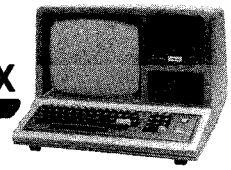
Section 3:
Appendices



A / Model III Summary

Special Characters and Abbreviations

Command Mode	Function
(ENTER)	Return carriage and interpret command
⊖	Cursor backspace and delete last character typed
(SHIFT)⊖	Cursor to beginning of line; erase line
⊕	Linefeed
:	Statement delimiter; use between statements on same logical line
⊖	Move cursor to next tab stop. Tab stops are at positions 0, 8, 16, 24, 32, 48, and 56.
(SHIFT)⊖	Convert display to 32 characters per line
(CLEAR)	Clear Display and convert to 64 characters per line

Execute Mode	Function
(SHIFT)@	Pause in execution; freeze display during LIST
(BREAK)	Stop execution
(ENTER)	Interpret data entered from Keyboard with INPUT statement

Abbreviations	Function
?	Use in place of PRINT.
'	Use in place of :REM
.	"current line"; use in place of line number with LIST, EDIT, etc.

To output a control character, press (SHIFT) then ⊕; while holding down both keys, press the key for which a control character is desired. For example, to key a control —Z press:

(SHIFT)⊕(Z)

Type Declaration Characters

Character	Type	Examples	Section 2 Page
$	String	A$, ZZ$	1/13
%	Integer	A1%, SUM%	1/13
!	Single-Precision	B!, NI!	1/12
#	Double-Precision	A#, 1/3#	1/12
D	Double-Precision (exponential notation)	1.23456789D-12	1/12
E	Single-Precision (exponential notation)	1.23456E+30	1/12

Arithmetic Operators

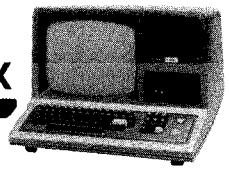
		Section 2 Page
+	add	1/19
−	subtract	1/19
*	multiply	1/19
/	divide	1/19
[	exponentiate (e.g., 2 [3 = 8) Press ↑ to generate "[".	1/19

String Operator

			Section 2 Page
+	concatenate (string together)	"2" + "2" = "22"	1/22

Relational Operators

Symbol	in numeric expressions	in string expressions	Section 2 Page
<	is less than	precedes	1/23
>	is greater than	follows	1/23
=	is equal to	equals	1/23
< = or = <	is less than or equal to	precedes or equals	1/23
> = or = >	is greater than or equal to	follows or equals	1/23
<> or ><	does not equal	does not equal	1/23

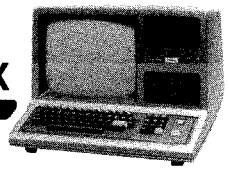


Order of Operations	Section 2 Page
[or ↑ (Exponentiation) Press ↑ to enter this character.	1/26
− (Negation)	1/26
*,/	1/26
+, −	1/26
Relational operators	1/26
NOT	1/26
AND	1/26
OR	1/26
Precedence order is from left to right for operators on the same level	1/26

Commands

Command/Function	Examples	Section 2 Page
AUTO *mm, nn* Turn on automatic line numbering beginning with *mm*, using increment of *nn*.	AUTO AUTO 10 AUTO 5,5 AUTO .,10	2/1
CLEAR Set numeric variables to zero, strings to null.	CLEAR	
CLEAR *n* Same as CLEAR but also sets aside *n* bytes for strings.	CLEAR 500 CLEAR MEM/4	2/2
CLOAD Load a BASIC program from tape	CLOAD"A"	2/2
CLOAD? Verifies BASIC program on tape to one in memory	CLOAD?"A"	2/3
CONT Continue after BREAK or STOP in execution.	CONT	2/3

CSAVE Save a BASIC program on tape	CSAVE"A"	2/3
DELETE *mm-nn* Delete program line from line *mm* to line *nn*.	DELETE 100 DELETE 10-50 DELETE.	2/4
EDIT *mm* Enter Edit Mode for line *mm*. See Edit Mode Sub-commands below.	EDIT 100 EDIT.	2/4
LIST *mm-nn* List all program lines from *mm* to *nn*.	LIST LIST 30-60 LIST 30- LIST-90 LIST.	2/4
LLIST *mm-nn* Lists all program lines from *mm* to *nn* on the line printer.	LLIST LLIST 30-60	2/5
NEW Delete entire program and reset all variables, pointers etc.	NEW	2/5
RUN *mm* Execute program beginning at lowest numbered line or *mm* if specified.	RUN RUN 55	2/6
SYSTEM Enter Monitor Mode for loading of machine-language file from cassette.	SYSTEM	2/6
TROFF Turn off Trace	TROFF	2/7
TRON Turn on Trace	TRON	2/7



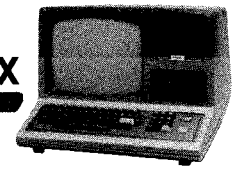
Edit Mode Subcommands and Functions

Sub-Command	Function	Section 2 Page
(ENTER)	End editing and return to Command Mode.	9/2
(SHIFT) (↑)	Escape from X, I, and H subcommands and remain in Edit Mode.	9/3
n(SPACEBAR)	Move cursor *n* spaces to right.	9/2
n(←)	Move cursor *n* spaces to left.	9/3
(L)	List remainder of program line and return to beginning of line.	9/3
(X)	List remainder of program line, move cursor to end of line, and start Insert subcommand.	9/4
(I)	Insert the following sequence of characters at current cursor position; use Escape to exit this subcommand.	9/4
(A)	Cancel changes and return cursor to beginning of line	9/5
(E)	End editing, save all changes and return to Command Mode.	9/5
(Q)	End editing, cancel all changes made and return to Command Mode.	9/5
(H)	Delete remainder of line and insert following sequence of characters; use Escape to exit this subcommand.	9/5
n(D)	Delete specified number of characters *n* beginning at current cursor position.	9/6
n(C)	Change (or replace) the specified number of characters *n* using the next *n* characters entered.	9/6
n(S)*c*	Move cursor to *n*th occurrence of character *c*, counting from current cursor position.	9/7
n(K)*c*	Delete all characters from current cursor position up to *n*th occurrence of character *c*, counting from current cursor position.	9/7

Input/Output Statements

Statement/Function	Examples	Section 2 Page
PRINT *exp**		
Output to Display the value of *exp*. *Exp* may be a numeric or string expression or constant, or a list of such items.	PRINT A$ PRINT X+3 PRINT "D=" D	3/1
Comma serves as a PRINT modifier. Causes cursor to advance to next print zone.	PRINT 1,2,3,4 PRINT "1", "2" PRINT 1,,2	
Semi-colon serves as a PRINT modifier. Inserts a space after a numeric item in PRINT list. Inserts no space after a string item. At end of PRINT list, suppresses the automatic carriage return.	PRINT X;" = ANSWER" PRINT X;Y;Z PRINT "ANSWER IS";	
PRINT@*n*		
PRINT modifier; begin PRINTing at specified display position *n*.	PRINT @ 540, "CENTER" PRINT @ N+3,X*3	3/2
PRINT TAB *n*		
Print modifier: moves cursor to specified Display position *n* (expression).	PRINT TAB(N) N	3/3
PRINT USING *string;exp*		
PRINT format specifier; output *exp* in form specified by *string* field (see below).	PRINT USING A$;X PRINT USING "#.#";Y+Z	3/4
INPUT *"message";variable*		
Print message (if any) and await input from Keyboard.	INPUT"ENTER NAME";A$ INPUT"VALUE";X INPUT"ENTER NUMBERS";X,Y INPUT A,B,C,D$	3/8
LPRINT		
Output to line printer.	LPRINT A$	3/12
PRINT # − 1		
Output to Cassette.	PRINT # − 1,A,B,C,D$	3/12

**exp* may be a string of numeric constant or variable, or a list of such items.



INPUT # – 1
Input from Cassette. — INPUT # – 1,A,B,C,D$ — 3/13

DATA *item list*
Hold data for access by READ statement. — DATA 22,33,11,1.2345
DATA "HALL", "SMITH", "DOE" — 3/10

READ *variable list*
Assign value(s) to the specified variable(s), starting with current DATA element. — READ A,A1,A2,A3
READ A$,B$,C$,D — 3/10

RESTORE
Reset DATA pointer to first item in first DATA statement. — RESTORE — 3/11

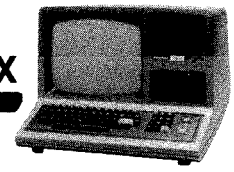
Field Specifiers for PRINT USING statements

Numeric Character	Function	Example	Section 2 Page
#	Numeric field (one digit digit per #).	###	3/4
.	Decimal point position.	##.###	3/4
+	Print leading or trailing signs (plus for positive numbers, minus for negative numbers).	+#.### #.###+	3/5
–	Print trailing sign only if value printed is negative.	###.##–	3/5
**	Fill leading blanks with asterisk.	**###.##	3/4
$$	Place dollar sign immediately to left of leading digit.	$$####.##	3/4
**$	Dollar sign to left of leading digit and fill leading blanks with asterisks.	**$####.##	3/4
[[[[or ⓐⓐⓐⓐ	Exponential format, with one significant digit to left of decimal. Press ⓐ to input this character.	#.##[[[[	3/4

,	Prints out number with commas, as in 1,356,000	#,######	3/4
!	Single character.	!	3/5
%spaces%	String with length equal to 2 plus number of spaces between % symbols.	%%	3/5

Program Statements

Statement/Function	Examples	Section 2 Page
(Type Definition)		
DEFDBL *letter list or range* Define as double-precision all variables beginning with specified letter, letters or range of letters.	DEFDBL J DEFDBL X,Y,A DEFDBL A-E,J	4/3
DEFINT *letter list or range* Define as integer all variables beginning with specified letter, letters or range of letters.	DEFINT A DEFINT C,E,G DEFINT A-K	4/2
DEFSNG *letter list or range* Define as single-precision all variables beginning with specified letter, letters or range of letters	DEFSNG L DEFSNG A-L, Z DEFSNG P,R,A-K	4/2
DEFSTR *letter list or range* Define as string all variables beginning with the specified letter, letters, or range of letters.	DEFSTR A-J	4/3
(Assignment and Allocation)		
CLEAR *n* Set aside specified number of bytes *n* for string storage. Clears value and type of all variables.	CLEAR 750 CLEAR MEM/10 CLEAR 0	4/4



DIM *array(dim#1,...,dim#k)*

Allocate storage for *k*-dimensional *array* with the specified size per dimension: dim #1, dim#2,..., etc. DIM may be followed by a list of arrays separated by commas.

DIM A(2,3)
DIM A1(15), A2(15)
DIM B(X+2),C(J,K)
DIM T(3,3,5)

4/4

LET *variable = expression*

Assign value of *expression* to *variable*. LET is optional in LEVEL II BASIC.

LET A$ = "CHARLIE"
LET B1 = C1
LET A% = 1#

4/5

(Sequence of Execution)

END

End execution, return to Command Mode.

99 END

4/5

STOP

Stop execution, print Break message with current line number. User may continue with CONT.

100 STOP

4/6

GOTO *line-number*

Branch to specified *line-number*.

GOTO 100

4/6

GOSUB *line-number*

Branch to sub-routine beginning at *line-number*.

GOSUB 3000

4/7

RETURN

Branch to statement following last-executed GOSUB.

RETURN

4/7

ON *exp* GOTO *line#1,...,line#k*

Evaluate expression; if INT (exp) equals one of the numbers 1 through *k*, branch to the appropriate line number. Otherwise go to next statement.

ON K+1 GOTO 100,200,300

4/8

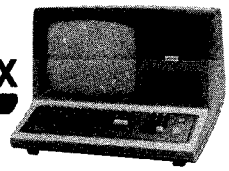
ON *exp* GOSUB *line#1,...,line#k*

Same as ON...GOTO except branch is sub-routine beginning at *line#1, line#2,...,* or *line#k*, depending on *exp*.

ON J GOSUB 330,700

4/9

Statement/Functions	Examples	Section 2 Page
FOR *var* = *exp* TO *exp* STEP *exp* Open a FOR-NEXT loop. STEP is optional; if not used, increment of one is used.	FOR I = 1 TO 50 STEP 1.5 FOR M% = J% TO K% – 1	4/9
NEXT *variable* Close FOR-NEXT loop. *Variable* may be omitted. To close nested loops, a variable list may be used. See Chapter 4.	NEXT NEXT I NEXT I,J,K	4/9
ERROR *(code)* Simulate the error specified by *code* (See Error Code Table).	ERROR(14)	4/12
ON ERROR GOTO *line-number* If an error occurs in subsequent program lines, branch to error routine beginning at *line-number*.	ON ERROR GOTO 999	4/12
RESUME *n* Return from error routine to line specified by *n*. If *n* is zero or not specified, return to statement containing error. If *n* is "NEXT", return to statement following error-statement.	RESUME RESUME 0 RESUME 100 RESUME NEXT	4/3
RANDOM Reseeds random number generator.	RANDOM	7/4
REM REMark indicator; ignore rest of line.	REM A IS ALTITUDE	4/14



(Tests – Conditional Statements)

IF *exp-1* THEN *statement-1*
ELSE *statement-2*

Tests *exp-1:* If True, execute *statement-1* then jump to next program line (unless statement-1 was a GOTO).

If *exp-1* is False, jump directly to ELSE statement and execute subsequent statements.

IF A = 0 THEN PRINT "ZERO"
ELSE PRINT "NOT ZERO"

4/14-**4**/15

(Graphics Statements)

CLS

Clear Video Display

CLS

8/2

RESET*(x,y)*

Turn off the graphics block with horizontal coordinate *x* and vertical coordinate *y*, 0< = X<128 and 0< = Y<48

RESET (8 + B,11)

8/2

SET *(x,y)*

Turn on the graphics block specified by coordinates *x* and *y*. Same argument limits as RESET

SET(A*2,B + C)

8/1

(Special Statements)

POKE *location, value*

Load *value* into memory *location* (both arguments in decimal form) 0< = *value*< = 255.

POKE 15635,34
POKE 17770,A + N

8/5

OUT *port, value*

Send *value* to *port* (both arguments between 0 and 255 inclusive)

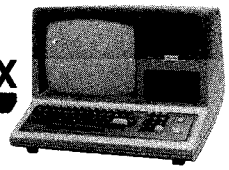
OUT 255,10
OUT 55,A

8/5

String Functions*

Function	Operation	Examples	Section 2 Page
ASC(*string*)	Returns ASCII code of first character in string argument.	ASC(B$) ASC("H")	5/2
CHR$(*code exp*)	Returns a one-character string defined by *code*. If *code* specifies a control function, that function is activated.	CHR$(34) CHR$(1)	5/2
FRE(*string*)	Returns amount of memory available for string storage. Argument is a dummy variable.	FRE(A$)	5/3
INKEY$	Strobes Keyboard and returns a one-character string corresponding to key pressed during strobe (null string if no key is pressed).	INKEY$	5/4
LEFT$(*string,n*)	Returns first *n* characters of *string*.	LEFT$(A$,1) LEFT$(L1$ + C$,8) LEFT$(A$,M + L)	5/5
LEN(*string*)	Returns length of *string* (zero for null string).	LEN(A$ + B$) LEN("HOURS")	5/5
MID$(*string,p,n,*)	Returns substring of *string* with length *n* and starting at position *p* in *string*.	MID$(M$,5,2) MID$(M$ + B$,P,L − 1)	5/6
RIGHT$(*string,n*)	Returns last *n* characters of *string*.	RIGHT$(NA$,7) RIGHT$(AB$,M2)	5/6
STR$(*numeric exp*)	Returns a string representation of the evaluated argument.	STR$(1.2345) STR$(A + B*2)	5/6
STRING$(*n,char*)	Returns a sequence of *n char* symbols using first character of *char*.	STRING$(30, ".") STRING$(25, "A") STRING$(5,C$)	5/7
TIME$	Returns date and time.	TIME$	5/8
VAL(*string*)	Returns a numeric value corresponding to a numeric-valued string.	VAL("1" + A$ + "." + C$) VAL(A$ + B$) VAL(G1$)	5/8

**string* may be a string variable, expression, or constant.



Arithmetic Functions*

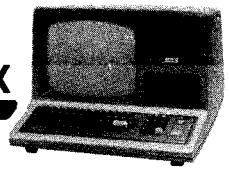
Function	Operation (unless noted otherwise, $-1.7E+38 <= exp <= 1.7E+38$)	Examples	Section 2 Page
ABS(*exp*)	Returns absolute value.	ABS(L*.7) ABS(SIN(X))	7/1
ATN(*exp*)	Returns arctangent in radians.	ATN(2.7) ATN(A*3)	7/1
CDBL(*exp*)	Returns double-precision representation of *exp*.	CDBL(A) CDBL(A+1/3#)	7/2
CINT(*exp*)	Returns largest integer not greater than *exp*. Limits: $-32768 <= exp <= +32768$.	CINT(A#+B)	7/2
COS(*exp*)	Returns the cosine of *exp*; assumes *exp is in radians*.	COS(2*A) COS(A/57.29578)	7/2
CSNG(*exp*)	Returns single-precision representation, with 5/4 rounding in least significant decimal when *exp* is double-precision.	CSNG(A#) CSNG(.33*B#)	7/2
EXP(*exp*)	Returns the natural exponential, $e^{exp} = \text{EXP}(exp)$.	EXP(34.5) EXP(A*B*C-1)	7/3
FIX(*exp*)	Returns the integer equivalent to truncated *exp* (fractional part of *exp* is chopped off).	FIX(A-B)	7/3
INT(*exp*)	Returns largest integer not greater than *exp*.	INT(A+B*C)	7/3
LOG(*exp*)	Returns natural logarithm (base e) of *exp*. Limits: *exp* must be positive.	LOG(12.33) LOG(A B+B)	7/3
RND(0)	Returns a pseudo-random number between 0.000001 and 0.999999 inclusive.	RND(0)	7/4
RND(*exp*)	Returns a pseudo-random number between 1 and INT(*exp*) inclusive. Limits: $1 <= exp < 32768$.	RND(40) RND(A+B)	7/4
SGN(*exp*)	Returns -1 for negative *exp*; 0 for zero *exp*; $+1$ for positive *exp*.	SGN(A*B+3) SGN(COS(X))	7/4

**exp* is any numeric-valued expression or constant.

Function	Operation	Examples	Section 2 Page
SIN(*exp*)	Returns the sine of *exp*; assumes *exp* is in radians.	SIN(A/B) SIN(90/57.29578)	7/5
SQR(*exp*)	Returns square root of *exp*. Limits: *exp* must be non-negative.	SQR(A*A – B*B)	7/5
TAN(*exp*)	Returns the tangent of *exp;* assumes *exp* is in radians.	TAN(X) TAN(X*.01745329)	7/5

Special Functions

Function	Operation and Limits	Examples	Section 2 Page
ERL	Returns line number of current error.	ERL	8/3
ERR	Returns a value related to current error code (if error has occurred). ERR = (error code – 1)*2. Also: (ERR/2) + 1 = error code.	ERR/2 + 1	8/3
INP(*port*)	Inputs and returns the current value from the specified *port*. Both argument and result are in the range 0 to 255 inclusive.	INP(55)	8/4
MEM	Returns total unused and unprotected bytes in memory. Does not include unused string storage space.	MEM	8/4
PEEK(*location*)	Returns value stored in the specified memory byte. *location* must be a valid memory address in decimal form (see Memory Map in Appendix D).	PEEK(15370)	8/4
POINT(*x,y*)	Checks the graphics block specified by horizontal coordinate *x* and vertical coordinate *y*. If block is "on", returns a True (– 1); if block is "off", returns a False (0). Limits: 0< =*x* <128;0< =*y*<48.		8/2
POS(0)	Returns a number indicating the current cursor position. The argument "0" is a dummy variable.	POS(0)	8/4
USR(*n*)	Branches to machine language subroutine. See Chapter 8.	USR(0)	8/7
VARPTR(*var*)	Returns the address where the specified variable's name, value and pointer are stored, *var* must be a valid variable name.	VARPTR(A$) VARPTR(N1)	8/9



Model III BASIC Reserved Words*

@	ELSE	LLIST	RENAME
ABS	END	LPRINT	RESET
AND	EOF	LOAD	RESTORE
ASC	ERL	LOC	RESUME
ATN	ERR	LOF	RETURN
AUTO	ERROR	LOG	RIGHT$
CDBL	EXP	MEM	RND
CHR$	FIELD	MERGE	RSET
CINT	FIX	MID$	RUN
CLEAR	FN	MKD$	SAVE
CLOCK	FOR	MKI$	SET
CLOSE	FORMAT	MKS$	SGN
CLS	FRE	NAME	SIN
CMD	FREE	NEW	SQR
CONT	GET	NEXT	STEP
COS	GOSUB	NOT	STOP
CSNG	GOTO	ON	STRING$
CVD	IF	OPEN	STR$
CVI	INKEY$	OR	SYSTEM
CVS	INP	OUT	TAB
DATA	INPUT	PEEK	TAN
DEFDBL	INSTR	POINT	THEN
DEFFN	INT	POKE	TIME$
DEFINT	KILL	POS	TO
DEFSNG	LEFT$	POSN	TROFF
DEFUSR	LET	PRINT	TRON
DEFSTR	LSET	PUT	USING
DELETE	LEN	RANDOM	USR
DIM	LINE	READ	VAL
EDIT	LIST	REM	VARPTR
			VERIFY

*Some of these words have no function in Model III BASIC; they are reserved for use in Disk BASIC. None of these words can be used inside a variable name. You'll get a syntax error if you try to use these words as variables.

Program Limits and Memory Overhead

Ranges

Integers −32768 to +32767 inclusive
Single Precision −1.701411E±38 to +1.701411E±38 inclusive
Double Precision −1.701411834544556D±38 to +1.701411834544556D±38 inclusive

String Range: Up to 255 characters

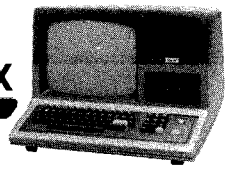
Line Numbers Allowed: 0 to 65529 inclusive

Program Line Length: Up to 255 characters (input 240, edit to 255)

Memory Overhead

Program lines require 5 bytes minimum, as follows:
- Line Number — 2 bytes
- Line Pointer — 2 bytes
- Carriage Return — 1 byte

In addition, each reserved word, operator, variable name, special character and constant character requires one byte.



Dynamic (RUN-Time) Memory Allocation

Integer variables: 5 bytes each
(2 for value, 3 for variable name)

Single-precision variables: 7 bytes each
(4 for value, 3 for variable name)

Double-precision variables: 11 bytes each
(8 for value, 3 for variable name)

String variables: 6 bytes minimum
(3 for variable name, 3 for stack and variable pointers, 1 for each character)

Array variables: 12 bytes minimum
(3 for variable name, 2 for total size, 1 for number of dimensions, 2 for size of each dimension, and 2, 3, 4 or 8 [depending on array type]
for each element in the array)

Each active FOR-NEXT loop requires 16 bytes.

Each active (non-returned) GOSUB requires 6 bytes.

Each level of parentheses requires 4 bytes plus 12 bytes for each temporary value.

General Formula for Computing Memory Requirements of Arrays

The array G (N1, N2, ..., Nk) requires the following amount of memory:

$$14 + (k*2) + T*\{(N1+1)*(N2+1)*...*(Nk+1)\}$$

where k is the number of dimensions in the array, and the value of T depends on the array type:

Type	T =
Integer	2
Single-Precision	4
Double-Precision	8
String*	3

*In computing the actual memory requirements of string arrays, you must add the text length of each element in the array. When the array is first dimensioned, all elements have length 0. The string text will be stored in the string space (reserved by the CLEAR n statement).

Accuracy

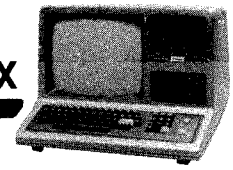
Single-precision calculations involving +, −, *, and / are accurate to six significant digits; double-precision calculations involving the same operations are accurate to 16 significant digits.

The exponentiation operator ↑ (displayed as "[") is single-precision.

The trigonometric and logarithmic functions are single-precision; other functions have a precision depending on the input argument and on the function. For example, CDBL returns a double-precision value; ABS returns a value with the same precision as the input argument.

When converting from single- to double-precision, use the following technique to avoid introduction of incorrect values in the extra digits of precision:

double-precision variable = VAL(STR$(*Single-precision variable*))



B / Error Codes

CODE	ABBREVIATION	ERROR
1	NF	NEXT without FOR
2	SN	Syntax error
3	RG	Return without GOSUB
4	OD	Out of data
5	FC	Illegal function call
6	OV	Overflow
7	OM	Out of memory
8	UL	Undefined line
9	BS	Subscript out of range
10	DD	Redimensioned array
11	/0	Division by zero
12	ID	Illegal direct
13	TM	Type mismatch
14	OS	Out of string space
15	LS	String too long
16	ST	String formula too complex
17	CN	Can't continue
18	NR	NO RESUME
19	RW	RESUME without error
20	UE	Unprintable error
21	MO	Missing operand
22	FD	Bad file data
23	L3	Disk BASIC only

Explanation of Error Messages

NF NEXT without FOR: NEXT is used without a matching FOR statement. This error may also occur if NEXT *variable* statements are reversed in a nested loop.

SN Syntax Error: This usually is the result of incorrect punctuation, open parenthesis, an illegal character or a mis-spelled command.

RG RETURN without GOSUB: A RETURN statement was encountered before a matching GOSUB was executed.

OD Out of Data. A READ or INPUT # statement was executed with insufficient data available. DATA statement may have been left out or all data may have been read from tape or DATA.

FC Illegal Function Call: An attempt was made to execute an operation using an illegal parameter. Examples: square root of a negative argument, negative matrix dimension, negative or zero LOG arguments, etc. Or USR call without first POKEing the entry point.

OV Overflow: The magnitude of the number input or derived is too large for the Computer to handle. NOTE: There is no underflow error. Numbers smaller than $\pm 1.701411E-38$ single precision or $\pm 1.701411834544556E-38$ double precision are rounded to 0. See /0 below.

OM Out of Memory: All available memory has been used or reserved. This may occur with very large matrix dimensions, nested branches such as GOTO, GOSUB, and FOR-NEXT Loops.

UL Undefined Line: An attempt was made to refer or branch to a non-existent line.

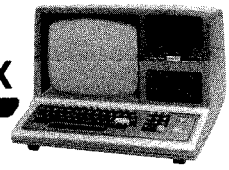
BS Subscript out of Range: An attempt was made to assign a matrix element with a subscript beyond the DIMensioned range.

DD Redimensioned Array: An attempt was made to DIMension a matrix which had previously been dimensioned by DIM or by default statements. It is a good idea to put all dimension statements at the beginning of a program.

/0 Division by Zero: An attempt was made to use a value of zero in the denominator. NOTE: If you can't find an obvious division by zero check for division by numbers smaller than allowable ranges. See OV above and RANGES page A/17.

ID Illegal Direct: The use of INPUT as a direct command.

TM Type Mismatch: An attempt was made to assign a non-string variable to a string or vice-versa.



OS Out of String Space: The amount of string space allocated was exceeded.

LS String Too Long: A string variable was assigned a string value which exceeded 255 characters in length.

ST String Formula Too Complex: A string operation was too complex to handle. Break up the operation into shorter steps.

CN Can't Continue: A CONT was issued at a point where no continuable program exists, e.g., after program was ENDed or EDITed.

NR No RESUME: End of program reached in error-trapping mode.

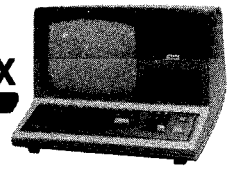
RW RESUME without ERROR: A RESUME was encountered before ON ERROR GOTO was executed.

UE Unprintable Error: An attempt was made to generate an error using an ERROR statement with an invalid code.

MO Missing Operand: An operation was attempted without providing one of the required operands.

FD Bad File Data: Data input from an external source (i.e., tape) was not correct or was in improper sequence, etc.

L3 DISK BASIC only: An attempt was made to use a statement, function or command which is available only with the Disk System.



C / TRS-80 Model III Character Codes

Text is represented in the Computer by codes. For example, the letter "A" is represented by the code 65. **Control functions and graphics** are also represented by codes. The character codes range from zero through 255.

Codes **zero through 31** usually represent certain **control functions.** For example, code 13 represents a carriage return or "end of line". However, in the Model III, these same codes also represent 32 special display characters. For this application, they must be loaded (POKEd) into video RAM, not PRINTed.

Codes **32 through 127** represent the **text characters** — all those letters, numbers and other characters that are commonly used to represent textual information. The Model III text characters conform to the American National Standard Code for Information Interchange.

Codes **128 through 191**, when output to the video display, represent 64 **graphics** characters.

Codes **192 through 255**, when output to the video display, represent either **space compression codes** or **special characters,** as determined by software.

Many of the codes may be input from the keyboard; all of them may be stored in a string and output to any device. For example, to output a code 31 to the video display, use a statement like this:

```
PRINT CHR$(31)
```

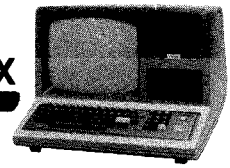
For further details, see Using the Video Display in Section One of this manual.

Note: In the following table, *vidram* refers to Video RAM, i.e., addresses from 15360 to 16383.

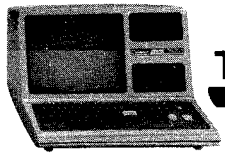
In the following table, we summarize the keyboard and video display control characters.

Code Dec.	Code Hex.	Keyboard	Video Display PRINT CHR$ *(code)*	POKE *vidram, code**
1	00		No effect	See **Special Characters** 0 through 31 later in this Appendix.
1	01	BREAK SHIFT ↑ A	No effect	
2	02	SHIFT ↑ B	No effect	
3	03	SHIFT ↑ C	No effect	
4	04	SHIFT ↑ D	No effect	
5	05	SHIFT ↑ E	No effect	
6	06	SHIFT ↑ F	No effect	
7	07	SHIFT ↑ G	No effect	
8	08	← SHIFT ↑ H	Backspace and erase	
9	09	→ SHIFT ↑ I	Tab (0, 8, 16, 24, . . .)	
10	0A	↓ SHIFT ↑ J	Move cursor to start of next line and erase line	
11	0B	SHIFT ↑ K	No effect	
12	0C	SHIFT ↑ L	No effect	
13	0D	ENTER SHIFT ↑ M	Move cursor to start of next line and erase line	
14	0E	SHIFT ↑ N	Cursor on	
15	0F	SHIFT ↑ O	Cursor off	
16	10	SHIFT ↑ P	No effect	
17	11	SHIFT ↑ Q	No effect	
18	12	SHIFT ↑ R	No effect	
19	13	SHIFT ↑ S	No effect	
20	14	SHIFT ↑ T	No effect	
21	15	SHIFT ↑ U	Swap space compression/ special characters	
22	16	SHIFT ↑ V	Swap special/alternate characters	
23	17	SHIFT ↑ W	Double-size characters	
24	18	SHIFT ← SHIFT ↑ X	Backspace without erasing	
25	19	SHIFT ↑ Y	Advance cursor	
26	1A	SHIFT ↑ Z	Move cursor down	
27	1B	SHIFT ↑	Move cursor up	
28	1C	SHIFT ↑ .	Move cursor to upper left corner	
29	1D	SHIFT ↑ 9	Erase line and start over	
30	1E	SHIFT ↑ -	Erase to end of line	
31	1F	CLEAR SHIFT ↑ /	Erase to end of display	

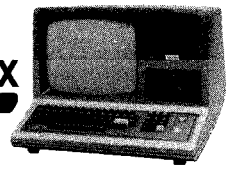
*See Special Characters 0 through 31 later in this Appendix.



Code		Key-board	Video Display	
Dec.	Hex.		PRINT CHR$ *(code)*	POKE *vidram, code*
32	20	(SPACEBAR)	ø	ø
33	21	!	!	!
34	22	"	"	"
35	23	#	#	#
36	24	$	$	$
37	25	%	%	%
38	26	&	&	&
39	27	'	'	'
40	28	(	(	(
41	29	)	)	)
42	2A	*	*	*
43	2B	+	+	+
44	2C	,	,	,
45	2D	–	–	–
46	2E	.	.	.
47	2F	/	/	/
48	30	0	0	0
49	31	1	1	1
50	32	2	2	2
51	33	3	3	3
52	34	4	4	4
53	35	5	5	5
54	36	6	6	6
55	37	7	7	7
56	38	8	8	8
57	39	9	9	9
58	3A	:	:	:
59	3B	;	;	;
60	3C	<	<	<
61	3D	=	=	=
62	3E	>	>	>
63	3F	?	?	?
64	40	@	@	@
65	41	A	A	A
66	42	B	B	B
67	43	C	C	C
68	44	D	D	D

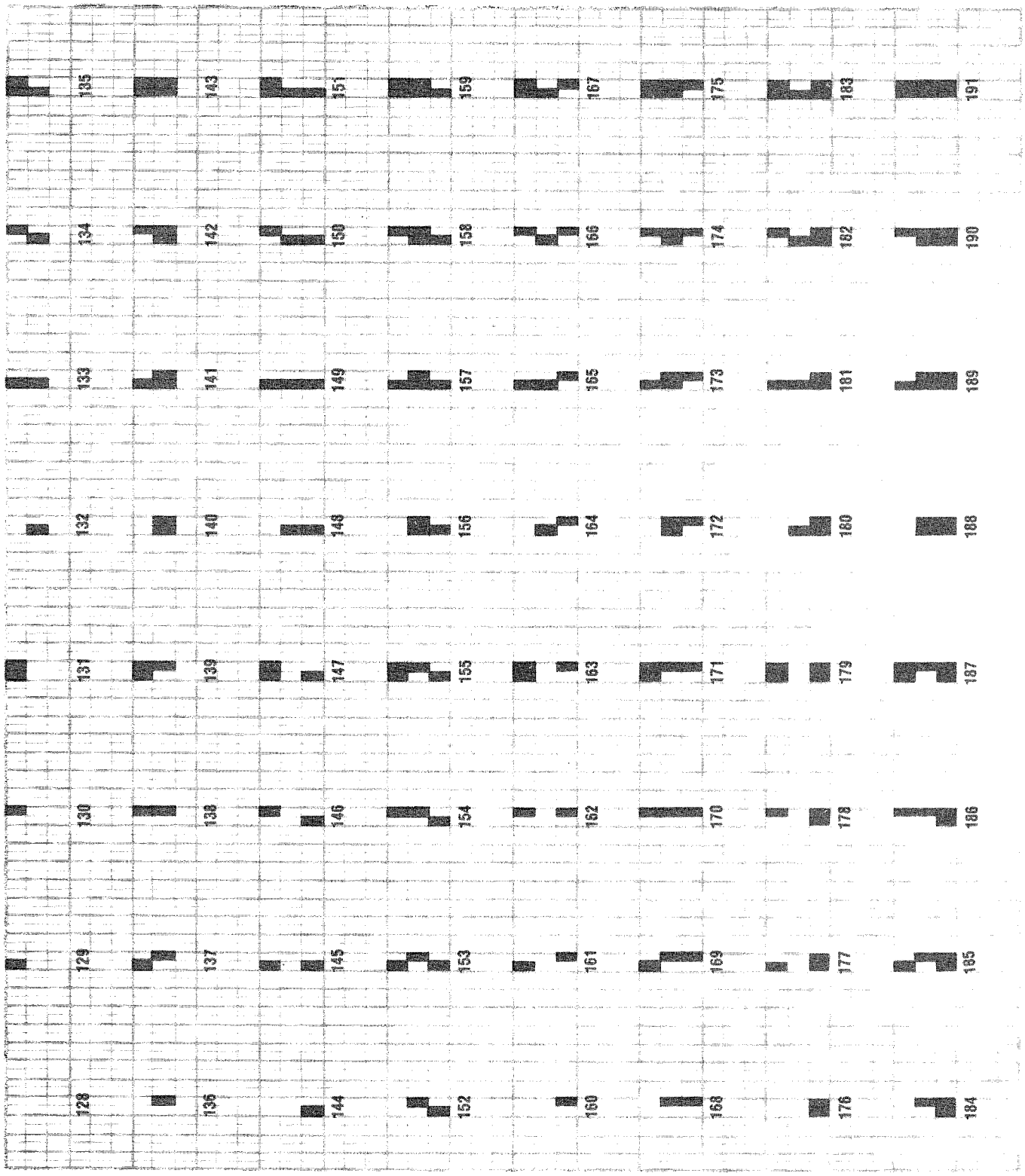


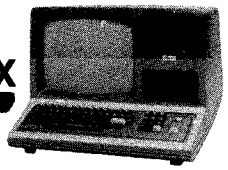
Code		Key-board	Video Display	
Dec.	Hex.		PRINT CHR$ *(code)*	POKE *vidram, code*
69	45	E	E	E
70	46	F	F	F
71	47	G	G	G
72	48	H	H	H
73	49	I	I	I
74	4A	J	J	J
75	4B	K	K	K
76	4C	L	L	L
77	4D	M	M	M
78	4E	N	N	N
79	4F	O	O	O
80	50	P	P	P
81	51	Q	Q	Q
82	52	R	R	R
83	53	S	S	S
84	54	T	T	T
85	55	U	U	U
86	56	V	V	V
87	57	W	W	W
88	58	X	X	X
89	59	Y	Y	Y
90	5A	Z	Z	Z
91	5B	↑	[	[
92	5C		\	\
93	5D		]	]
94	5E		^	^
95	5F		—	—
96	60	SHIFT @	`	`
97	61	A	a	a
98	62	B	b	b
99	63	C	c	c
100	64	D	d	d
101	65	E	e	e
102	66	F	f	f
103	67	G	g	g
104	68	H	h	h
105	69	I	i	i



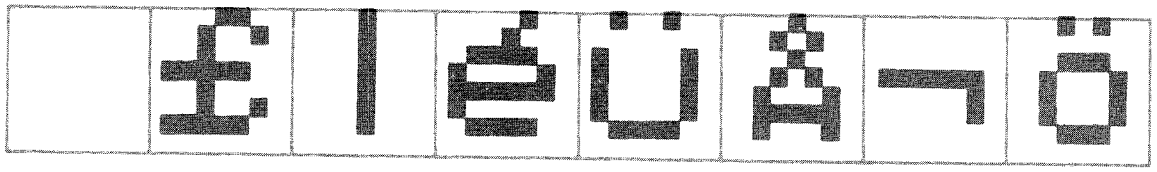
Code		Key-board	Video Display	
Dec.	Hex.		PRINT CHR$ *(code)*	POKE *vidram, code*
106	6A	J	j	j
107	6B	K	k	k
108	6C	L	l	l
109	6D	M	m	m
110	6E	N	n	n
111	6F	O	o	o
112	70	P	p	p
113	71	Q	q	q
114	72	R	r	r
115	73	S	s	s
116	74	T	t	t
117	75	U	u	u
118	76	V	v	v
119	77	W	w	w
120	78	X	x	x
121	79	Y	y	y
122	7A	Z	z	z
123	7B		{	{
124	7C		¦	¦
125	7D		}	}
126	7E		~	~
127	7F		±	±
128	80	Codes 128-191 output graphics characters. See the graphic display table in this Appendix.		
192 : :	C0	Codes 192-255 output either space compression codes or special characters when used with PRINT CHR$ *(code)*.		
255	FF	They always output special characters when used with POKE *vidram, code*. See the special character table in this Appendix.		

Graphics Characters (Codes 128-191)

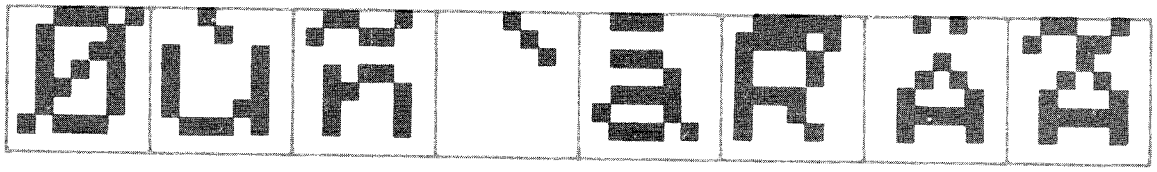




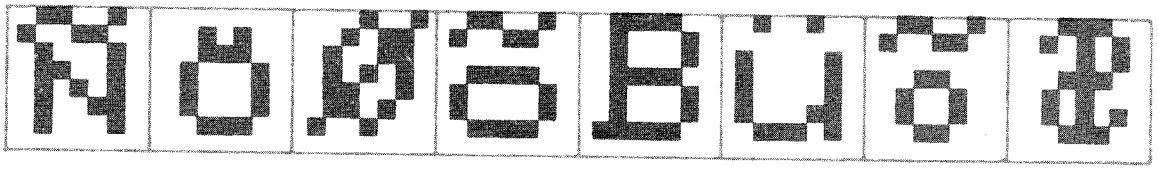
Special Characters (0-31, 192-255)



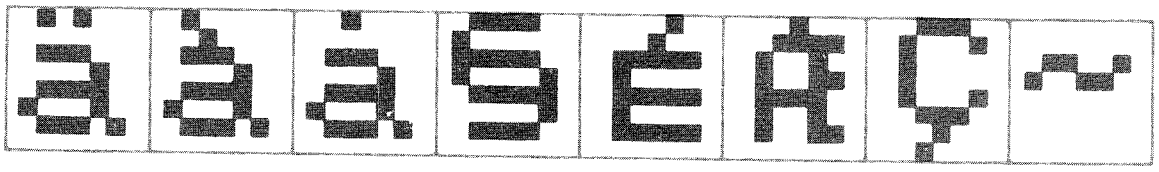
0 1 2 3 4 5 6 7



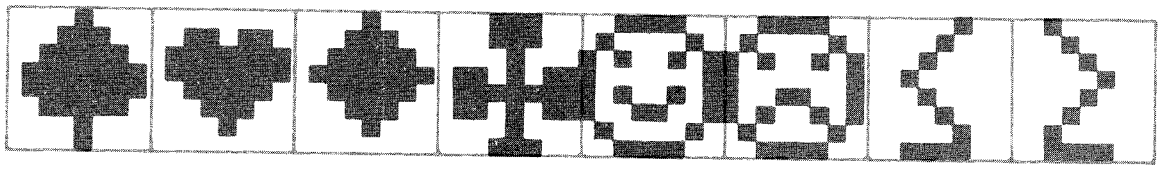
8 9 10 11 12 13 14 15



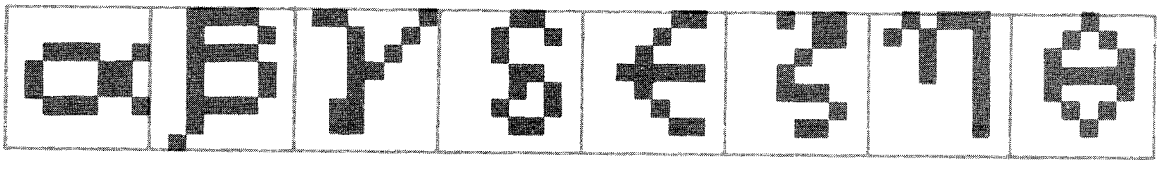
16 17 18 19 20 21 22 23



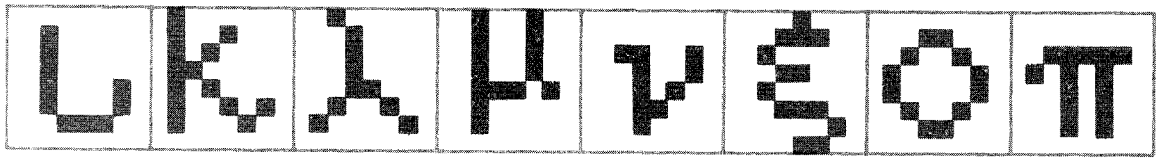
24 25 26 27 28 29 30 31



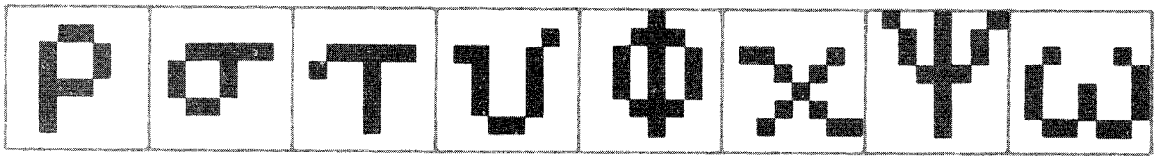
192 193 194 195 196 197 198 199



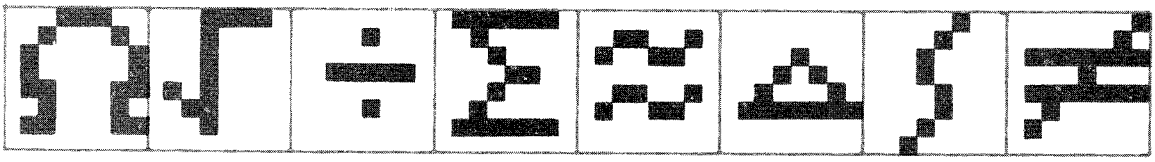
200 201 202 203 204 205 206 207



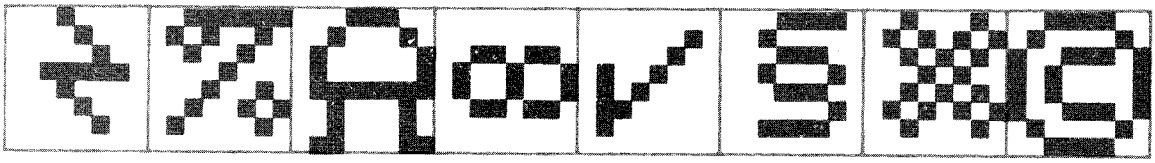
208 209 210 211 212 213 214 215



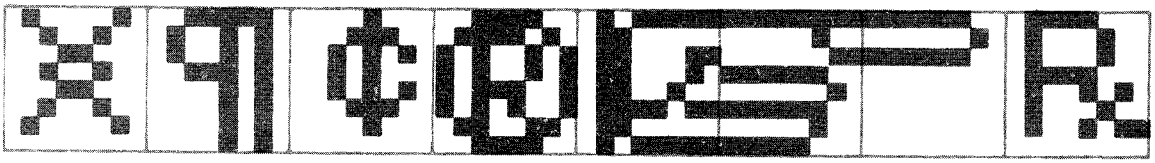
216 217 218 219 220 221 222 223



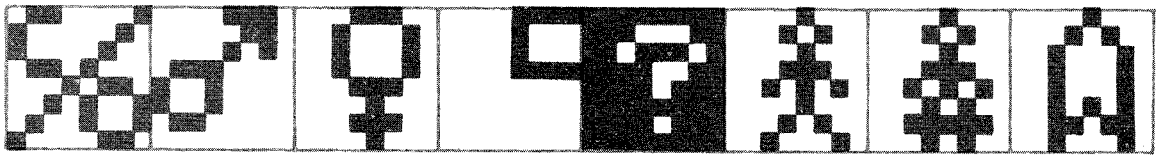
224 225 226 227 228 229 230 231



232 233 234 235 236 237 238 239



240 241 242 243 244 245 246 247

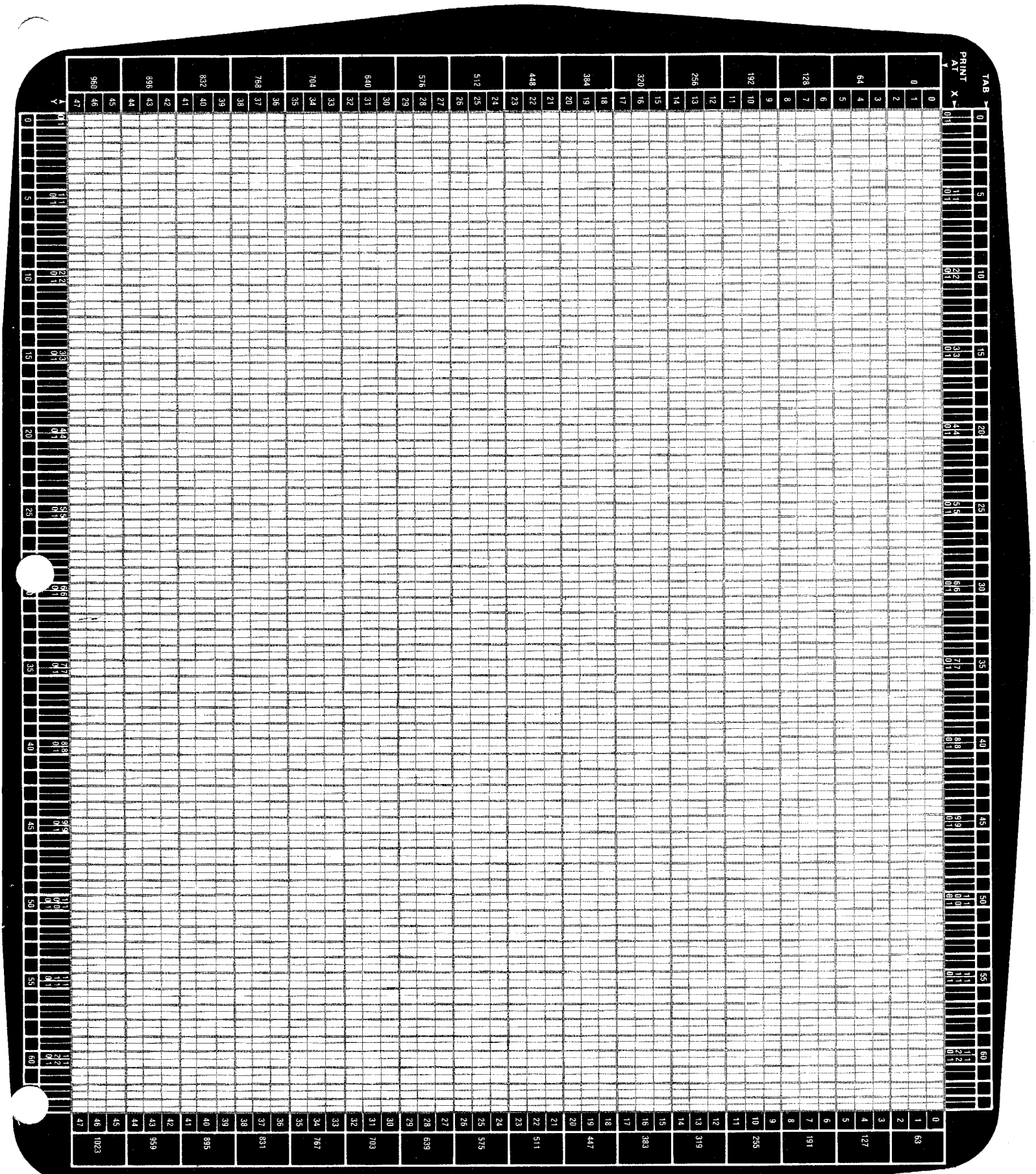


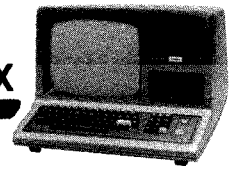
248 249 250 251 252 253 254 255

TRS-80 Video Display Worksheet

Radio Shack®

TITLE _____ PROGRAMMER _____ PAGE ____ OF ____



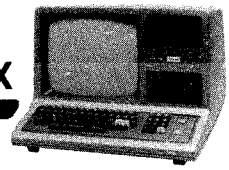


D / Internal Codes for BASIC Keywords

The following are the internal codes that the Computer uses to store BASIC keywords. If you PEEK at the program buffer area (starting at address 17129 in decimal) you will find your program stored in the following codes.

Dec. Code	BASIC Keyword	Dec. Code	BASIC Keyword
129	FOR	167	LOAD
130	RESET	168	MERGE
131	SET	169	NAME
132	CLS	170	KILL
133	CMD	171	LSET
134	RANDOM	172	RSET
135	NEXT	173	SAVE
136	DATA	174	SYSTEM
137	INPUT	175	LPRINT
138	DIM	176	DEF
139	READ	177	POKE
140	LET	178	PRINT
141	GOTO	179	CONT
142	RUN	180	LIST
143	IF	181	LLIST
144	RESTORE	182	DELETE
145	GOSUB	183	AUTO
146	RETURN	184	CLEAR
147	REM	185	CLOAD
148	STOP	186	CSAVE
149	ELSE	187	NEW
150	TRON	188	TAB
151	TROFF	189	TO
152	DEFSTR	190	FN
153	DEFINT	191	USING
154	DEFSNG	192	VARPTR
155	DEFDBL	193	USR
156	LINE	194	ERL
157	EDIT	195	ERR
158	ERROR	196	STRING$
159	RESUME	197	INSTR
160	OUT	198	POINT
161	ON	199	TIME$
162	OPEN	200	MEM
163	FIELD	201	INKEY$
164	GET	202	THEN
165	PUT	203	NOT
166	CLOSE	204	STEP

Dec. Code	BASIC Keyword	Dec. Code	BASIC Keyword
205	+	231	CVS
206	–	232	CVD
207	*	233	EOF
208	/	234	LOC
209		235	LOF
210	AND	236	MKI$
211	OR	237	MKS$
212	>	238	MKD$
213	=	239	CINT
214	<	240	CSNG
215	SGN	241	CDBL
216	INT	242	FIX
217	ABS	243	LEN
218	FRE	244	STR$
219	INP	245	VAL
220	POS	246	ASC
221	SQR	247	CHR$
222	RND	248	LEFT$
223	LOG	249	RIGHT$
224	EXP	250	MID$
225	COS		
226	SIN		
227	TAN		
228	ATN		
229	PEEK		
230	CVI		



E / Derived Functions

Function	Function Expressed in Terms of Model III BASIC Functions. X is in radians.
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X) = -ATN(X/SQR(-X*X+1))+1.5708
INVERSE SECANT	ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COTANGENT	ARCCOT(X) = -ATN(X)+1.5708
HYPERBOLIC SINE	SINH(X) = (EXP(X)-EXP(-X))/2
HYPOBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1
INVERSE HYPERBOLIC SINE	ARGSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARGCOSH(X) = LOG(X+SQR(X*X-1))
INVERSE HYPERBOLIC TANGENT	ARGTANH(X) = LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARGSECH(X) = LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARGCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)
INVERSE HYPERBOLIC COTANGENT	ARGCOTH(X) = LOG((X+1)/(X-1))/2

Valid Input Ranges

Inverse Sine	$-1 < X < 1$
Inverse Cosine	$-1 < X < 1$
Inverse Secant	$X < -$ or $X > 1$
Inverse Cosecant	$X < -1$ or $X > 1$
Inverse Hyper. Cosine	$X > 1$
Inverse Hyper. Tangent	$X*X < 1$
Inverse Hyper. Secant	$0 < X < 1$
Inverse Hyper. Cosecant	$X <> 0$
Inverse Hyper. Cotangent	$X*X > 1$

Certain special values are mathematically undefined, but our functions may provide invalid values:

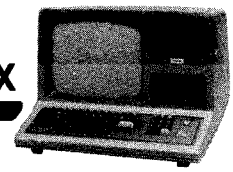
TAN and SEC of 90 and 270 degrees
COT and CSC of 0 and 180 degrees

For example, TAN(1.5708) returns a value but TAN(90* .01745329) returns a DIVISION BY ZERO error. 90* .01745329 = 1.5708

Other values which are not available from these functions are:

ARSCIN (−1) = − PI / 2
ARCSIN (1) = PI / 2
ARCCOS (−1) = PI
ARCCOS (1) = 0
ARCSEC (−1) = − PI
ARCSEC (1) = 0
ARCCSC (−1) = − PI / 2
ARCCSC (1) = PI / 2

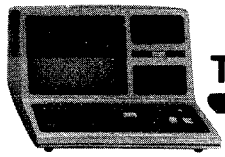
Please note that the above information may not be exhaustive.



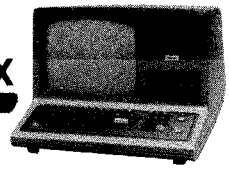
F / Base Conversions

DEC.	HEX.	BINARY
0	00	00000000
1	01	00000001
2	02	00000010
3	03	00000011
4	04	00000100
5	05	00000101
6	06	00000110
7	07	00000111
8	08	00001000
9	09	00001001
10	0A	00001010
11	0B	00001011
12	0C	00001100
13	0D	00001101
14	0E	00001110
15	0F	00001111
16	10	00010000
17	11	00010001
18	12	00010010
19	13	00010011
20	14	00010100
21	15	00010101
22	16	00010110
23	17	00010111
24	18	00011000
25	19	00011001
26	1A	00011010
27	1B	00011011
28	1C	00011100
29	1D	00011101
30	1E	00011110
31	1F	00011111
32	20	00100000
33	21	00100001
34	22	00100010
35	23	00100011
36	24	00100100
37	25	00100101
38	26	00100110
39	27	00100111

DEC.	HEX.	BINARY
40	28	00101000
41	29	00101001
42	2A	00101010
43	2B	00101011
44	2C	00101100
45	2D	00101101
46	2E	00101110
47	2F	00101111
48	30	00110000
49	31	00110001
50	32	00110010
51	33	00110011
52	34	00110100
53	35	00110101
54	36	00110110
55	37	00110111
56	38	00111000
57	39	00111001
58	3A	00111010
59	3B	00111011
60	3C	00111100
61	3D	00111101
62	3E	00111110
63	3F	00111111
64	40	01000000
65	41	01000001
66	42	01000010
67	43	01000011
68	44	01000100
69	45	01000101
70	46	01000110
71	47	01000111
72	48	01001000
73	49	01001001
74	4A	01001010
75	4B	01001011
76	4C	01001100
77	4D	01001101
78	4E	01001110
79	4F	01001111

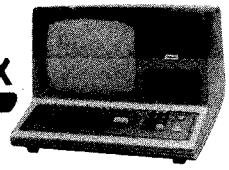


DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
80	50	01010000	120	78	01111000
81	51	01010001	121	79	01111001
82	52	01010010	122	7A	01111010
83	53	01010011	123	7B	01111011
84	54	01010100	124	7C	01111100
85	55	01010101	125	7D	01111101
86	56	01010110	126	7E	01111110
87	57	01010111	127	7F	01111111
88	58	01011000	128	80	10000000
89	59	01011001	129	81	10000001
90	5A	01011010	130	82	10000010
91	5B	01011011	131	83	10000011
92	5C	01011100	132	84	10000100
93	5D	01011101	133	85	10000101
94	5E	01011110	134	86	10000110
95	5F	01011111	135	87	10000111
96	60	01100000	136	88	10001000
97	61	01100001	137	89	10001001
98	62	01100010	138	8A	10001010
99	63	01100011	139	8B	10001011
100	64	01100100	140	8C	10001100
101	65	01100101	141	8D	10001101
102	66	01100110	142	8E	10001110
103	67	01100111	143	8F	10001111
104	68	01101000	144	90	10010000
105	69	01101001	145	91	10010001
106	6A	01101010	146	92	10010010
107	6B	01101011	147	93	10010011
108	6C	01101100	148	94	10010100
109	6D	01101101	149	95	10010101
110	6E	01101110	150	96	10010110
111	6F	01101111	151	97	10010111
112	70	01110000	152	98	10011000
113	71	01110001	153	99	10011001
114	72	01110010	154	9A	10011010
115	73	01110011	155	9B	10011011
116	74	01110100	156	9C	10011100
117	75	01110101	157	9D	10011101
118	76	01110110	158	9E	10011110
119	77	01110111	159	9F	10011111



DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
160	A0	10100000	200	C8	11001000
161	A1	10100001	201	C9	11001001
162	A2	10100010	202	CA	11001010
163	A3	10100011	203	CB	11001011
164	A4	10100100	204	CC	11001100
165	A5	10100101	205	CD	11001101
166	A6	10100110	206	CE	11001110
167	A7	10100111	207	CF	11001111
168	A8	10101000	208	D0	11010000
169	A9	10101001	209	D1	11010001
170	AA	10101010	210	D2	11010010
171	AB	10101011	211	D3	11010011
172	AC	10101100	212	D4	11010100
173	AD	10101101	213	D5	11010101
174	AE	10101110	214	D6	11010110
175	AF	10101111	215	D7	11010111
176	B0	10110000	216	D8	11011000
177	B1	10110001	217	D9	11011001
178	B2	10110010	218	DA	11011010
179	B3	10110011	219	DB	11011011
180	B4	10110100	220	DC	11011100
181	B5	10110101	221	DD	11011101
182	B6	10110110	222	DE	11011110
183	B7	10110111	223	DF	11011111
184	B8	10111000	224	E0	11100000
185	B9	10111001	225	E1	11100001
186	BA	10111010	226	E2	11100010
187	BB	10111011	227	E3	11100011
188	BC	10111100	228	E4	11100100
189	BD	10111101	229	E5	11100101
190	BE	10111110	230	E6	11100110
191	BF	10111111	231	E7	11100111
192	C0	11000000	232	E8	11101000
193	C1	11000001	233	E9	11101001
194	C2	11000010	234	EA	11101010
195	C3	11000011	235	EB	11101011
196	C4	11000100	236	EC	11101100
197	C5	11000101	237	ED	11101101
198	C6	11000110	238	EE	11101110
199	C7	11000111	239	EF	11101111

DEC.	HEX.	BINARY
240	F0	11110000
241	F1	11110001
242	F2	11110010
243	F3	11110011
244	F4	11110100
245	F5	11110101
246	F6	11110110
247	F7	11110111
248	F8	11111000
249	F9	11111001
250	FA	11111010
251	FB	11111011
252	FC	11111100
253	FD	11111101
254	FE	11111110
255	FF	11111111



G / Model I to Model III Program Conversion Hints

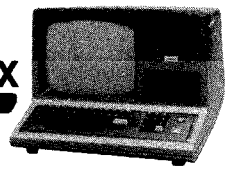
From a language standpoint, Model III BASIC is fully compatible with Model I Level II BASIC. In fact, the two BASIC's are identical, except that Model III BASIC includes one more function, TIME$.

However, because of Model III's many special features not available in Model I, there are some internal differences which may require that you modify any Model I Level II BASIC programs you may have.

1. For a given TRS-80 (16K, 32K or 48K RAM), the amount of user memory in Model III is 258 bytes less than the amount in Model I.
2. To load a Level II BASIC program, you must select the Low (500 baud) cassette speed on your Model III.
3. When running a Level II BASIC program which requires all-capitals keyboard entries, be sure to select all-caps mode. (SHIFT)(0) is the on/off toggle for all-caps.
4. Unlike the Model I, Model III lets you interrupt a cassette, line printer, or RS-232-C operation by holding down the (BREAK) key. Some of your Level II programs may need modification to take this feature into account.
5. The video display character sets are slightly different in Model I and Model III. Model III produces standard ASCII characters for codes 32 through 127; Model I does not. In particular, there is no up arrow, down arrow, left arrow or right arrow in the Model III character set. However, Model III has an additional set of 96 special characters from which you can probably find whatever you need. See the table of Model III Character Codes for details.

Radio Shack Applications Programs

For a list of which Model I programs will run on Model III and which won't, see the Radio Shack Computer Catalog. Most Model I-only programs will be available in Model III versions. Check at your local Radio Shack.



H / Glossary

address A location in memory, usually specified as a two-byte hexadecimal number. The address range [0 to FFFF] is represented in decimal as [0 to 32767] [−32768, . . . , −1].

alphabetic Referring strictly to the letters A to Z.

alphanumeric Referring to the set of letters A to Z and the numerals 0-9.

argument The string or numeric quantity which is supplied to a function and is then operated on to derive a result; this result is referred to as the **value** of the function.

array An organized set of elements which can be referenced in total or individually, using the array name and one or more subscripts. In BASIC, any variable name can be used to name an array; and arrays can have one or more dimensions. AR( ) signifies a one-dimensional array named AR; AR(,) signifies a two-dimensional array named AR; etc.

ASCII American Standard Code for Information Interchange. This method of coding is used to store textual data. Numeric data is typically stored in a more compressed format.

BASIC Beginners' All-purpose Symbolic Instruction Code.

binary Having two possible states, e.g., the binary digits 0 and 1. The binary (base 2) numbering system uses sequences of zeroes and ones to represent quantities. This is analagous to the Computer's internal representation of data, using electrical values for 0 and 1.

bit Binary digit; the smallest unit of memory in the Computer, capable of representing the values 0 and 1.

break To interrupt execution of a program. In BASIC the statement STOP causes a break in execution, as does pressing the (BREAK) key.

buffer An area in RAM where data is accumulated for further processing.

byte The smallest addressable unit of memory in the Computer, consisting of 8 consecutive bits, and capable of representing 256 different values, e.g., decimal values from 0 to 255.

compressed-format A method of storing information in less space than a standard ASCII representation would require. An integer always requires two bytes; a single-precision number, four; a double-precision number, 8 — regardless of how many characters are required to represent the numbers as text. String values are not stored in compressed format; each character requires one byte.

BASIC programs in RAM are stored in compressed-format, with all BASIC keywords stored as special one-byte codes.

data Information that is passed to or output from a program. There are four types of data:

- Integer numbers
- Single-precision numbers
- Double-precision numbers
- Character-string sequences (strings)

debug To find and remove logical or syntactic errors from a program.

decimal Capable of assuming one of ten states, e.g., the decimal digits 0,1, . . . ,9. Decimal (base 10) numbering is the everyday system, using sequences of decimal digits. Decimal numbers are stored in binary code in Model III BASIC.

default An action or value which is supplied by a program when you do not specify an action or value to be used.

delimiter A character which marks the beginning or end of a data item, and is not a part of the data. For example, the double-quote symbol is a string delimiter to BASIC.

device A physical part of the computer system used for data I/O, e.g., keyboard, display, or line printer.

diskette A magnetic recording medium for mass data storage.

dummy variable A variable name which is used in an expression to meet syntactic requirements, but whose value is insignificant.

edit To change existing information.

entry point The address of a machine-language program or routine where execution is to begin. This is not necessarily the same as the starting address. Entry point is also referred to as the **transfer address.**

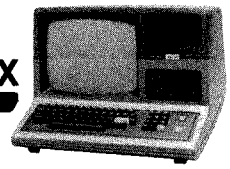
hexadecimal or **hex** Capable of existing in one of 16 possible states. For example, the hexadecimal digits are 0,1,2, . . . ,9,A,B,C,D,E,F. Hexadecimal (base-16) numbers are sequences of hexadecimal digits. Address and byte values are frequently given in hexadecimal form. In Model III BASIC, hexadecimal constants can be input by prefixing the constant with &H.

increment The value which is added to a counter each time one cycle of a repetitive procedure is completed.

input To transfer data from outside the Computer (from a cassette file, keyboard, etc.) into RAM.

kilobyte or **K** 1024 bytes of memory. Thus a 64K System includes 64*1024=65536 bytes of memory.

logical expression An expression which is evaluated as either TRUE (= – 1) or FALSE (=0).



machine language The Z-80A instruction set, usually specified in hexadecimal code. All higher-level languages must be translated into machine-language, or interpreted by machine language, in order to be executed by the Computer.

null string A string which has a length of zero. For example, the assignment A$ = "" makes A$ a null string.

object code Machine language derived from "source code", typically, from assembly language.

octal Capable of existing in one of eight states, for example, the octal digits are 0, 1, . . . ,7. Octal (base-8) numbers are sequences of octal digits. Address and byte values are frequently given in octal form. Under Model III BASIC, an octal constant can be input by prefixing the octal number with the symbol &O.

output To transfer data from inside the Computer's memory to some external area, e.g., a disk file or a line printer.

parameter Information supplied with a command to specify how the command is to operate.

prompt A character or message provided by a program to indicate that it's ready to accept keyboard input.

random access memory or **RAM** Semiconductor memory which can be addressed directly and either read from or written to.

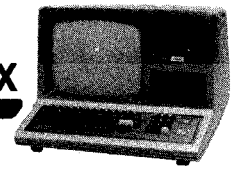
routine A sequence of instructions to carry out a certain function; typically, a routine called from multiple points in a program.

statement A complete instruction in BASIC.

string Any sequence of characters which must be examined verbatim for meaning: in other words, the string does not correspond to a quantity. For example, the *number* 1234 represents the same quantity as 1000+234, but the *string* "1234" does not. (String addition is actually concatenation, or stringing-together, so that: "1234" equals "1" + "2" + "3" + "4").

syntax The "grammatical" requirements for a command or statement. Syntax generally refers to punctuation and ordering of elements within a statement.

transfer address See **entry point.**



I / RS-232-C Technical Information

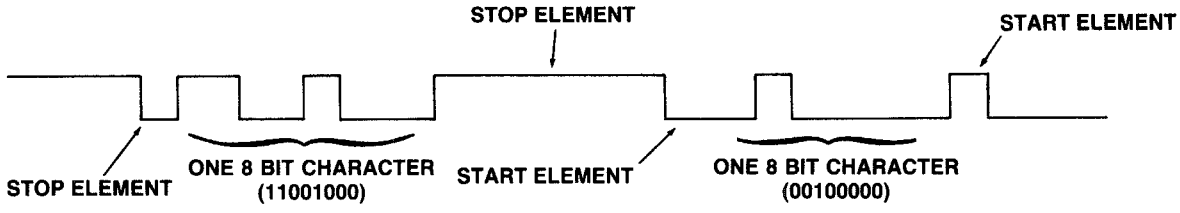
Transmission of Digital Data

The transfer of digital data over relatively long distances is generally accomplished by sending data in serial form using a single twisted wire pair to connect the transmitting and receiving devices. One of two general transmission techniques is commonly used, asynchronous or synchronous. The transmission technique used in the Radio Shack system is asynchronous-bit-serial. Since we don't use the synchronous technique, we'll not mention it again. Asynchronous transmission does not require a synchronizing clock to be transmitted with the data and, the characters need not be contiguous. This means that gaps of varying lengths may be present between transmission of individual characters.

The bits which comprise a data character (generally from five to eight bits in length) and synchronizing start and stop elements are added to each character as shown below. The start element is a single logic zero (0) data bit that is added to the front
character. The stop element is maintained until the start element of the next character is transmitted. There is no upper limit to the length of the stop element. However, there is a lower limit that depends on system characteristics. Typical lower limits are 1.0, 1.42 or 2.0 data-bit intervals (although most modern systems use 1.0 or 2.0 stop bits). The negative-going transition of the start element defines the location of the data bits in the character being transmitted. A clock source at the receiver is reset by this transition and is used to locate the center of each data bit.

There are several good reasons for using the asynchronous data transmission system. A clock signal does not need to be transmitted with the data, thus, equipment is simpler. Also, the characters don't need to be sent all at one time; they can be transmitted as they become available. This is particularly useful when transmitting data from manual-entry input devices (e.g. a keyboard). The major disadvantage of asynchronous transmission is that it requires a significant portion of the communications bandwidth for start and stop elements.

The rate at which asynchronous data is transmitted is defined as the **baud rate.** Baud rate is the inverse of the time duration of the shortest signal element. Normally, this is one data bit interval. The baud rate is equal to the bit rate if one stop bit is used; but for systems which use more than one stop bit, the baud rate does not equal the bit rate.



Asynchronous Data

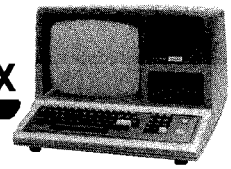
Asynchronous transmission over a simple twisted wire pair can be accomplished at moderately high baud rates (10K baud or higher, depending on the length of wire, type of drivers, etc.). Transmission over the telephone network is generally limited to approximately 2K baud and a modem is required to convert the data pulses to tones that can be transmitted through the telephone network. Radio Shack's Telephone Interface is the ideal modem for this RS-232-C Interface.

Signal Conventions

The E.I.A. RS-232-C electrical specification defines voltage levels and corresponding logic conventions associated with data and control information transmitted between equipment. For data interchange, the signal is considered in the **marking** condition when the voltage measured at the interface point is more negative than −3 Volts(with respect to signal ground). The signal is considered in the **spacing** condition when the voltage is more positive than +3 Volts(with respect to signal ground). The marking condition corresponds to a logic one (1) and the space condition corresponds to a logic zero (0). For timing and control interchange circuits, the function is considered to be "on" when the voltage on the interchange circuit is more positive than +3 Volts(with respect to signal ground); and is considered to be "off" when the voltage is more negative than −3 Volts(with respect to signal ground). The "on" condition corresponds to a logic zero (0) and the "off" condition corresponds to a logic one (1). The following table summarizes this information.

NOTATION	INTERCHANGE VOLTAGE	
	Negative	Positive
Binary State	1	0
Signal Condition	Marking	Spacing
Function	OFF	ON

Table. On/Off Condition



Pin Designations and Signal Descriptions

The mechanical specification of the RS-232-C requires a 25-pin connector (called a DB-25). The following table specifies the pin assignments and signal descriptions as they apply to the Radio Shack RS-232-C Interface.

Pin Number	Abbreviation	Description
1	PGND	Protective Ground
2	TD	Transmit Data
3	RD	Receive Data
4	RTS	Request-to-Send
5	CTS	Clear-to-Send
6	DSR	Data Set Ready
7	SGND	Signal Ground
8	CD	Carrier Detect
14	STD	Secondary Transmit Data
18	SUN	Secondary Unassigned
19	SRTS	Secondary Request-to-Send
20	DTR	Data Terminal Ready
22	RI	Ring Indicator

Table 2. Pin Designations and Signal Description

Protective Ground: This must be bonded to the chassis or equipment frame. It may also be connected to Signal Ground.

Transmit Data: Direction-to data communication equipment. Signals on this circuit are generated by the data terminal equipment for transmission of data to remote equipment. This signal should be held in the marking condition during intervals between characters and at all times when no data is being transmitted.

Received Data: Direction-from data communication equipment. Signals on this circuit are received from remote equipment which transmits data to the terminal. This signal should be held in the marking condition during intervals between characters and at all times when no data is being received.

Request-to-send: Direction-to data communication equipment. This signal is required by the terminal equipment to control the direction of data transmission by the data communication equipment. On one-way or duplex channels, the "on" condition maintains the data communication equipment in the transmit mode. The "off" condition maintains the data communication equipment in the non-transmit mode.

On a half duplex channel, the "on" condition maintains the data communication equipment in the transmit mode and inhibits the receive mode. The "off" condition maintains the data communication equipment in the receive mode.

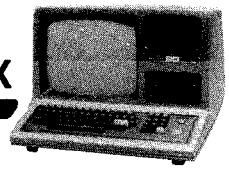
Clear-to-Send: Direction-from data communication equipment. This signal is generated by the data communication equipment and indicates whether or not the data set (modem) is ready to transmit data. The "on" condition is an indication to the data terminal equipment that the data set can accept data on the Transmit Data circuit. The "off" condition is an indication to the data terminal equipment that it should not transfer data to the data set.

Data Set Ready: Direction-from data communication equipment. This signal indicates the status of the local data set to the data terminal equipment. The "on" condition of this circuit indicates that the data communication equipment is not in test, talk or dial mode and has completed any timing functions required to complete call establishment (answer tone, etc.). The "off" condition will appear at all other times and indicates that the data terminal should accept only Ring Indicator signals and ignore all other signals (appearing on any other interchange circuit).

Data Terminal Ready: Direction-to data communication equipment. This signal is used to control the switching of the data communication equipment to the communications channel. The "on" condition indicates to data communication equipment that it should connect to the communications channel and that it should maintain the connection as long as the "on" condition is present. The "off" condition causes the data communication equipment to be removed from the communications channel following any in-process transmission of data.

Ring Indicator: Direction-from communication equipment. The "on" condition of the circuit indicates that a ringing signal is being received on the communications channel. In general, this means that the data set is being polled and that data communication is desired by the polling device. The "off" condition is held during the off segment of the ringing cycle (between actual rings) and at all other times when ringing is not being received.

Carrier Detect (Receive Line Signal Detector): Direction-from data communication equipment. When "on", this signal indicates that the data set is receiving a carrier from a remote data set via the communications channel. The "off" condition indicates that no carrier is being received or that the signal quality is unsuitable for data demodulation.



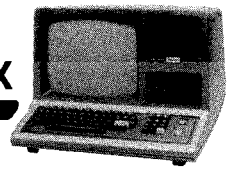
Index

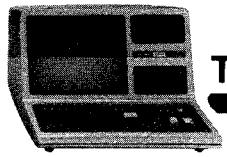
The prefix "Op" means "Operation Section"; "Ba" means "BASIC Section". Pages referenced by a letter/number are in the Appendices.

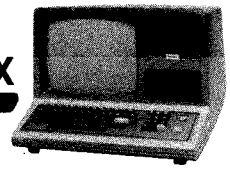
Examples:

Op 3/4 - 8	Operation, Chapter 3, pages 4 through 8
Ba 2/1, 8/3	BASIC, Chapter 2, page 8; Chapter 8, page 3
A/1, 20	Appendix A, pages 1 and 20

Page references in **boldface** indicate the most important information for a particular index entry.







Figures and Tables

Customer Information

Service Policy

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.

2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.

3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.