Flashlite

User's Manual

Proprietary Notice and Disclaimer

Unless otherwise noted, this document and the information herein disclosed are proprietary to JK Microsystems. Any person or entity to whom this document is furnished or having possession thereof, by acceptance, assumes custody thereof and agrees that the document is given in confidence and will not be copied or reproduced in whole or part except to meet the purposes for which it was delivered.

The information in this document is subject to change without notice, and should not be construed as a commitment by JK Microsystems. JK Microsystems will make every effort to inform users of substantive errors. JK Microsystems disclaims all liability for any loss or damage resulting from the use of this manual or any software described herein, including without limitation contingent, special, or incidental liability.

Flashlite is a trademark of JK Microsystems. MS-DOS is a registered trademark of Microsoft Corporation. XDOS is a copyright of HBS Corporation and JK Microsystems. All other brand and product names are trademarks or registered trademarks of their respective companies.

Printed in U.S.A.
Document Part No. 98-0001
Published November 1996

Limited Warranty

JK Microsystems warrants each Flashlite to be free from defects in material and workmanship for a period of 90 days from the date of purchase. This warranty extends only to the original purchaser and shall not apply to any unit which has been subject to misuse, neglect, accident, or abnormal conditions of operation.

JK Microsystems' obligation under this warranty is limited to repairing or replacing, at JK Microsystems' option, any unit returned to the factory within 90 days of the date of purchase, provided that JK Microsystems determines that the unit is defective and has been used in compliance with the terms of this warranty. If the failure has been caused by misuse, neglect, accident, or abnormal conditions of operation, repairs will be billed at a nominal cost.

The foregoing warranty is exclusive and in lieu of all other warranties, expressed or implied, including, but not limited to, any warranty of merchantability or fitness for any particular purpose. JK Microsystems shall not be liable for any special, incidental or consequential damages, whether in contract, tort, or otherwise.

If a failure occurs, forward the unit(s) postage paid to JK Microsystems, giving full description of the difficulty. Repair or replacement will be made at JK Microsystems' discretion, and the unit(s) returned, transportation prepaid. JK Microsystems shall assume no risk for damage in transit.

Life Support System Application Disclaimer

JK Microsystems products may not be used as critical components in life support devices or systems without the written consent of an officer of JK Microsystems. As used herein, life support systems are devices or systems which (a) are intended for surgical implant in the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided in the labeling, can reasonably be expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Table of Contents

Overview

The Flashlite single board computer is based on the NEC V-25 Plus microcomputer. The V-25 is a high-performance, 16-bit, single-chip microcomputer that is software compatible with the Intel 8086/8088 family of microprocessors. DOS compatibility allows development in a familiar environment. The high endurance flash memory eliminates the EPROM programming without worry of damaging the onboard non-volatile memory with repeated program cycles. Applications are uploaded directly into the flash disk.

Software development for the Flashlite is remarkably simple and quick. Programs are written on a PC compatible computer in the language of your choice. After your application has been compiled or assembled and linked into .EXE or .COM form, it is uploaded to the Flashlite's flash disk with your favorite telecommunications program using X-Modem protocol. The application can then be tested and debugged through the console serial port. When the application is running to your satisfaction, the startup batch file can be modified so that the application will load and execute upon reset or powerup.

Features

- 8 Mhz V-25 Plus CPU Chip (10 MHz EN Model)
- 512k Bytes System Memory
- 256k Bytes Flash Disk (512K EN Model)
- 2 Serial Ports
- 2 Fast DMA Channels
- 2 Timer Channels
- 24 Parallel I/O Lines
- 8 Analog Comparators with programmable levels
- 4 Hardware Interrupt Lines
- Priority interrupt controller with vector or register bank switching
- 5 Volt Only Operation
- Low Power Consumption
- No EPROM Programming
- DOS 3.3 compatible file system
- Utilities: Upload, Flash disk formatter, BASIC interpreter, and Line editor

Operation

The Flashlite is configured with two `disk drives' A: and B:. The A: drive is a read-only drive that contains the DOS system files and Flashlite utilities. Drive A: in combination with the BIOS occupy 128Kb of flash memory. The remaining 128Kb of flash memory is configured as the B: drive, a read-write drive available for user programs.

The first serial port is the console for the Flashlite. The port is configured for 9600 baud, 8 data bits, 1 stop bit, no parity and uses hardware handshaking. This is the primary mode of communicating with the Flashlite. DOS and the BIOS treat the console port as the logical devices STDIN and STDOUT. The second port is available for use with another device.

When power is applied to the Flashlite or when it comes out of reset, the board goes through its startup procedure then starts DOS. A simple AUTOEXEC.BAT file is executed and then the board is ready to use. The batch file performs several functions before the user is given control. The DOS search path is set, the DOS prompt is set, and finally, an attempt is made to execute a file named STARTUP on the B: drive. STARTUP may be a .COM, .EXE or .BAT file. This provides a convenient way for custom applications to execute immediately after initialization of the Flashlite. When an application is finished, either rename it to STARTUP, or create a batch file that calls the program. If the Flashlite is to be used as a stand-alone device without a console connection, avoid writing to the console. The application will hang due to the hardware handshaking if it tries to write to the console.

! Although the flash memory devices used have a guaranteed lifetime of over 10,000 write cycles, it is possible for an application to quickly wear them out. The flash memory is intended to store programs and setup data which is normally not changed. Avoid storing data or frequently changed information on the flash disk.

The Special Function registers of the V-25 are located in the 0F000h segment of the memory. The following discussions assume the current segment is 0F000h. The V-25 ports and their associated configuration registers are also located in the 0F000h memory segment. For a more detailed description of the V-25 architecture and its special functions, see the NEC data books #50134-1 (V-25 Plus Data Sheet) and #UUI-UP50069 (V-25/V-35 Users's Manual). See the Contact Information section of this manual for details on how to contact the NEC literature department.

PORT T

Port T contains 8 analog comparator with a programmable threshold. The two registers associated with Port T are PT (Port T) at location 0FF38h and PMT (Port T Mode Register) at location 0FF3Bh. PMT sets the threshold for the comparators as a function of (n/16)*Vcc (+5V), where n is the decimal value in PMT, or 16 when PMT is zero. PMT is a read/write register with a reset value of 00h. The four high bits are always clear (0). The comparator results are stored in PT. Each comparison takes 64 clock cycles or approximately 16µS.

PORT 0,1,2

The following tables illustrate the configuration of the 3 ports. There are three configuration registers for each port, the Port (Px) register, the Port Mode (PMx) register, and the Port Mode Control (PMCx) register. These registers allow the port pins to be configured as inputs, outputs, or a special function. The following table shows the register addresses for each port.

	Port 0	Port 1	Port2
P	0FF00h	0FF08h	0FF10h
PM	0FF01h	0FF09h	0FF11h
PMC	0FF02h	0FF0Ah	0FF12h

		PMC0$_n$ = 0	
Port Pin	PMC0$_7$ = 1	PM0$_n$ = 1	PM0$_n$ = 0
P0$_7$	CLKOUT	Input Port	Output Port
P0$_6$	-	Input Port	Output Port
P0$_5$	-	Input Port	Output Port
P0$_4$	-	Input Port	Output Port
P0$_3$	-	Input Port	Output Port
P0$_2$	-	Input Port	Output Port
P0$_1$	-	Input Port	Output Port
P0$_0$	-	Input Port	Output Port

		PMC1$_n$ = 0	
Port Pin	PMC1$_n$ = 1	PM1$_n$ = 1	PM1$_n$ = 0
P1$_7$	READY Input	Input Port	Output Port
P1$_6$	SCKO/ Output	Input Port	Output Port
P1$_5$	TOUT Output	Input Port	Output Port
P1$_4$	INT Input	POLL Input	Output Port
P1$_3$	INTAK/ Output	INTP2 Input	-
P1$_2$	-	INTP1 Input	-
P1$_1$	-	INTP0 Input	-
P1$_0$	-	NMI Input	-

		PMC2$_n$ = 0	
Port Pin	PMC2$_n$ = 1	PM2$_n$ = 1	PM2$_n$ = 0
P2$_7$	HLDRQ Input	Input Port	Output Port
P2$_6$	HLDAK/ Output	Input Port	Output Port
P2$_5$	TC1/ output	Input Port	Output Port
P2$_4$	DMAAK1/ Input	Input Port	Output Port
P2$_3$	DMARQ1 Output	Input Port	Output Port
P2$_2$	TC0/ Output	Input Port	Output Port
P2$_1$	DMAAK0 Output	Input Port	Output Port
P2$_0$	DMARQ0/ Output	Input Port	Output Port

Timer Unit

The V-25 has two internal, programmable 16-bit interval timers (TM0 and TM1), each with variable input clock frequencies. Each of the two 16-bit timer registers has an associated 16-bit modulus register (MD0 and MD1). Timer 0 operates in the interval timer mode or one-shot mode; timer 1 has only the interval timer mode. The registers are all read/write, 16-bit (word) registers (except TMC0 and TMC1 which are 8-bit) and are located in memory as follows: TM0 - 0FF80h, MD0 - 0FF82h, TM1 - 0FF88h, MD1 - 0FF8A, TMC0 - 0FF90h, TMC1 - 0FF91h.

Interval Timer Mode

In this mode, TM0/TM1 are decremented by the selected input clock, and, after counting out, the registers are automatically reloaded from the modulus registers and counting continues. Each time TM1 counts out, interrupts are generated through TF1 and TF2 (timer flags 1 and 2). When TM0 counts out, an interrupt is generated through TF0. The timer-out signal can be used as a square-wave output whose half-cycle is equal to the count time.

Two input clocks derived from the system clock are SCLK/6 and SCLK/128. Typical timer values shown below are based on f_{OSC}= 8MHz and f_{SCLK}= f_{OSC}/2.

Clock	Timer Resolution	Full Count
SCLK/6	1.5 µS	98.304 µS
SCLK/128	32.0 µS	2.097 S

One-Shot Mode

In the one-shot mode, TM0 and MD0 operate as independent one-shot timers. Starting with a preset value, each is decremented to zero. At zero, counting ceases and an interrupt is generated by TF0 (from TM0) or TF1 (from MD0). When TM0 is programmed to one-shot mode, TM1 may still operate in interval mode.

Two input clocks derived from the system clock are SCLK/12 and SCLK/128. Typical timer values shown below are based on f_{OSC}= 8MHz and f_{SCLK}= f_{OSC}/2.

Clock	Timer Resolution	Full Count
SCLK/12	3.0 µS	196.608 µS
SCLK/128	32.0 µS	2.097 S

Timer Control Registers

Setting the desired timer mode requires programming the timer control register. See below for the TMC register format.

Timer Control Register 0 (TMC0)

TS0	TCLK0	MS0	MCLK0	ENT0	ALV	MOD_1	MOD_0
7							0

TS0	TM0 in Either Mode
0	Stop countdown
1	Start countdown

MOD_1	MOD_0	$TCLK_0$	TM0 Register Clock Frequency
0	0	0	$f_{SCLK}/6$ (Interval)
0	0	1	$f_{SCLK}/128$ (Interval)
0	1	0	$f_{SCLK}/12$ (One-Shot)
0	1	1	$f_{SCLK}/128$ (One-Shot)

MS0	MD0 Register Countdown (one-shot)
0	Stop
1	Start

MCLK0	MD0 Register Clock Frequency
0	$f_{SCLK}/12$
1	$f_{SCLK}/128$

ENT0	TOUT Square-Wave Output
0	Disable
1	Enable

ALV	TOUT Initial Level When Disabled by ENT0= 0
0	Low
1	High

MOD_1	MOD_0	Timer Unit Mode
0	0	Interval Timer
0	1	One-Shot
1	X	Reserved

Timer Control Register 1 (TMC1)

TS1	TCLK1	0	0	0	0	0	0
7							0

TS1	Timer 1 Countdown
0	Stop
1	Start

TCLK1	Timer 1 Clock Frequency
0	$f_{SCLK}/6$
1	$f_{SCLK}/128$

Serial Interface

The V-25 has two full-duplex UARTs, channels 0 and 1. Each channel has a transmit line (TxDn), a receive (RxDn), and a clear-to-send (CTSn) handshaking line. Communication is synchronized by a start bit, and either even, odd, or no parity may be programmed. Character length may be programmed to either 7 or 8 bits, and either 1 or 2 stop bits may be selected.

Each serial channel has a dedicated baud rate generator. The baud rate generators allow individual transfer rates for each channel up to 1.25 Mb/S. All standard baud rates are available. Each generator has an 8-bit data register (BRGn) that functions as a prescaler to a programmable input clock selected by the serial communications control register (SCCn). Together these must be set to generate a frequency equivalent to the desired baud rate.

The baud rate generator is programmed according to the following formula:

B = SCLK/(2^(n+1)*G)

where: B = baud rate
G = baud rate generator register (BRGn) value
n = input clock specification; value in SCCn register (0< n <8)
SCLK = system clock frequency (Hz)

Ex: For a desired baud rate of 19.2k baud, a MHz system clock (Flashlite), n would be 0 giving a BRGn value of 104.

Serial Communication Control Registers (SCCn)

0	0	0	0	PRS3	PRS2	PRS1	PRS0

7 0

PRS3	PRS2	PRS1	PRS0	Baud Rate Generator Input Clock Frequency
0	0	0	0	$f_{SCLK}/2$ (n= 0)
0	0	0	1	$f_{SCLK}/4$
0	0	0	0	$f_{SCLK}/8$
0	0	0	1	$f_{SCLK}/16$
0	1	1	0	$f_{SCLK}/32$
0	1	1	1	$f_{SCLK}/64$
0	1	0	0	$f_{SCLK}/128$
0	1	0	1	$f_{SCLK}/256$
1	0	0	0	$f_{SCLK}/512$ (n= 8)

Serial Interface Registers

Serial Communication Mode Registers (SCMn)

TxRDY	RxB	PRTY1	PRTY0	CLTSK	SLRSCK	MD1	MD0
7							0

TxRDY		Transmitter Control
0		Falling edge
1		Rising edge
RxB		**Receiver Control**
0		Falling edge
1		Rising edge
PRTY1	**PRTY1**	**Parity Control**
0	0	No parity
0	1	0 Parity (0 during Tx ignored for Rx)
1	0	Odd parity
1	1	Even parity
CLTSK		**Character Length**
0		7 Bits
1		8 Bits
SLRSCK		**Stop Bits**
0		1 stop bit
1		2 stop bits
MD1	**MD0**	**Mode**
0	0	I/O interface (channel 0 only)
0	1	Asynchronous
1	x	Reserved

Serial Status Register (SSTn)

RxDn	AS	TxBE	RxBF	0	ERP	ERF	ERO
7							0

RxDn	Receive Terminal Status
0,1	Status of RxD pin
AS	**All Sent Flag**
0 1	Data has been written in transmit buffer Data in transmit buffer and shift register sent
TxBE	**Transmit Buffer Empty Flag**
0 1	Data has been written in transmit buffer Data in buffer has been sent to shift register
RxBF	**Receive Buffer Full Flag**
0 1	Data has been read from receive buffer Data has been sent from shift register to buffer
ERP	**Parity Error Flag**
0 1	No Error Transmit and receive parity are different
ERF	**Framing Error Flag**
0 1	No Error Stop bit not detected
ERO	**Overrun Error Flag**
0 1	No Error Data is received before receive buffer is emptied

These registers provide error flags and buffer status information. The error bits are automatically cleared when the next data byte is received; otherwise these flags are persistent. The TxBE and RxBF bits signal the status of the respective transmit and receive buffers. These bits are reset automatically when either the baud rate generator or the serial control register contents are written. The AS bit is set when both the transmit buffer and transmit shift register are empty.

Interrupts

Processor Interrupts

Address	Vector	Assigned Use
00	0	Divide error
04	1	Break flag
08	2	NMI
0C	3	BRK3 instruction
10	4	BRKV instruction
14	5	CHKIND instruction
18	6	General purpose
1C	7	FPO instructions
20-2C	8-11	General purpose
30	12	INTSER0 (serial error, chan. 0)
34	13	INTSR0 (serial receive, chan. 0)
38	14	INTST0 (serial transmit, chan. 0)
3C	15	General purpose
40	16	INTSER1 (serial error, chan. 1)
44	17	INTSR1 (serial receive, chan. 1)
48	18	INTST1 (serial transmit, chan. 1)
4C	19	I/O trap
50	20	INTD0 (DMA channel 0)
54	21	INTD1 (DMA channel 1)
58	22	General purpose
5C	23	General purpose
60	24	INTP0 (peripheral 0)
64	25	INTP1 (peripheral 1)
68	26	INTP2 (peripheral 2)
6C	27	General purpose
70	28	INTTU0 (timer unit 0)
74	29	INTTU1 (timer unit 1)
78	30	INTTU2 (timer unit 2)
7C	31	INTTB (time base counter)
80-3FF	32-255	General purpose

Hardware Interrupts

The V-25 Plus features a high-performance on-chip controller capable of handling multiple processing for interrupts from up to 17 different sources (5 external, 12 internal). The 17 sources are divided into groups for management by the interrupt controller. Using software, each of the groups can be assigned a priority from 0 (highest) to 7 (lowest). The priority of individual interrupts within a group is fixed in hardware.

Group	Interrupt Source (Priority Within Group) 1	2	3	Default Priority
Nonmaskable interrupt	NMI	-	-	0
Timer unit	INTTU0	INTTU1	INTTU2	1
DMA controller	INTD0	INTD1	-	2
External peripheral interrupt	INTP0	INTP1	INTP2	3
Serial channel 0	INTSER0	INTSR0	INTST0	4
Serial channel 1	INTSER1	INTSR1	INTST1	5
Time base counter	INTTB	-	-	6
Interrupt request	INT	-	-	7

The priority of the currently active interrupt is stored in the ISPR special function register. Bits PR_7-PR_0 correspond to the eight possible interrupt request priorities. The ISPR keeps track of the priority of active interrupts by setting the appropriate bit of this register. The address of this register is 0FFFCh.

NMI and INT are system type, external, vectored interrupts. NMI is non maskable via software, and is recognized during DMA demand-release transfers. INT is maskable by the IE bit in the PSW and requires that an external device provide the interrupt vector number. It is designed to allow the interrupt controller to be expanded by another external interrupt controller.

NMI, INTP0-INTP1 are edge-sensitive inputs. By selecting the appropriate bits in the interrupt mode register, these inputs can be programmed to be either rising or falling edge triggered. Bits ES0-ES2 correspond to INTP0-INTP2, respectively, as shown below:

Interrupt Mode Register (IMR)

0	ES2	0	ES1	0	ES0	0	ESNMI
7							0

ES2	INTP2 Input Effective Edge
0	Falling edge
1	Rising edge
ES1	**INTP1 Input Effective Edge**
0	Falling edge
1	Rising edge
ES0	**INTP0 Input Effective Edge**
0	Falling edge
1	Rising edge
ESNMI	**NMI Input Effective Edge**
0	Falling edge
1	Rising edge

Interrupt Processing

Interrupts, with the exception of NMI, INT, and INTTB have high-performance capability and can be processed in any of three modes: standard vector method, register bank context switching (supported in hardware), and macroservice (SFR transfers). The processing mode for a given interrupt can be chosen by enabling the appropriate bits in the corresponding interrupt request control register. Each interrupt, except INT and NMI, has its own associated IRC register. The general format of these registers is shown below.

Interrupt Request Control Register (IRC)

IF	IMK	MS/INT	ENCS	0	PR_2	PR_1	PR_0
7							0

IF	Interrupt Flag
0	No interrupt request generated
1	Interrupt request generated
IMK	**Interrupt Mask**
0	Open (interrupts enabled)
1	Closed (interrupts disabled)
MS/INT	**Interrupt Response Method**
0	Vectored interrupt or register bank switching
1	Macroservice function
ENCS	**Register Bank Switching Function**
0	Not used
1	Used
PR_2-PR_0	**Interrupt Group Priority (0-7)**
0 0 0	Highest (0)
1 1 1	Lowest (7)

All interrupt processing routines other than those for NMI and INT must end with the execution of the FINT instruction. This instruction informs the interrupt controller that the current interrupt service routine in complete; if FINT is not executed within the service routine, only future interrupts of higher priority will be accepted.

In the vectored service mode, the CPU traps to a vector location.

Register Bank Switching

Register bank context switching allows interrupts to be processed rapidly by switching register banks. After an interrupt, the newly selected register bank has the same bank number (0-7) as the priority programmed in the associated IRC register. The PC and PSW are automatically sorted in the save areas of the new register bank, and the address of the interrupt routine is loaded from the vector PC storage register in the new register bank.

As in the vectored mode, the IE and BRK bits in the PSW are cleared to zero. After processing, the RETRBI instruction must be executed to return control to the original register bank and restore the former PC and PSW.

This method of interrupt service offers a dramatic performance advantage over normal vectored service because there is no need to store and retrieve data/registers on the stack. This also allows hardware-based real-time task switching in high-speed environments.

In addition to context switching, the task switch opcode (TSKSW) allows multiple independent processes to be internally resident.

BIOS

Supported Functions

The Flashlite Basic Input Output System supports the following software interrupts:

10h	Console Output
11h	Equipment Check
12h	Memory Size
13h	Disk Services
16h	Keyboard Driver
19h	Bootstrap

V-25 Resources and the BIOS

The BIOS uses the following V-25 ports, registers and register banks. The user should exercise extreme caution when using or modifying these resources. Some resources could be freed by writing some clever TSRs or by eliminating functions that are not needed for a specific application.

The PRC, WTC and several other registers associated with the hardware configuration, memory refresh, and system timing, hold vital system configuration information and should not be altered. Register bank 6 and timer channel 1 are used to handle the DOS timer tick and its associated interrupt. Registers TM1, MD1, TMC1 and TMIC1 are control the parameters of timer channel 1. Serial port 0 is used for the console. Registers SCM0, SCC0, BRG0, STIC0 and SRIC0 are all associated with serial port 0 and used for the console.

All of the I/O pins (Ports 0,1,2) are configured as inputs. Serial port 1 is configured for 300 baud, 8 data bits and 1 stop bit using registers SCM1, SCC1, BRG1, STIC1 and SRIC1. These resources are not used, just initialized to a known state.

XDOS

General Information

The Flashlite uses XDOS, compact operating system for embedded applications. The XDOS command structure is nearly identical to MS/PC DOS version 3.3. The switches for the dir command have been changed and expanded. XDOS does not support redirected input or output with the use of < and >, but does support pipes (|). None of the external DOS commands are available due to size constraints. XDOS does not support installable file system functions.

Command Reference

In the list below, XDOS commands are followed by a **function** description and their **format** including available **parameters** and **switches**. Items in **boldface** type must be entered. Captials or lowercase letters may be used. Items in *italics* are parameters. Those in ***boldface italics*** must be entered, those in [ ] are optional. All switches are optional. They are shown as [/X]. Spaces and punctuation are to be included. An ellipsis ... following items means that you may repeat the items as often as needed. Do not enter the ellipsis or the square brackets. Most XDOS commands allow the use of **wildcards** in filenames and extensions. When wildcards (?=one character, *=any character or characters) are used, the command is executed once for each matching file.

Common parameters are:

[*d:*]	drive specification - a letter followed by a colon (:), e.g. A:, if no drive is specified, the default drive is used.
[*path*]	the path DOS must take in traveling from one directory to another; directory names are separated by a backslash (\).
[*filename*]	up to eight characters used to name a file
[*.ext*]	a three character extension may be added to a filename; an extension is separated from a filename by a period.

CD / CHDIR

Function: Changes the current directory
Format: **CD** or **CHDIR** [[*d:*]*path*]

COPY

Function: Copies a file, combines two or more files into one file, or transfers data between files and DOS devices

Format: **COPY** [*d:*][*path*]***filename***[*.ext*][*switches*]
[+[*d:*][*path*]*filename*[*.ext*][*switches*]
[*d:*][*path*][*filename*[*.ext*]][*switches*]

Switches: /V - verify the contents of new file
/A - copy file in ASCII format
/B - copy file in binary format

DATE

Function: Displays or changes the current DOS date.

Format: **DATE** [*mm-dd-yy*]

DEL / ERASE

Function: Deletes (erases) one or more files from a disk

Format: **DEL** or **ERASE** [*d:*][*path*][*filename*[*.ext*]]

DIR

Function: Lists directory entries

Format: **DIR** [*d:*][*path*][*filename*[*.ext*]][*switches*]

Switches: /a - display file attributes
/b - sort by file size (in bytes)
/d - sort entries by date and time
/f - display entries by alphabetic file name order
/n - display entries in directory order (do not sort)
/s - include system and hidden files in output
/h - display this Help screen (any invalid key)

MD / MKDIR

Function: Creates a subdirectory

Format: **MD** or **MKDIR** [*d:*]***path***

PATH

Function: Specifies directories that DOS is to search when trying to locate executable files

Format: **PATH** [[*d:*]*path*[;[*d:*]*path* ...]]

PROMPT

Function: Sets the DOS system prompt
Format: **PROMPT** [*text]*

text string	**resulting character(s)**
$t	The current time stored by DOS
$d	The current date stored by DOS
$p	The current directory of the default drive
$v	The version of DOS being used
$n	The default drive
$g	The character >
$l	The character <
$b	The character |
$q	The character =
$$	The character $
$_	Carriage return plus line feed

REN

Function: Renames a file
Format: **REN** [*d:*][*path*]***filename***[*.ext*] ***filename***[*.ext*]

RD / RMDIR

Function: Deletes a subdirectory
Format: **RD** or **RMDIR** [*d:*]***path***

TIME

Function: Displays or changes the current DOS time
Format: **TIME** [*hh:mm:ss.xx*]

TYPE

Function: Display the contents of a file
Format: **TYPE** [*d:*][*path*]***filename***[*.ext*]

VER

Function: Displays the DOS version number
Format: **VER**

VOL

Function: Displays the volume label of specified drive
Format: **VOL** [*d:*]

Utilities

The Flashlite comes preloaded with several utilities to aid system development. These utilities are located on the A: drive of the Flashlite.

UP.COM

This utility facilitates uploading files to the Flashlite via the console port using the X-MODEM transfer protocol. The utility requires the user to supply the name of the incoming file. Unless otherwise specified, the file is placed in the active directory of the current drive. Be sure that B: is the current drive or a write-protect error will occur when UP tries to write to the read-only A: drive.

```
B:\>up
V-25 File Upload Program Version 1.1

Enter filename then upload file with X-Modem
protocol
... Press Cntl-A then Enter to Abort...
> test.exe
```

FLASHFMT.COM

```
If it becomes necessary to reformat the B: drive, FLASHFMT
provides this function.  CAUTION, all information on the drive
will be lost during the formatting process.

A:\>flashfmt
Flashlite FLASH Drive Format Program - Version
2.0
System will reboot after successful format...
Press 1 to initialize Drive B as 128k disk
Press ESC to exit with no changes

> 1

Flash Drive is now formatted
  Rebooting system...
```

EDIT.COM

A simple line editor is included to allow quick creation and modification of batch files or other text files. EDIT is similar to Microsoft's EDLIN provided in earlier versions of MS-DOS. It allows list, insert, delete, and modify. Upon exit, a backup of the original file is created (filename.BAK) and the edits are saved. If a backup file with the same name already exists, it is overwritten. A list of commands and their usage is available by

entering `h' at the edit prompt (>>). The name of the file to edit must be supplied following the command EDIT on the command line.

```
B:\>edit test.bat
Flashlite Line Editor v1.0

Enter h for help

New File: test.bat

>> i

     0: @echo Batch file being processed...
     1: mytsr
     2: myapp
     3: ^Z

>> l

     0: @echo Batch file being processed...
     1: mytsr
->   2: myapp

>> q

Save before exit (Y,n): y
File Saved
```

BASIC.COM

MICRO-BASIC is a compact, full featured, BASIC language interpreter that allows immediate development and testing of small programs on the Flashlite. Most of the examples in this document written in BASIC will run in MICRO-BASIC. Due to its size, the list of supported commands and functions is smaller than some other common BASIC languages, but it does provide some functions only found in large development packages. MICRO-BASIC supports a limited number of variables and arrays and only character/string and integer data types. MICRO-BASIC was written and is provided free of charge courtesy of Dave Dunfield of Dunfield Development Systems. Dunfield Development Systems provides a complete line of quality microprocessor development tools. See the Contact Information section for more details.

See Appendix B for a complete list of the commands and their usage.

Programming Examples

QuickBASIC I/O

Some of the code produced by Microsoft QuickBASIC and QuickBASIC Professional compilers does not execute properly on the Flashlite. In the case of console I/O, we believe that QuickBASIC is generating code for specific hardware and software not present on the Flashlite controller.

There are two problems with console I/O. The first is that a PRINT statement will not send output to the console port. To output text to the console, open "cons:" as a file and print to it. The second problem is that an INPUT statement will not echo the data entered by the user. To work around this problem, we have added a feature which allows the application to enable a console echo function in the BIOS. This feature is enabled by setting the byte at 40:8Ah to a one. Likewise, the local echo is disabled by setting 40:8Ah to a zero.

The following BASIC code demonstrates both of these workarounds:

```
start:
OPEN "o", 1, "cons:"         ' console output
PRINT #1, "What is your name? ";    ' get string
GOSUB echoOn                 ' turn on local echo
INPUT name$                  ' get the name
GOSUB echoOff                ' turn off local echo
PRINT #1, ""                 ' go down a line
PRINT #1, "Good to know you, "; name$      ' print line and
name
END

echoOn:
DEF SEG = &H40                       ' point to BIOS data seg
POKE &H8A, 1                 ' set local echo flag
RETURN

echoOff:
DEF SEG = &H40                       ' point to BIOS data seg
POKE &H8A, 0                 ' clear local echo flag
RETURN
```

Stoplight in MICRO-BASIC

The following two examples show implementations of a simple traffic light controller wirtten in MICRO-BASIC. This program uses Port 2 for both input and output. Six of the pins are configured as outputs that would turn on the appropriate lights, and the one input is an active low signal that informs the program it is time to change the direction of traffic flow. Two constants are used, 1) to control the delay before the red to yellow transition and 2) the duration of the yellow light. The status of the lights is also output to the console.

```
10 POKE 65298,0
15 REM set PMC2 to allow I/O, not special function
20 POKE 65297,128
25 REM set PM2 to configure pin 7 as input others as
output
30 POKE 65296,33
35 REM turn on one red and one green light (bit 0 and
bit 5 on)
40 D1=700
50 D2=500
55 REM delay times
60 L1$="GREEN"
65 L2$="RED"
70 L=1
80 PRINT L1$,"     ",L2$
90 IF TEST(7,PEEK(65296)) =0 THEN GOSUB 200
95 REM test to see if time to change, if yes do it.
100 GOTO 80
200 PRINT "change"
210 FOR I=1 TO D1
220 J=I/I
230 NEXT I
235 REM first delay loop
240 LIF L=1 THEN J = 34: L1$="YELLOW"
245 REM make lights yellow and red (bit 1 and bit 5 on)
250 LIF L=0 THEN J = 20: L2$="YELLOW"
255 REM make lights red and yellow (bit 2 and bit 4 on)
260 POKE 65296,J
270 PRINT L1$,"     ",L2$
280 FOR I=1 TO D2
290 J=I/I
300 NEXT I
305 REM second delay loop
310 IF L=0 THEN 350
320 L=0:L1$="RED":L2$="GREEN"
330 POKE 65296,12
335 REM make lights red and green (bit 2 and bit 3 on)
340 RETURN
350 L=1:L1$="GREEN":L2$="RED"
360 POKE 65296,33
365 REM make lights green and red (bit 0 and bit 5 on)
370 RETURN
```

Stoplight in C

This example is a C implementation of the above BASIC stoplight.

```
#include <stdio.h>
#define SEG 0xF000

int     d1=10000, /* delay before changing to
yellow */
        d2=7000,  /* length of yellow light */
        l=1;      /* keeps track of which set
is green */
char    l1[10], l2[10]; /*  strings  for  each
light group */

change()      /* function to change lights */
{
int i,j;
        printf( "change\n" );
                for(i=1; i< d1; i++) j=sqrt(i/i); /*
delay1 */
        if (l) {
                                j = 0x22;  strcpy( l1,
"YELLOW" );
        }   /* lights are yellow and red (bit1
& bit5 on)*/
        else {
                                j = 0x14;  strcpy( l2,
"YELLOW" );
        }   /* lights are red and yellow (bit2
& bit4 on)*/
        printf("%-10s%-10s\n", l1, l2 );
        poke ( SEG, 0xFF10, j );

                for( i=1; i<d2; i++) j=sqrt(i/i); /*
delay 2 */
        if (l) {
                 l=0;
                          strcpy( l1, "RED" ); strcpy(
l2, "GREEN" );
                  poke( SEG, 0xFF10, 0x0C );
                                /* lights are red and green
(bit2 & bit3 on) */
                 return(0);
        }
        l=1;
                 strcpy( l1, "GREEN" ); strcpy( l2,
"RED" );
        poke( SEG, 0xFF10, 0x21 );
                /* make lights green and red (bit 0
and bit 5 on) */
        return(0);
}

main(int argc, char *argv[])
{           /* see if delays specified on cmd
```

```
line*/
        if (argc == 3) {
                                    d1 = atoi(argv[1]); d2 =
atoi(argv[2]);
        }
                 strcpy( l1, "GREEN" ); strcpy( l2,
"RED" );
        poke( SEG, 0xFF12, 0 );
                 /* set PMC2 to allow I/O, not special
functions */
        poke( SEG, 0xFF11, 0x80 );
                 /* configure pin 7 as input others as
output */
        poke( SEG, 0xFF10, 0x21 );
                  /* red and green light (bit0 & bit5
on) */
                   while (1) {                    /*
infinite loop */
                                  printf( "%-10s%-10s\n", l1, l2
);
                                   if (! (peek( SEG, 0xFF10) &
0x80) ) change();
            /* see if time to change */
        }
}
```

Specifications

Processor	8 Mhz NEC V-25 Plus CPU Chip (10 MHz EN Model)
RAM Memory	512K Bytes Pseudo Static RAM
Flash Memory	256K Bytes Flash Memory (512K EN Model)
Serial Ports	(2) Fully Compliant RS-232 Ports, configurable for TTL Serial I/O
Power Requirements	5 Volts ±10%
Current Consumption	110 mA, Typical
Operating Temperature	-4 to 185 °F (-20 to +85 °C)
Dimensions	4.2" x 3.6" (106.7 mm x 91.4 mm) All holes on 0.100" centers, #6 mounting holes 0.250" from board edges
Weight	2.5 Oz. (70 gm.)

Contact Information

JK Microsystems
1275 Yuba Ave.
San Pablo, CA 94806

Voice/FAX: (510) 236-1151

Dunfield Development Systems
MICRO-BASIC
Microprocessor development tools
P.O. Box 31044
Nepean, Ontario, Canada, K2B 8S8

Voice: (613) 256-5820
FAX: (613) 256-5821
BBS: (613) 256-6289

NEC Electronics Inc. - Literature
V-25 Plus Data Sheet (80 pages) #50134-1
V-25/V-35 User's Manual #UUI-UP50069

Voice: (800) 632-3531

Appendix A: Pin Descriptions

J1 - Processor Bus

Processor Bus J1

GND	1	2	VCC
GND	3	4	VCC
MREQ/	5	6	D7
MSTB/	7	8	D6
IOSTB/	9	10	D5
RW/	11	12	D4
REFRQ/	13	14	D3
RESET/	15	16	D2
IORD/	17	18	D1
IOWR/	19	20	D0
A9	21	22	A19
A8	23	24	A18
A7	25	26	A17
A6	27	28	A16
A5	29	30	A15
A4	31	32	A14
A3	33	34	A13
A2	35	36	A12
A1	37	38	A11
A0	39	40	A10

Vcc Two power pins

GND Two ground pins

MREQ/ Memory request (active low), indicates memory access cycle

MSTB/ Memory Strobe (active low), asserted during memory operations

RW/ Read/Write, read or write status of current cycle

REFRQ/ Refresh (active low), pulse for memory refresh

IOSTB/ I/O device strobe

IORD/ I/O device read (active low), asserted during I/O read. Data is read on positive edge of pulse.

IOWR/ I/O device write (active low), asserted during I/O write. Data is written on positive edge of pulse.

A0-A19 Processor address line (0-19)

D0-D7 Processor data lines (0-7)

J2 - Power

Power J2

1	VCC
2	RESET/
3	GND

Vcc 5 Volt power supply for Flashlite

GND System ground

RESET/ Active low processor reset pin (input)

J3 - Port T and Port 0

Port T and Port 0 J3

GND	1	2	VCC
GND	3	4	VCC
GND	5	6	TTL-TXD0/
GND	7	8	TTL-RXD0/
RS232-0 ENABLE	9	10	TTL-CTS0/
PT7	11	12	P0.7,CLKOUT
PT6	13	14	P0.6
PT5	15	16	P0.5
PT4	17	18	P0.4
PT3	19	20	P0.3
PT2	21	22	P0.2
PT1	23	24	P0.1
PT0	25	26	P0.0

Vcc Two power pins

GND Four ground pins

RS-232-0 ENABLE

This pin sets the state of the channel 0 RS-232 driver. Take this pin low to disable the driver and allow TTL input serial I/O. The pin is pulled to Vcc by a 10k resistor. NOTE: This will disable the RS-232 console, all console I/O must be at TTL levels.

TTL-TXD0, RXD0/ CTS0/

TTL serial I/O pins (transmit data, receive data, and clear to send). Use these pins when RS-232-0 ENABLE is LOW.

PORT T (Comparator Port)

PT0-PT7 are inputs to the analog comparator port.

PORT 0

P0.0-P0.7 are lines of an 8-bit bidirectional I/O port.
P0.7 also provides the internal system clock output signal

J4 - Serial Port 0 (Console)

Serial Port 0 (Console) J4

JUMPER TO PIN 7	1	2	RS232-DSR0
RS232-TXD0	3	4	RS232-CTS0
RS232-RXD0	5	6	RS232-DSR0
JUMPER TO PIN 1	7	8	N/C
GND	9	10	N/C

RS-232 Inputs and outputs for serial port 0.
TXD0 Data Transmitted by Flashlite (output)
RXD0 Data Received by Flashlite (input)
DSR0 Data Set Ready (output)
CTS0 Clear to Send (input)

J5 - Port 1 and Port 2

Port 1 and Port 2 J5

GND	1	2	VCC
GND	3	4	VCC
GND	5	6	TTL-TXD1/
GND	7	8	TTL-RXD1/
RS232-1 ENABLE	9	10	TTL-CTS1/
P1.7, READY	11	12	P2.7, HLDRQ
P1.6, SCK0/	13	14	P2.6, HLDAK/
P1.5, TOUT	15	16	P2.5, TC1/
P1.4, INT, POLL/	17	18	P2.4, DMAAK1/
P1.3, INTP2, AK/	19	20	P2.3, DMARQ1
P1.2, INTP1	21	22	P2.2, TC0/
P1.1, INTP0	23	24	P2.1, DMAAK0/
P1.0, NMI	25	26	P2.0, DMARQ0

Vcc, GND - Two power and four ground pins

RS-232-1 ENABLE

This pin sets the state of the channel 1 RS-232 driver. Take this pin low to disable the driver and allow TTL input serial I/O. The pin is pulled to Vcc by a 10k resistor.

TTL-TXD0, RXD0/ CTS0/

TTL serial I/O pins (transmit data, receive data, and clear to send). Use these pins when RS-232-ENABLE-1 is LOW.

PORT 1

Port 1 is a 5-bit bidirectional I/O port.
P1.0 - P1.3 provide control functions.
P1.4 - P1.7 individually configurable as inputs, outputs, or control functions
P1.0, NMI - Nonmaskable interrupt input, pulled high by a 10k resistor
P1.1, INTP0 - Port 0 interrupt input
P1.2, INTP1 - Port 1 interrupt input
P1.3, INTP2,AK/ - Port 2 interrupt input or interrupt acknowledge output
P1.4, INT, POLL/ - maskable, active-high, vectored interrupt or poll input
P1.5, TOUT - square-wave output signal from the internal timer
P1.6, SCK0/ - serial clock output
P1.7, READY - input

PORT 2

Port 2 is a 8-bit bidirectional I/O port. It is also configurable for use with the DMA controller.
P2.0, DMARQ0 - DMA request, channel 0
P2.1, DMAAK0/ - DMA acknowledge, channel 0
P2.2, TC0/ - Terminal count, channel 0
P2.3, DMARQ1 - DMA request, channel 1

P2.4, DMAAK1/ - DMA acknowledge, channel 1
P2.5, TC1/ - Terminal count, channel 1
P2.6, HLDAK/ - active low output indicating the CPU has released the bus
P2.7, HLDRQ - active high input to request the CPU release the system bus

---

! J4 and J6 are not identical, functionally or electrically.

J6 - Serial Port 1

RS-232 Inputs and Outputs for Serial Port 1

TXD1	Transmit Data (output)
RXD1	Receive Data (input)
DSR1	Data Set Ready (output)
CTS1	Clear to Send (input)
N/C	No Connection

Serial Port 1 (COM1) J6

N/C	1	2	N/C
RS232-RXD1	3	4	RS232-DSR1
RS232-TXD1	5	6	NC
RS232-DSR1	7	8	RS232-CTS1
GND	9	10	N/C

Note: CTS is connected to +5 volts via a weak pullup resistor and may be left unconnected if not needed.

Appendix B: Application Hints

A malfunctioning application can be terminated at startup by typing CTRL-C immediately after applying power or resetting the board.

If the Flashlite board hangs while sending characters, insure that the handshaking lines are high. For the console port, J4 pin 4 must be high. For the second serial port, J6 pin 4 must be high.

Borland Turbo C erroneously detects the presence of a floating point coprocessor on the Flashlite board, causing floating point functions to hang. To explicitly overide the detection, add the following line to the STARTUP.BAT file:

```
SET 87=N
```

The timer registers TM and MD must be loaded and read with 16-bit move instructions. 8-bit PEEK and POKE commands will not work. See the Utilities Disk for linkable QuickBASIC routines that perform these functions.

Appendix C: MICRO-BASIC Language Reference

Things to remember:

Program lines must be numbered.
MICRO-BASIC only supports characters and signed integers (-32767 to 32767), no floats.

Variables:

260 Numeric variables : A0-A9 ... Z0-Z9
260 Character variables : A0$-A9$ ... Z0$-Z9$
260 Numeric arrays : A0()-A9() ... Z0()-Z9()

For convenience the '0' variables can be referenced by letter only. IE: A is equivalent to A0 ... Z$ is equivalent to Z0$

Conventions:

var - numeric variable or expression that evaluates to a number
char - character variable or expression that evaluates to a character
line - program line number
value - signed decimal integer (-32767 to 32767) or similar expression
port - decimal equivalent of a hardware port
loc[ation] - decimal equivalent of a memory location
bit - a value from 0 to 7 representing a bit location in a 8-bit integer
name - the name of a file
string - a constant string of characters, must be delimited by quotes (")
[*xxx*] - optionally include the expression in brackets

Operators:

+	- Addition, string concatenation
-	- Unary minus, subtraction
*, /, %	- multiplication, division, modulus
&, ¦, ^	- AND, OR, Exclusive OR
=, < >	- Assign/test equal, test NOTequal (num or string)
<, < =, >, > =	- LT, LE, GT, GE (numbers only)
!	- Unary NOT

The `test' operators (=, < >, <, < =, >, > =) can be used in any expression, and evaluate to 1 if the test is TRUE, and 0 if it is FALSE. The IF and LIF commands accept any non-zero value to indicate a TRUE condition.

Output from sample statements appears in a `different font` following the example; descriptions of results appear at the right margin following the example.

Statements:

CLEAR

Erases (clears) all program variables
CLEAR

CLOSE

Close file (0-9) opened with OPEN
See also: OPEN
CLOSE *#n*

CLR

Clear a specified bit (0-7) in passed variable
See also: SET, TEST
CLR *bit*, *var*
EX: i= 7: CLR 0,i: PRINT i 6

DATA

Enter "inline" data statements to be loaded into variables in the program
See also: READ, ORDER
DATA *values*
EX: see READ for an example

DEF SEG

Define or reset memory segment. The default segment is 0F000h, the V-25 register area.
See also: PEEK, POKE
DEF SEG [*value*]
EX: DEF SEG resets segment to hex F000
DEF SEG DEC("1000") sets segment to hex 1000

DIM

Dimension an array(s) with the given range
DIM *var*(*size*)[, ...]
EX: DIM a(10),b(20)

END

Terminate program with no message
END

EXIT

Terminate MICRO-BASIC. Remember to save programs before EXITing
EXIT

FOR

Perform a counted loop with the given start and end limits, optionally a variable change other than 1 can be specified.
See also: NEXT
FOR *var*= init TO limit [STEP increment]
EX: FOR i= 1 TO 10 : PRINT i: NEXT
FOR i= 10 TO 1 STEP -1 : PRINT i: NEXT i

GOSUB

Call a subroutine
See also: RETURN
GOSUB *line*
EX: 10 GOSUB 100
< more code >
100 PRINT x,y,z
110 RETURN

GOTO

Jump to specified line and continue executing program
GOTO *line*

IF...THEN

Conditional statement. If condition branch to line or execute statement.
NOTE: ELSE is not supported. Only one statement can follow the test, for multiple statements see LIF.
See also: LIF
If *test* THEN *line*
IF *test* THEN *statement*
EX: IF (i> j) 100
IF (i> j) PRINT j

INPUT

Get a value for variable from the console, optionally with a prompt, or from a file. The prompt must be a constant string, however you can use a char variable in prompt by concatenating it to such a string: INPUT ""+ a$,b$
See also: OPEN
INPUT[*# n*,] ["*prompt*",] *var*
EX: INPUT "Enter a string, default= "+ a$,j$
INPUT # 1,b

LET

Initialize a variable, or set a variable equal to an expression.
LET *var* = *expression*
EX: LET i= 5
i= ASC("a")

LIF

Long IF statement. All statement on the same line are processed.
See also: IF
LIF *test* THEN *statements*
EX: LIF (i< j) THEN i= k: i= j: j= k

LIST

List lines in program, optionally save listed lines to a file opened with OPEN
See also: OPEN, CLOSE
LIST[*# n*] [*start*,[*end*]]
EX: LIST # 1 10,100 Save lines 10 through 100 to open file # 1
LIST 10, List program starting at line 10

LOAD

Load program from disk file. When LOAD is used within a program, execution continues with the first line of the newly loaded program. In this case, the user variables are NOT cleared. This provides a means of chaining to a new program, and passing information to it. Also note that LOAD must be the LAST statement on a line.
LOAD *name*
EX: LOAD light.bas Load the program light.bas

NEW

Erase program and variables
NEW

NEXT

End of counted loop, variable name can be omitted if loops are not nested
See Also: FOR
NEXT[*var*]
EX: See FOR statement for example

OPEN

Open a data file (0-9) with the given name using the provided attributes. Attributes can be combined.
attributes:

a - Append to the file (must use with w)
b - Binary mode (text is the default)
r - Open file for read
w - Open file for output
v - Verbose, issue error messages on failure

See also: CLOSE
OPEN *# n*,"*name*","*attributes*"
EX: OPEN # 1, "test.dat","wv" Open file # 1 for output
OPEN # 2, "in.dat","r" Open file # 2 for input

ORDER

Positions the data read pointer to a valid data line. This must be done before a READ statement is issued.
See also: DATA, READ
ORDER *line*
EX: See READ for an example of ORDER.

OUT

Output value to the specified hardware port.
See also: IN
OUT *port*, *value*
EX: OUT DEC("300"),5 Output to port 300h the value 5

POKE

Set the contents of memory location loc to the value specified. The memory segment is specified by the DEF SEG statement.
See also: DEF SEG, PEEK
POKE *loc*, *value*
EX: POKE 500,1 Set location decimal 500 to 1

PRINT

Output information to the console or optionally an open file.
See also: OPEN
PRINT[*# n*] *expr*[,*expr* ...]
EX: PRINT i,t$
PRINT# 1 "This is a test", i

READ

Read data from program DATA statements. An ORDER statement targeting a line containing a valid DATA statement must be issued prior to the READ
See also: DATA, ORDER
READ *var*[,*var* ...]
EX: 10 ORDER 100
20 READ i,j,k,l
100 DATA 100,12,34,17

REM

Comment, reminder of line is ignored
REM
EX: REM Prints the hex values of numbers 0 through 16

RETURN

Return from execution of a subroutine
RETURN

RUN

Run program
RUN [*line*]
EX: RUN Begins executing the program at the first line
RUN 20 Begins execution at line 20

SAVE

Save program to disk file
SAVE [*name*]
EX: SAVE
SAVE light.bas

SET

Set a specified bit (0-7) in passed variable
See also: CLR, TEST
SET *bit, var*
EX: i= 6: SET 0,i: PRINT i **7**

STOP

Terminate program & issue message
STOP

Functions:

ABS

Returns absolute value of argument
ABS(*value*)
EX: PRINT ABS(-5), ABS(5) **5 5**

ASC

Returns the decimal ASCII value of the passed character
See also: CHR$
ASC(*char*)
EX: PRINT ASC("A") **65**

CHR$

Returns the ASCII character equivalent to a value between 0 and 255
See also: ASC
CHR$(*value*)
EX: PRINT CHR$(65) **A**

DEC

Returns the decimal equivalent of the input hex string
See also: HEX$
DEC(*string*)
EX: PRINT DEC("FF") **255**

HEX$

Returns a string containing the hexadecimal equivalent of the passed value
See also: DEC
HEX$(*value*)
EX: PRINT HEX$(100) **80**

INP

Returns value from hardware port
See also: OUT
INP(*port*)
EX: PRINT INP(DEC("200")) **125**

PEEK

Returns value from specified memory location in the defined segment.
See also: DEF SEG, POKE
PEEK(*location*)
EX: PRINT PEEK(DEC("F38")) **8**

RND

Returns a random number from 0 to (value-1)
RND(*value*)
EX: PRINT RND(5) **3**

STR$

Returns an ASCII string of representing the numeric argument
STR$(*value*)
EX: D$ = STR$(365) + "Days"

TEST

Returns status of requested bit (0-7) passed value
See also: SET, CLR
TEST(*bit*, *value*)
EX: TEST(1,5) **0**