and DCDO signals. Software support exists in the BASIC ROM to configure transmission parameters and receive data automatically into a predefined buffer. A FIFO algorithm is used to retrieve the buffered data. The CPU I/O registers that manage the serial channels are summarized in the 'Reference Material' section along with the corresponding hardware information.

HARDWARE ARRANGEMENT: Interpretation of RS-232 communication standards can often be ambiguous. Pin definitions depend on whether equipment is configured as Data Terminal Equipment (DTE) or Data Communication (DCE). Most of the confusion involves the choice of pins for Receive (Rx) and transmit (Tx). Since Rx and Tx signals may need to be transposed, jumper block W8 is provided. If data is received on pin 2 and transmitted on pin 3, install both jumper blocks on W8 in the ***horizontal***

position. Conversely, if incoming data is received on pin 3 and outgoing data transmitted on pin 2, install both jumper blocks on W8 in the ***vertical*** position.

INCOMING PIN NUMBER	OUTGOING PIN NUMBER	W8 JUMPER BLOCK POSITION
2	3	Both Horizontal
3	2	Both vertical

TRANSMIT PATH:						
DATA BUS	**TDR** Register Addr 06	**TSR** Register	**U1-TXA0** Pin 45	**U13** pin 6 to 7	**Jmpr W-8** pin 1 to 3	**DB9-3 or DB9-2** (if W8 1 to 3) (if W8 1 to 2)

RECEIVE PATH:						
DB9-3 or DB9-2 (if W8 3 to 4) (if W8 2 to 4)	**Jmpr W-8**	**U12** pin 4 to 6	**U1-RXA0** Pin 46	**RSR** Register Addr 08	**RDR** Register	**DATA BUS**

For secure Operation, DCD0 should be grounded and RTS bit 4 (CNTLAO at I/O address 00) reset to 0. See Jumper Table for W9 setting and pc board trace at W9.

CONFIGURATION: Before a serial channel can be put into operation, its transmission parameters must be defined. Memory locations have been allocated and reserved for baud rate, number of data and stop bits, parity/no-parity, parity type and transmission mode. To configure this channel, simply **POKE** the appropriate data byte into the memory location reserved for each parameter. The following table specifies the location for each communication parameter and the data associated with it: Example: to set the baud rate to 9600, use the command **POKE 65409,2.** Refer also to the programming example just before the Summary Section.

CONFIGURATION PARAMETER	MEMORY LOCATION Decimal	Hex	DATA
Baud Rate	65409	FF81	**1**=19200 **2**-9600 **3**=4800 **4**=2400 **5**=1200 **6**=600 **7**=300
Data Bits	65410	FF82	**7**=7 data bits; **8**=8 data bits
Parity Enable	65411	FF83	**0** for no parity; **1** to enable parity
Parity Type	65412	FF84	**0**=odd parity; **1**= even parity
Stop Bits	65413	FF85	**1**=1 stop bit; **2**=two stop bits
Protocol	65414	FF86	**0** for X-0n/X-off; **1** for RTS handshaking

If you plan to transfer ASCII data, the use of X-ON/X-OFF protocol may be more helpful. When binary data is involved, or hardware control via RTS/CTS is employed, select "**1**" as the transfer format.

DATA BUFFER and its SETUP: Vitrax BASIC contains routines to configure and manage data flow through this channel. This routine simplifies use of the second serial channel (0) to receive data... Once the configuration parameters are stored, this routine, CALLable from BASIC, establishes a 128-byte FIFO buffer to receive data. Its operation is interrupt-driven to automatically receive data in background mode and place it in the buffer. It also organizes the buffer for easy access by BASIC while optimizing the space for new data. CALL 8208

BUFFER STATUS BYTE: Memory address 65408 is reserved as a status byte to indicate when data is present in the buffer. When data exists, the flag address contains 85 decimal (55 hex) - zero when empty. BASIC can use its **PEEK** function to determine if data is in the buffer: LET A=PEEK(65408).

BUFFER ACCESS ADDRESS: The first buffer address (65280) is also the transfer address for BASIC to obtain each byte of incoming data. For simplified operation, BASIC need **PEEK** at only this address to fetch the oldest byte from the buffer: LET B=PEEK(65280).

BUFFER MANAGER: The buffer manager is a CALLable routine that shifts buffered data toward the exit location (65280) after BASIC fetches a character. After taking each byte, BASIC should CALL the buffer manager to place the next byte at the transfer address and automatically shift each byte in the buffer by one address toward the access address. It should be called each time a character is removed from the access location: CALL 8408

This routine also initiates an X-ON or RTS (high) signal when there is room in the buffer for incoming data. If X-ON/OFF protocol is selected to control data flow, an X-OFF character is transmitted when the buffer is filled to within 10 characters of the top. Ten additional characters can be received, but additional characters are lost. X-ON is issued when the buffer has room for at least 20 characters. When RTS is selected, the RTS signal (J4-7) drops to a low signal state when the buffer is within 10 characters from the top. Ten additional characters can be received before the buffer is full, but additional characters are lost. RTS is raised high when the buffer has room for 20 characters.

TRANSMITTING THROUGH THE SECOND SERIAL CHANNEL: Up to this point most of the discussion has centered around receiving data. The configuration of the RS-232 port serves for both reception and transmission. In its simplest form, transmission can be performed with a one-line BASIC routine: **OUT 6, data byte**. There are three register bits that can affect data transmission:

* **The Transmit Enable flag (TE)**: Bit 5 in Register 0 must be set=1 to enable transmission. The Port Setup routine automatically enables this bit.

* **The Transmit Data Register Empty (TDRE) flag:** Bit 1 in Register 4. When TRDE=1, the Transmit Data Register (6) is empty and the next data byte can be written to it. TDRE is momentarily cleared (0) while the byte in TDR is moved to TSR for output; and then set active to 1 again.

* **The Clear-To-Send flag (CTS)**: Bit 5 in Register 2. When read, the CTS bit reflects the state of the CTS signal at connector J4-8. The incoming signal is inverted by U12 before appearing in Register 2 bit 5 as the CTS status bit. For transmission to occur, the CTS signal must be high. The CTS flag is read as 0. When the external CTS signal is low (not clear to send), the CTS bit is read as 1 causing the TDRE bit to be held at 0 which halts transmission. (This register bit is important only when CTS handshaking is used. Otherwise, resistor pullup R9 at U12-10 permanently places CTS0 in the active state. Be certain to remove R9 when using hardware handshaking signals).

The TDRE flag can be polled by:

```
10 A=INP(4)              'read Control Reg 4
20 B=A & 2               'check bit 2 (TDRE)
30 IF B=O GOTO 10        'loop back if bit 2 low
40 OUT 6, data byte      'xmit data byte
50 GOTO 10               'go back for next byte
```

PROGRAMMING EXAMPLE: The following example uses BASIC code to configure the second serial channel and to receive data using the data buffer and manager. The example assumes that a CRT terminal, connected to J4, transmits keystrokes for display on the programming console attached at J3: [RS232-V2]

```
10 POKE 65409,2          Set baud rate to 9600
20 POKE 65410,8          Set data bits to 8
30 POKE 65411,0          No parity
40 POKE 65413,1          1 stop bit
```

```
 50 POKE 65414,0                          Use X-on/X-off protocol
 70 CALL 8208                             Create incoming buffer
 80 LET A=PEEK(65408)                     See if data is received by polling Status Byte
 90 IF A=0 THEN GOTO 80                   Wait on received data
100 LET B=PEEK(65280)                     Have data - fetch character into B
110 CALL 8408                             Shift all data down in buffer by one location
120 OUT 7,B                               Output character to console
125 IF B=13 THEN OUT 7,10                 Output LF after CR to console via J3 - Chnl #1
128 IF B=13 THEN GOSUB 200                On CR branch to send confirming message
130 GOTO 80                               Get more data
200 OUT 6,79:OUT 6,46:OUT 6,75:OUT 6,46   Send "O.K." message to terminal at J4
210 OUT 6,13                              Xmit CR to sending terminal at J4
220 RETURN                                Return to get more data
```

Lines 125, 128 and 200-220 are embellishments to the fundamental code. Line 125 sends a line feed after a carriage return to compensate for the typical configuration of the programming console (CR only vs CRLF). Delete this line if the received messages are double spaced. Line 128 looks for an incoming CR on the console and, when found, jumps to the GOSUB routine at lines 200-220 to create and transmit an "O.K" message back to the sending terminal.

SUMMARY: Here is a summary of the procedure to configure and use the second serial channel:

A 32Kx8 Static RAM is required for this application in socket Z8 along with the use of the RAM space FF00 to FF89.

1. Connect a cable between the external serial equipment and connector J4. Example: a 3-wire cable supporting Tx, Rx and GND (J4-2, 3 & 5).

2. Place two jumper blocks on W8 for proper RX and Tx arrangement. Install *both* blocks in either the horizontal or vertical position. Refer to the Jumper Table and Schematic Diagram.

3. For stability, DCD0 is pulled low by a +5 volt trace to the inverted DCD0 signal at W9. To use DCD0, cut the trace between the pins of W9 and place jumper blocks on the appropriate pins at W9.

4. When the CTS0 signal is *not* used in an application, it is normalized by resistor R9 that pulls the inverted CTS0 signal high at U12-10. ***When using CTS0, remove this resistor (R9)***.

5. Set baud rate, parity, data and stop bits, and transfer format using **POKE** and the reserved memory locations described under Configuration above - or use I/O registers 0, 2 and 4 described below.

6. Write code to handle data transmission: Set up buffer, monitor Status Byte, fetch data at Access Location, shift buffer data by CALLing Data Manager. Transmit Data with **OUT 6, databyte** command. See software support section.

REFERENCE MATERIAL

OVERVIEW OF THE CONTROL REGISTERS: Both channels have similar configuration capability. Hardware support is provided for RTS, CTS and DCD for channel #0. Three registers are provided to configure each channel. Two additional register pairs, TSR and RSR, handle the incoming and outgoing data for each channel.

	---I/O ADDRESS---	
REGISTER	CHNL #0	CHNL #1
STAT	04	05
CNTLA	00	01
CNTLB	02	03
TDR	06	07
TSR	-	-
RDR	08	09
RSR	-	-

REGISTER SETTINGS - SECONDARY CHANNEL: For most applications, some simplifying assumptions can be made about the software settings for the serial interface. For complete register information, refer to the Supplement Manual -- or to the optional Machine Code Programmer's Manual.

STAT0 Register						**Address: 04 hex 04 dec**		
BIT	7	6	5	4	3	2	1	0
NAME	RDRF	OVRN	PE	FE	RIE	DCD0	TDRE	TIE
RESET	0	0	0	0	0	-	-	0
R/W	R	R	R	R	R/W	R	R	R/W

SUGGESTED SETTING - - - - 0 - - 0 = **0**

CNTLA0 Register						**Address: 00 hex 00 dec**		
BIT	7	6	5	4	3	2	1	0
NAME	MPE	RE	TE	RTS0	MPBR/EFR	MOD2	MOD1	MOD0
RESET	0	0	0	1	-	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SUGGESTED SETTING 0 1 1 0 0 1 0 0 **100** for 8 data bits
(1 Stop bit/no parity) 0 0 0 **96** for 7 data bits

CNTLB0 Register						**Address: 02 hex 02 dec**		
BIT	7	6	5	4	3	2	1	0
NAME	MPBT	MP	CTS/PS	PEO	DR	SS2	SS1	SS0
RESET	0	-	0	0	1	1	1	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SUGGESTED SETTING 0 0 0 0 1 0 0 0 = **8** 9600 Baud
0 0 1 = **9** 4800 Baud
0 1 0 = **10** 2400 Baud
0 1 1 = **11** 1200 Baud
1 0 0 = **12** 600 Baud
1 0 1 = **13** 300 Baud

HIGHER BAUD RATES: CNTLB0 at ADDR 02, shown above, can be reprogrammed for higher baud rates. When DR, bit 3 = 0, baud rates from 600 to 38400 can be used. The following tables lists the CNTLB0 values for all baud rate settings. Alternative values exists in the center portion of the table:

BAUD RATE	CNTLB0
38400	0
19200	1
9600	2 or 8
4800	3 or 9
2400	4 or 10

1200	5 or 11
600	6 or 12
300	13
150	14

For parity error detection, check Bit 5 in STAT0 at I/O address 04 for "1". To reset the bit, write 0 to Bit 3 in CNTLA0 (I/O Addr 00) masking other bits to preserve their values. Reminder: when using parity, it is necessary to:

Enable Parity	Bit 1=1 CNTLA0	I/O ADDR	00
Poll Parity Error Flag	Bit 5=1 STAT0		04
Reset Parity Flag	Bit 3=0 CNTLA0		00

(Mask other Bits When Writing to These I/O Registers)

** MACHINE CODE PROGRAMMING **

The 64180 microprocessor used in the **Vitrax PLUS** Controller contains a full set of machine code instructions that are a super set of the Z-80 instructions. The BASIC Interpreter uses these instructions to execute user written application programs. Most users prefer that BASIC transform their program logic to the required machine code. There may be times, however, when a short program in machine language is desirable or necessary. Usually speed of execution is the issue.

The use of assembly level and machine code programming involve major considerations on the part of the user. Programming in assembly level mnemonics often requires relatively expensive and complex support tools. While this type of support is available, its cost, familiarization burden and time consuming use may rule out total assembly level language programs where money and time considerations are important. Fortunately, the popularity of the Z-80 has been responsible for an abundance of code that is readily adaptable to the 64180.

Short machine code programs are another story. These programs - brief hand coded subroutines - can fit neatly into the **Vitrax PLUS** Controller concept using BASIC **CALL** statements to incorporate them into the main BASIC program. Machine code can be POKEd into memory or READ in from DATA statements. Programmers who have achieved reasonable fluency in assembly level and machine code programming may wish to consider using these features of the BASIC Interpreter. If your experience involves only high level languages, the choice of machine coded routines may be marginal. To support machine code subroutines, **CALL** and **POKE** support is built into the **Vitrax PLUS** Controller.

Two data books may help you to expand the capability of the Controller in this area: A hardware oriented user's manual and a programmer's software manual. They contain detailed summaries of the 64180/Z80 instruction set emphasizing the additional commands accommodated by the 64180. Also included are pinouts, signal and status line definitions , register descriptions, clock cycles per instruction, and timing diagrams. All internal registers are discussed including configuration options for I/O registers. Excellent documentation for experienced programmers.

The current hardware and software manuals are available as an accessory package from your Controller dealer. Ask for **Machine Code Manuals for the 64180**.

** VITRAX SYSTEM MONITOR **

This section describes the operation of the Vitrax System Monitor (VSM) contained in the Vitrax BASIC ROM. VSM can help you develop callable machine code software to speed up critical subroutines and to condense system code. It has the ability to inspect and change memory., to set and execute to breakpoints, and to display CPU registers. VSM can also generate relocatable precise delay routines and test the RAM memory on your Vitrax PLUS Controller. Four basic functions are incorporated:

1. **DEBUG/EDIT OPERATIONS**: Machine code development and debug tools
2. **TIMER-COUNTERS**: Use of timer-counters as interval timers
3. **RAM MEMORY DIAGNOSTICS**: RAM memory diagnostics
4. **HEX / DECIMAL CONVERSIONS**: Decimal/Hex & Hex/Decimal conversion routines

To use the Vitrax System Monitor, connect a CRT terminal or computer with serial port and communication software to the Controller at J3. Boot the Vitrax PLUS Controller in the usual manner using three or four <Enter> keystrokes to set the baud rate and generate the BASIC sign-on message.

DEBUG / EDIT OPERATION: At the system prompt, type '**CALL 10700 <Enter>**' to access the VSM main menu. Type the code letter that corresponds to the desired function.

SUMMARY OF DEBUG/EDIT COMMANDS

D Display
M Modify
B Block Move
S Set Breakpoint
R Remove Breakpoint
E Execute a program
X Examine CPU Registers
V Verify two blocks of memory
Q Quit DEBUG Operation

DISPLAY: The Display function provides a Hex/Ascii block display of memory contents. When **D** is entered, the VSM prompts for the starting display address. Use four-digit hex format. Respond either **N** or **Y** about modifying the contents:

N: The monitor asks for the block size to display. <Enter>, without a size input, continually displays and scrolls memory contents until a keystroke is entered. Each byte is displayed as two hex digits, grouped 16 per line, followed by the ASCII values - or a dot for non-displayable characters.

Y: The monitor displays the selected address, current contents and a message "*Enter two digit hex value followed by return - space to leave unchanged*" followed by "*New Data*:". Type the new data (hex) then <Enter> - or hit <space> to leave data unchanged. The monitor prompts for the next address ... and the display-modify sequence begins again. For the menu press <esc>.

MODIFY: If **M** is entered, the monitor asks for the starting hex address to begin modifications and type of input. Response with **A** (Ascii) or **H** (hex).

A: Ascii characters from the keyboard are placed in memory, location by location, until the <Esc> key is struck. A null (00 hex) is added to the next location. The reason: if you are entering text to be display on the screen, you can write a simple routine to point at the first character and output continually until a null is detected. If a null is not wanted, it can be easily removed with the Display-with-Modify command.

H: Enter two-digit hex values followed by a space or <Enter> to complete the modification and advance to the next address. This mode is useful when keying in machine code. To end the sequence and return to the main menu, press <Esc>.

BLOCK MOVE: This routine moves (actually *copies*) data from one memory area to another. The original block of data remains intact at the original address. After **B** is selected, enter the starting hex address of the block in RAM or EPROM memory (four digits) and <enter> - followed by the destination address in either RAM (or EPROM memory) as four hex digits and <enter> - and then the number of bytes to be moved - then <Enter>. This routine can transfer code for EPROM to RAM for debugging. With the hardware capability of the Controller, this command can be used to program code into EPROM at U7. Two prerequisites are necessary: Connect the Vpp Power supply to connector J1; and reprogram the CPU Wait State Register with the BASIC command **OUT 50,240** before entering the Block Move Command. When the block move is complete, exit the monitor and return the Wait State Register to its original state with the command **OUT 50,48**.

BREAKPOINT: Type the address where a break in program execution should occur. When debugging machine code, it is helpful to test in small manageable "chunks" by executing code to a selected point where parameters can be examined.

To set a breakpoint, type **S**. The program in memory executes to a specified point and then returns to the monitor. The contents of memory and the processor's registers are displayed so that you can compare these values to your expectations. Type **R** to remove the breakpoint. Setting a breakpoint automatically clears a previously set breakpoint.

Important: The specified breakpoint address is changed to a three-byte instruction that jumps back to a special monitor routine. If the program should crash during debugging and alter the reserved memory, unpredictable results may occur. It is good practice to record breakpoint addresses when you enter them. If disaster occurs, you can examine those locations (display command) and take corrective action.

EXECUTE: You can execute a program by typing **E** at the main menu. The monitor prompts for the starting hex address. If a breakpoint is detected during execution, the program halts and return control to the monitor.

EXAMINE: If **X** is entered from the main menu, the contents of the processor's internal registers as they were when the last breakpoint was detected.

VERIFY: When **V** is typed, the VSM compares two blocks of memory to verify if their contents are identical. The monitor prompts for the source address (4-digit hex), address to begin the comparison, and the number of consecutive bytes of data to compare. When this routine completes its comparison, it responds with either:

Verify Passed or
Error at [address] ...and return to the main menu

QUIT: To exit from the routines in the Monitor and return to BASIC, type **Q** at the main menu. <Enter> is not required. Control is returned to BASIC and the system prompt **>** appears.

:TIMER-COUNTERS: This routine programs the two CPU timer-counters for precise time delays. Parameters are selectable from the routine's menu. VSM compiles a relocatable subroutine, CALLable from BASIC, that occupies 56 hex location (86 decimal). Multiple subroutines can be placed in either RAM or EPROM memory using the Block Move command.

Type '**CALL 14320**' from the BASIC prompt (>) to begin this routine. VSM responds:

UNIVERSAL TIMER PROGRAM
A = TIMER 0
B = TIMER 1
YOUR SELECTION?

SELECT TIMER: Select either timer **A** or **B**.

SELECT THE TIME BASE: Select the time base using the corresponding code letter:

A 1/10 second
B 1/100 second

C 1/1000 second
D 1/10000 second

ENTER TIME BASE MULTIPLIER: Enter the number to multiply by the selected time base to achieve the precise delay interval of your choice. The number must be in decimal notation and less that 65535. The timer delay of the created subroutine time base) x (base multiplier).

Example: *SELECT TIME BASE*: B 1/100 is selected
ENTER TIME BASE MULTIPLIER: 500 multiplier is 500
Subroutine, when CALLed, will produce a 5 second delay

VSM compiles the data to produce a compact machine code routine which generates precise time delays The finished code may be relocated and combined with your BASIC program. The starting address along with the program length appears on your console at compile time. Be sure to note this address for relocation purposes.

At this point, the compiled timer program will be in contention with BASIC for memory space. Relocate the program using the Block Move command. Before moving a subroutine to EPROM, read the section on Block Move about changing the Wait State Register and connecting a Vpp supply.

The timer routines are highly accurate except for a negligible error that occurs during each reload cycle. Before using a routine in critical applications, you should confirm its accuracy. To prevent lengthy interrupts from affecting accuracy, interrupts are disabled during the timer intervals. If your program uses interrupts, be sure to re-enable them when the timer program returns control to BASIC.

RAM MEMORY DIAGNOSTICS : The Monitor contains a routine to test RAM memory for write/read operation, retention and addressibility. Type **CALL 15800** to begin the test. The routine displays the memory size and the results of a five-part test as either ***PASS*** or ***FAIL***. To repeat the test press the space bar. To end the test and return to the system prompt (**>**), press **Q** once - and then the <Enter> key three or four times to reinitialize the system and set the channel baud rate..

HEX / DECIMAL CONVERSIONS: It is often helpful when writing machine code routines that interface to BASIC to switch between hex and decimal notation. Two conversion routines are provided in VSM: Hex-to-Decimal and Decimal-to-Hex. To invoke these routines, type either of the following CALL statements at the system prompt (**>**):

Hex-to-Decimal: CALL 8960
Decimal-to-Hex: CALL 8750

When entering hex numbers, one to four digits may be entered (0 to FFFF). Decimal numbers may range from 0 to 65535. The monitor displays the corresponding conversion and prompts for 'more' (**M**) or 'quit')**Q**). To enter another number for conversion, first press **M**, <Enter> and then enter the new number. To return to the system prompt, press **Q**. <Enter> is not required.

*** PROGRAMMING & USING EPROMs ***

Central to almost every control and embedded system is the ability to store application code in non-volatile memory, typically EPROM. The Vitrax Plus Controller provides 16K bytes of EPROM space for your BASIC code and a built-in flexible EPROM programmer. A Vpp power supply of 12.5 or 21 vdc is required at Connector J1.

BASIC code that you develop on-board can be directly programmed into EPROM (U7) with the **EPROM** command which automates the process. Jumper W1 (with shorting block installed) allows EPROMs to be changed while maintaining the system's operating integrity.

PROGRAMMING SET-UP: Here are a few precautions to achieve reliable programming:

1. Provide a regulated Vpp supply of 12.5 ± 0.2 volts (21.0 ± 0.5 volts) depending on the type of EPROM to be programmed - and control the method of applying and removing the Vpp voltage to avoid spikes and overshoot.

2. Insert the EPROM to be programmed in U7 and connect the leads of the Vpp programming supply *before* writing code into RAM memory. If it becomes necessary to install or change EPROMs in U7 with Vcc power on, install jumper W1 ***temporarily*** to prevents the possible loss of code in RAM from noise being generated during the removal and insertion. ***Jumper W1 must be removed at all other times.***

3. Back up RAM code to disk before programming it into EPROM.

4. **Use only 150 ns (or 200 ns) EPROMs**, otherwise the zero-wait state performance expected by the processor may cause intermittent problems.

SUPPORTING SOFTWARE: Vitrax BASIC contains two commands to support EPROM programming: **EPROM** programs BASIC code from RAM directly to EPROM in Z7. **LOAD,** a complementary command to **EPROM**, copies the EPROM contents to RAM. Its provides a convenient method to edit or update BASIC code previously committed to EPROM. Be sure to clear RAM with **NEW** before **LOA**ding

EPROM PROGRAMMING PROCEDURE: The **Vitrax PLUS** Controller can be used to program 27128/A EPROMs at either 12.5 or 21.0 vdc depending on device type and manufacturer:

1. Start with the **+ 5 vdc** power OFF. Connect the Vpp Programming supply to the pads (J1) along the right edge of the board. Observe polarity. Vpp supply OFF.

2. Insert an erased EPROM to be programmed in Z7. Orient pin 1 upper left.

3. Turn ON the main **+5 vdc** power. Generate BASIC text in RAM.

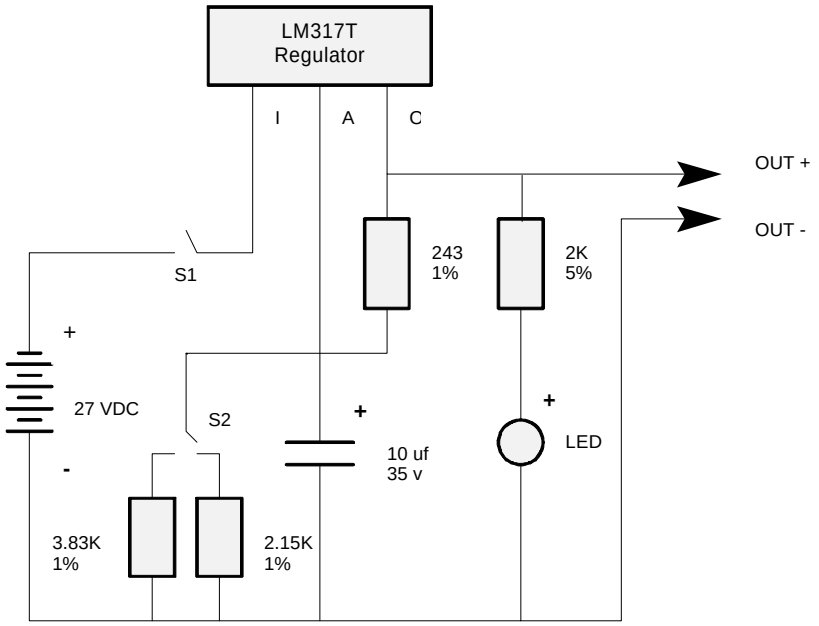
4. When code is ready for EPROM, turn ON the Vpp supply. Type **EPROM** <Enter> to program code from RAM (U8) to EPROM (U7). A "*" is displayed as each 256 bytes are programmed. The complete EPROM (16K bytes) is programmed with a byte-by-byte verify. Errors are reported. When programming is complete, the BASIC "**>**" appears.

5. Turn OFF the Vpp power supply. Procedure is complete. Confirm that the EPROM operates as expected: Press reset switch, SW1 or toggle the main power supply off - then on. The U7 EPROM code should begin **RUN**ning.

PROGRAMMING POWER SUPPLY - Vpp: EPROMs requires a Vpp supply of 12.5 or 21.0 vdc depending on the EPROM type. The power supply should be well regulated with transient-free switching characteristics. An LM317T adjustable regulator can serve as the main component. Its input side can be connected to a rectified AC source from a 110/24, 100 ma transformer -- or it can be battery-powered with three (or two) inexpensive 9-volt batteries wired in series.

A dual Vpp supply, shown below, allows both types of 27128 EPROMs to be programmed. The 10 mfd capacitor helps eliminate voltage overshoot when the supply is powered on. A red LED indicates when voltage is present at the output of the supply. If only 27128**A** EPROMs are to be programmed, the selection switch and one 1% resistor can be eliminated. Only two 9-vdc batteries (vs three) are needed.

Power Supply Schematic for Programming EPROMs



EPROM and **LOAD** commands handle data in 16K-byte increments. To program code segments with defined addresses into EPROM, follow the procedure already described with these exceptions**:**

1. Change the Wait Register: **OUT 50,240**.
2. Using the BASIC's command mode, type the following line of code:

R=aaaa:E=bbbb:FOR I=1 TO N:B=PEEK(R):POKE E,B::R=R+1:E=E+1:NEXT I (*NO* <Enter>)

Where R is the starting address (decimal) in RAM
E is the starting address (decimal) in EPROM
N is the number of bytes to be programmed

3. Turn on Vpp supply at J1 and press <Enter> to begin programming. After programming is complete (prompt (**>**) appears)), restore the settings in the Wait State register: **Type OUT 50,48**

AUTO-START MODE: When an EPROM with application code resides in U7 and main power is applied to the controller, automatic operation begins. A supporting console is *not* required. A routine in the BASIC ROM senses the presence of a EPROM (non-FF at the first U7 memory location). The process occurs automatically when main power is applied or the reset switch, SW1, is pressed. On start-up, the console serial channel (#1 at J3) defaults to 9600 baud N/7/1.

*** APPENDIX ***

ASCII CODES: DECIMAL and HEX VALUES

The following table lists ASCII Character Codes in hex and decimal format. Since BASIC handles its code in ASCII decimal format, this chart can be useful in converting one ASCII format to another.

ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC
NUL	0	0	**SYN**	16	22	**comma**	2C	44	**B**	42	66	**X**	58	88	**n**	6E	110
SOH	1	1	**ETB**	17	23	**hyphen**	2D	45	**C**	43	67	**Y**	59	89	**o**	6F	111
STX	2	2	**CAN**	18	24	**period**	2E	46	**D**	44	68	**Z**	5A	90	**p**	70	112
ETX	3	3	**EM**	19	25	**/**	2F	47	**E**	45	69	**[**	5B	91	**q**	71	113
EOT	4	4	**SUB**	1A	26	**0**	30	48	**F**	46	70	****	5C	92	**r**	72	114
ENQ	5	5	**ESC**	1B	27	**1**	31	49	**G**	47	71	**]**	5D	93	**s**	73	115
ACK	6	6	**FS**	1C	28	**2**	32	50	**H**	48	72	**^**	5E	94	**t**	74	116
BEL	7	7	**GS**	1D	29	**3**	33	51	**I**	49	73	**-**	5F	95	**u**	75	117
BS	8	8	**RS**	1E	30	**4**	34	52	**J**	4A	74	**`**	60	96	**v**	76	118
HT	9	9	**US**	1F	31	**5**	35	53	**K**	4B	75	**a**	61	97	**w**	77	119
LF	0A	10	**space**	20	32	**6**	36	54	**L**	4C	76	**b**	62	98	**x**	78	120
VT	0B	11	**!**	21	33	**7**	37	55	**M**	4D	77	**c**	63	99	**y**	79	121
FF	0C	12	**"**	22	34	**8**	38	56	**N**	4E	78	**d**	64	100	**z**	7A	122
CR	0D	13	**#**	23	35	**9**	39	57	**O**	4F	79	**e**	65	101	**{**	7B	123
SO	0E	14	**$**	24	36	**:**	3A	58	**P**	50	80	**f**	66	102	**|**	7C	124
SI	0F	15	**%**	25	37	**;**	3B	59	**Q**	51	81	**g**	67	103	**}**	7D	125
DLE	10	16	**&**	26	38	<	3C	60	**R**	52	82	**h**	68	104	**~**	7E	126
DC1	11	17	**'**	27	39	=	3D	61	**S**	53	83	**i**	69	105	**DEL**	7F	127
DC2	12	18	**(**	28	40	>	3E	62	**T**	54	84	**j**	6A	106			
DC3	13	19	**)**	29	41	**?**	3F	63	**U**	55	85	**k**	6B	107			
DC4	14	20	*****	2A	42	**@**	40	64	**V**	56	86	**l**	6C	108			
NAK	15	21	+	2B	43	**A**	41	65	**W**	57	87	**m**	6D	109			

*** MEMORY MAPS ***

DEC	HEX	MEMORY MAP OF I/O SPACE				
0	**0**	Microprocessor I/O Registers -see Supplement Section for details				
144	**90**	**ADR**	**R/W**	**A/D FUNCTION**		
		145	W	Data Format		
		145	W	Frequency Divide Ratio		
		144	W	Select Data Channel		
		145	R	Data bits 9 - 2		
		144	R	Data bits 1, 0		
160	**A0**	**REG**	**ADDRESS**		**R.T. CLOCK FUNCTION**	**ACCESS**
		0	160	A0	Control	R/W
		1	161	A1	1/10 second	R
		2	162	A2	seconds	R/W
		3	163	A3	tens seconds	R/W
		4	164	A4	minutes	R/W
		5	165	A5	tens minutes	R/W
		6	166	A6	hours	R/W
		7	167	A7	tens hours	R/W
		8	168	A8	days	R/W
		9	169	A9	tens days	R/W
		10	170	AA	months	R/W
		11	171	AB	tens month	R/W
		12	172	AC	years	R/W
		13	173	AD	tens years	R/W
		14	174	AE	day-of-week	R/W
		15	175	AF	settings	R/W
176	**80**	NOT USED				
192	**C0**	**Parallel Port:** Port A = 192 (C0)				
		Port B = 193 (C1)				
		Port C = 194 (C2)				
		Control = 195 (C3)				
208	**D0**	NOT USED				
224	**E0**	**LCD Display**: Instructions = 224 (E0)				
		Data = 225 (E1)				
240	**F0**	NOT USED				

DEC	HEX	MEMORY MAP: DIRECT MEMORY SPACE

0	**0**	BASIC INTREPRETER	
16384	**4000**	EPROM (Application Code) + EPROM Programmer	
32768	**8000**	Reserved for BASIC Initialization (695 Bytes)	
33463	**82B7**	**8K x 8 RAM**	**32K x 8 RAM**
40704	**9F00**	Stack for 8K x 8 (256 bytes)	
48896	**BF00**		Stack for 32K x 8 (256 bytes)
49152	**C000**		Protected Memory Area for CALLs, data tables and special user code (16128 bytes)
65280	**FF00**		Data buffer for 2nd serial channel (128 bytes) **65280** = data xfer address
65408	**FF80**		Data Received flag
65409	**FF81**		Baud rate setting
65410	**FF82**		Data bits setting
65411	**FF83**		Parity on/off
65412	**FF84**		Parity odd/ even
65413	**FF85**		Stop bits setting
65414	**FF86**		X-on/X-off or RTS
65415,6	**FF87,8**		Data Input pointer
65417	**FF89**		Buffer full flag
65418 **65535**	**FF8A** **FFFF**		Protected memory area for CALLs, data tables and special user codes (118 bytes)

*** COMPONENT PARTS LIST ***

INTEGRATED CIRCUITS

	U1	64180	Microprocessor
	U2, U3	LS138	(ALS138)
	U4	HC04	(ALS04)
	U5	27128	BASIC ROM
O	U7	27128-15	Application Code
	U8	6264LP-15	(62256LP-15)
	U9	LS123	
	U10	ALS02	
	U11	HC04	-5v generator
	U12	1489A	
	U13	TL084	(TL064, 4064)
	U15	8255A-5	(82C55A-5)
C	U16	58274	RT Clock Option
A	U23	7004	A/D Conv Option

Not used: U6, U14, U17 - 22

CAPACITORS

	C1, 2	20 pf	disc ceramic
	C3	1000 pf	disc ceramic
	C4	1.0 uf (0.47)	dipped tantalum
	C5	1000 pf	disc ceramic
	C6	0.1 uf	monolithic capc
	C7	100 pf	disc ceramic
	C8, 9	0.1 uf	monolithic capc
C	C10	15 pf	disc ceramic
C	C11	100 pf	disc ceramic
C	C12	5 - 30 pf	circular trimmer
	C13	2.2uF (1.0)	dipped tantalum
	C101-107	0.1 uf	monolithic capc
	C109-112	0.1 uf	monolithic capc
C	C113	0.1 uf	monolithic capc
	C120	10 uf	dipped tantalum

Not Used: C11, C108, C114-119

SOCKETS

	Z1	Socket - 64 pin	(0.070" spacing)
	Z5,7,8	Socket - 28 pin	(0.100" spacing)
	Z15	Socket - 40 pin	(0.100" spacing)
C	Z16	Socket - 16 pin	(0.100" spacing)
A	Z23	2 14-pin strips	(0.100" spacing)

DIODES

	CR1	1N914	
	CR2	1N270	(1N5817,8)
	CR3, 4	1N914	OR gate at U9
	CR5, 6, 7	1N914	-5V generator
C	CR8, 9	IN270	(1N5817,8)

MISCELLANEOUS

C	Q2	2N7000	U16 Write line
	SW1	Switch	mom push btn - n.o.
	X1	12.288 MHz crystal	U1 processor
C	X2	32.768 KHz crystal	RT Clock U16
	P/C	PC board	4-1/2" x 6-1/2"
	OAM-M	Operations Manual	
O	OAM-AS	Assembly Manual	
A	OAM-A	Oper/Assy Manual	A/D Converter option
C	OAM-C	Oper/Assy Manual	RT Clock option
L	OAM-L	Oper/Assy Manual	LCD Display option
C	BH	Battery Holder Assy	for RT Clock option

RESISTORS

L	POT1	5K (10K)	Single turn pot
A	POT2	5K (10K)	Multi turn pot
	R3, 4	10K - 1/8w	brn-blk-org
	R6	10K - 1/8w	brn-blk-org
	R9,10	1K	brn-blk-red
C	R13	10K	brn-blk-org
	RP-1, 2, 3	10K - 10 pin	9 resistor SIP

Not used: R1, R2,R5,R7,R8,R11,R12

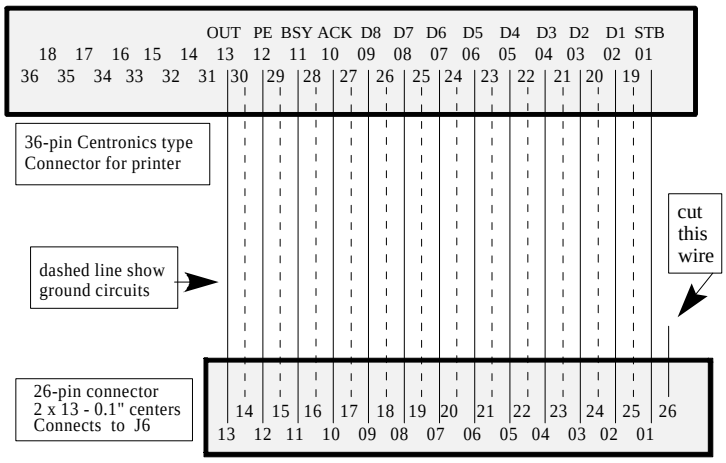
CONNECTORS, JUMPERS, HEADERS

	J1	1 x 2 pin header for Vpp vopltage input to program EPROMs
C	J2	1 x 2 pin header for battery input for RT clock option
	J3, J4	1DB9 male connector, pc mtg, for for serial I/O channels at J3 and J4
O, F	J6	26 pin (2 x 13) header for printer port
O, E	J7	50 pin (2 x 50) header for parallel port at U15
L	J8	14 pin (2 x 7) header furnished with LCD module and cable
A	J12	20 pin (2 x 10) header furnished with A/D Converter option
	J14	2-circuit, screw terminal block for Vcc power input, pc mtg
	W1	1 x 2 pin header and jumper block for EPROM change
	W4	2 x 2 pin header and jumper block for RAM selection
	W8	2 x 2 pin header and jumper blocks (2) for Rx /Tx interchange for auxiliary serial channel
	W9	2 x 2 pin header and jumper block for DCDO selection for auxiliay serial channel

EXPLANATION OF SYMBOLS IN LEFT MARGINS

A = Part is furnished in A/D Converter Module
C = Part is furnished in Real-Time Clock Module
E = Optional connector for I/O lines
F = Optional connector ;Centronics-type printer
J = Optional connector for A/D Converter inputs
L = Part is furnished with LCD module cable assy
O = Parts are optional
None = Parts furnished in Vitrax-PLUS Controller Kit

PRINTER CABLE ASSEMBLY



J6 Pin Number	01	03	05	07	09	11	13	15	17	19	21	23	25
J6 Signal Name	GND	NC	BSY	ACK	GND	D7	D6	D5	D4	D3	D2	D1	STB
U15 Pin Number	-	-	15	14	-	38	39	40	01	02	03	04	37
U15 Signal Name	-	-	PC1	PC0	-	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PA7

Cable Assembly is viewed from cable side with connector openings facing down