

Cat. No. 26-3194

\$4.95

Radio Shack®



TRS-80®

COLOR

COMPUTER

QUICK

REFERENCE

GUIDE

RADIO SHACK, A DIVISION OF TANDY CORPORATION

**U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5**

TANDY CORPORATION

AUSTRALIA

91 KURRAJONG ROAD
MOUNT DRUITT, N.S.W. 2770

BELGIUM

PARC INDUSTRIEL DE NANINNE
5140 NANINNE

U. K.

BILSTON ROAD WEDNESBURY
WEST MIDLANDS WS10 7JN

04/85-SP

Printed in U.S.A.

Contents

Start-Up	3
Statements	5
Functions	15
ROM Subroutines	19
Codes, Keys, and Abbreviations	21
Control Keys — Special Characters — Operators	21
Video Control Codes — Color Codes — Graphic Character Codes	23
ASCII Characters	25
Color Sets — Resolution Table — Musical Note/Number Table	27
Editor Commands	29
Assembler Switches	31
ZBUG Commands	33
Special Symbols	35
Mode Commands	37
6809 Instructions	39
Editor Assembler Error Messages	47
Memory Map	49
Error Messages	51
Line Printer Variables	53
Color Adjustment Test	55
Video Centering Test	57
Specifications	59
Definitions & Index	63

*TRS-80® Color Computer
Quick Reference Guide*

©1982 Tandy Corporation

All Rights Reserved

Reproduction or use, without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information obtained herein.

Start-Up

1. Turn the television set ON.
2. Select channel 3 or 4.
3. Set the Antenna Switch to "COMPUTER."
4. If you're using a Program Pak™, insert it now, before turning on the Computer.
5. Turn the Computer ON.
6. Turn any accessory equipment (e.g., a printer) ON.
7. If you're not using a Program Pak™, the Color BASIC or Extended Color BASIC start-up message will appear on the TV, followed by: DK

The Computer is now ready to use.

Information which is non-shaded (like this) pertains to both Extended and non-Extended Color BASIC.

Information pertaining to Extended Color BASIC *only* is shaded like this paragraph.

Information which is *shaded like this* pertains to Disk Extended Color BASIC.

All information stored on disk must be assigned a *filename*. *Filenames* must be in the following format:

filename/ext:d

filename is any combination of from one to eight characters. This is not optional.

ext is the filename extension and can be from one to three characters. This is optional; if omitted, the extension BAS is assigned. *:d* is the disk drive specification. This is usually optional; if omitted. Drive 0 is used (or whatever drive was previously specified as the Master Drive using the command DRIVE).

Statements

AUDIO Connects or disconnects cassette output to TV speaker.

```
AUDIO ON
AUDIO OFF
```

BACKUP *source TO destination* Creates an exact copy of an original diskette. *source* specifies the drive containing the original diskette. *destination* specifies the drive containing the diskette to receive the copy. For single-drive systems, BASIC will prompt you to switch diskettes.

```
BACKUP 0 TO 1      BACKUP 1 TO 0
BACKUP 0
```

CIRCLE (*x,y*),*r,c,hw,start,end* Draws a circle with center at point (*x,y*) radius *r*, specified color *c*, height/width ratio (*hw*) of 0-4. Circle can start and end at specified point (0-1).

```
CIRCLE(128,96),50,4,1,.5,.75
```

CLEAR *n,h* Reserves *n* bytes of string storage space. Erases variables. *h* specifies highest BASIC address.

```
CLEAR
CLEAR 500
CLEAR 100,14000
```

CLOAD Loads specified program file from cassette. If *filename* is not specified, first file encountered is loaded. *Filename* must be eight character/spaces or fewer.

```
CLOAD
CLOAD "PROGRAM"
```

CLOADM Loads machine-language program cassette. An offset address to add to the loading address may be specified.

```
CLOADM "PROG"
CLOADM
CLOADM "PROG",1000
```

CLOSE#*buffer* Closes access to specified file. If *buffer* is not specified, all Open files will be closed.

```
CLOSE #1      CLOSE #1,#2      CLOSE
```

CLS *c* Clears display to specified color *c*. If color is not specified, green is used.

```
0-Black
1-Green      5-Blue
2-Yellow     6-Cyan
3-Blue       7-Magenta
4-Red        8-Orange
CLS
CLS 3
```


COLOR (foreground,background) Sets foreground and background color.
COLOR 1,3

CONT Continues program execution after pressing **BREAK** or using STOP statement.
CONT

COPY source-file:d TO destination-file:d Copies a file. Each filename must include an extension. *source-file:d* is the file and drive specification for the file to be copied. *destination-file:d* is a file and drive specification of the duplicate file.
COPY "DOCTOR/EXP:0" TO "DOCTOR/EXP:1"

CSAVE Saves program on cassette (program name must be eight character/spaces or less). If **A** is specified program is saved in ASCII format.
CSAVE "PROGRAM"
CSAVE "PROGRAM",A

CSAVEM name, start, end, transfer Saves a machine-language file on cassette.
CSAVEM "X" &H4E,&H6F,&H5F

DATA Stores data in your program. Use READ to assign data to variables.
DATA 5,3,PEARS
DATA PAPER,PEN

DEF FN Defines numeric function.
DEF FN(X)=X * 3

DEFUSR n Defines entry point for USR function *n*. *n*=0-9.
DEFUSR5=45643

DEL Deletes program lines.
DEL -
DEL 25
DEL 25-
DEL -25
DEL 10-25

DIM Dimensions one or more arrays.
DIM R(65),W(40)
DIM AR\$(8,25)

DIR d Displays specified diskette's directory. *d* is the drive specification. This is optional; if omitted, Drive 0 (or whatever drive was specified as the Master Drive using DRIVE) is used. A typical directory will have five columns: the first column contains the filename; the second is its extension; the third is the file type (0 = BASIC program, 1 = BASIC data file, 2 = machine-language file, 3 = Editor source file). The fourth column is the storage format (A = ASCII, B = Binary) and the fifth column is the file length (in granules).
DIR DIR 1

DLOAD Loads machine-language program at specified baud.
0 = 300 baud 1 = 1200 baud
DLOAD "X", 1

DRAW Draws a line beginning at specified starting point of specified length of specified color. Will also draw to scale, draw blank lines, draw non-updated lines, and execute substrings. If starting point is not specified, last DRAW position or (128,96) is used.
DRAW "BM100, 100;S10;U25;BR25;ND25;XA\$;"

DRIVE d Sets the specified drive (*d*) as the Master read/write drive. The Computer will then automatically use the specified drive when *d* is not specified. (On power-up, the Computer uses Drive 0 as the Master Drive.)
DRIVE 0 DRIVE 1

DSKI\$ d,track,sector,string variable 1, string variable 2 Reads specified sector into specified string variables. The first 128 bytes of the sector are read into *string variable 1*. The remaining 128 bytes of the sector are read into *string variable 2*.
DSKI\$0,3,17,A\$,B\$

DSKINI d,skip Initializes or formats (prepares) a diskette in specified drive (*d*) to accept information. *skip* specifies the number of sectors to be skipped between each logical record in the disk. *skip* must be between zero and 16; if omitted, four (4) is used.
DSKINI 0 DSKINI 1 DSKINI 0,6

DSKO\$ d,track,sector,string expression 1, string expression 2 Writes the data specified by the two string expressions into the specified sector on the diskette. The data from *string expression 1* is written into the first 128 bytes of the sector. The data from *string expression 2* is written into the remaining 128 bytes of the sector.
DSKO\$1,34,18,A\$,B\$

EDIT Allows editing of program line.
*n*C Changes *n* number of characters.
*n*D Deletes *n* number of characters.
I Allows insertion of new characters.
H Deletes rest of line and allows insert.
L Lists current line and continues edit.
*n*Sc Searches for *n*th occurrence of character *c*.
X Extends line.
(SHIFT)↑ Escape from subcommand.
n (SPACEBAR) Moves cursor *n* spaces to right.
n ← Moves cursor *n* spaces to left.
EDIT 25 (ENTER)

END Ends program.
END

EXEC (address) Transfers control to machine-language programs at specified address. If *address* is omitted, control is transferred to address set in last CLOADM.
EXEC
EXEC 32453

FIELD # buffer, field size AS field name Organizes the space within a direct access buffer into fields, assigns field size, and names the field. (See OPEN.)

```
FIELD #1,10 AS A$, 12 AS B$, 5 AS C$
```

FILES buffer#, buffer size Tells BASIC how many buffers to reserve in memory (*buffer #*) and the total number of bytes to reserve for these buffers (*buffer size*). If this command is not used, BASIC will reserve enough memory space for buffers 1 and 2 and reserve a total of 256 bytes for those buffers.

```
FILES 5,500 FILES 3,100
```

FOR..TO Creates a loop in program which the Computer **STEP/** must repeat from the first number to the last number **NEXT** you specify. Use **STEP** to specify how much to increment the number each time through the loop. If you omit **STEP**, one is used.

```
FOR X=2 TO 5 :NEXT X
FOR A=1 TO 10 STEP 5:NEXT A
FOR M=30 TO 10 STEP -5:NEXT M
```

GET (start)-(end), destination, G Reads the graphic contents of a rectangle into an array for future use by **PUT**.

```
GET (5,20)-(3,8),V,G
```

GET #buffer,record# Gets the specified record number and puts it in the buffer. *record #* is optional; if omitted, the next record is used. (See OPEN.)

```
GET #1,5 GET#2,3 GET #1
```

GOSUB Calls a subroutine beginning at specified line number.

```
GOSUB 500
```

GOTO Jumps to specified line number.

```
GOTO 300
```

IF test THEN. . . action 1 ELSE, action 2 Performs a test. If it is true, the Computer executes *action 1*. If false, *action 2* is executed.

```
IF A=5 THEN 30
```

INPUT Causes the Computer to stop and await input from the keyboard.

```
INPUT X$
INPUT "NAME" ;N$
```

INPUT #buffer, variable name, . . . Inputs data from specified buffer and assigns each data segment in the buffer to the specified variable name. (See OPEN.)

```
INPUT #1, A$,B$
```

INPUT #-1 Inputs data from cassette.

```
INPUT #-1,A
```

INSTR (position, search, target) Searches for the first occurrence of *target* string in *search* string beginning at *position*. Returns the position at which the match is found.

```
? INSTR (5,X$,Y$)
```

KILL filename:d Deletes or erases specified file from diskette. The extension must be included on *filename*. If *d* is omitted, Drive 0 is used.

```
KILL "DOCTOR/EXP"
KILL "DINNER/REC:1"
```

LET Assigns value to variable (optional).

```
LET A$ = "JOB A"
```

LIST Lists specified line(s) or entire program on screen.

```
LIST
LIST 50-85
LIST 30
LIST -30
LIST 30-
```

LLIST List specified program line(s) or entire program to printer.

```
LLIST
LLIST 50-85
LLIST 30
LLIST -30
LLIST 30-
```

LINE (x1,y1)-(x2,y2), PSET or PRESET, BF Draw a line from (*x1,y1*) to (*x2,y2*). If (*x1,y1*) is omitted, the last end point or (128,96) is used. **PSET** selects foreground color and **PRESET** selects background color. **B** draws a box with (*x1,y1*) and (*x2,y2*) as the opposing corners. **BF** will fill in the box with foreground color.

```
LINE (5,3)-(6,6),PSET
```

LINE INPUT Input line from keyboard.

```
LINE INPUT "ANSWER" ;X$
```

LOAD filename:d,R Load specified BASIC program file from diskette into memory. *d* is optional; if omitted, Drive 0 (or whatever drive was selected for Master Drive specified by **DRIVE**) is used. **R** is optional; if used, BASIC will run the program immediately after it is loaded. If *filename* does not have an extension, BASIC uses **/BAS**.

```
LOAD "PROGRAM",R LOAD "ACCTS/BAS:1"
```

LOADM filename, offset address Loads a machine-language program file from disk. If *filename* does not have an extension, BASIC uses **/BIN**. *offset address* is optional; if omitted, program loads where specified inside the program.

```
LOADM "PROG/COR,3522"
LOADM "TEST1"
```

LSET field name = data Left-justifies data within specified field name.

```
LSET A$="APPLES"
```

MERGE filename,R Loads a program disk (which was saved in ASCII — the **A** option) and merges it with program currently in memory. If programs have corresponding line numbers, the program on disk will erase the program in memory. **R** is optional; if used, BASIC will immediately run the merged program after it is loaded.

```
MERGE "SUB/BAS" MERGE "NEW/BAS",R
```


MID\$ (oldstr, position, length) Replaces a portion of oldstr with another string.
MID\$ (A\$,14,2) = "KS"

MKN\$ (number) Converts *number* to a five-character string it can be stored in a formatted disk file.
LSET B\$ = MKN\$(10)

MOTOR Turns cassette ON or OFF.
MOTOR ON
MOTOR OFF

NEW Erases everything in memory.
NEW

ON. . .GOSUB Multi-way branch to call specified subroutines.
ON Y GOSUB 50,100

ON. . .GOTO Multi-way branch to specified lines.
ON X GOTO 190,200

OPEN mode, buffer, filename, record-length Opens a file. *mode* can be :I = Inputs data from a sequential access file; O = Outputs data to a sequential access file; D = Inputs or outputs data to a direct access file. *buffer* and the devices they communicate with are: 0 = Display or Printer; -1 = Tape Recorder; -2 = Printer; 1-15 = Disk Drive. If *filename* does not have an extension, BASIC will assign the extension /DAT. *record-length* must be included for use with direct access files.
OPEN "D",#1,"FILE",15
OPEN "I",#2,"CNGE/DAT"

OPEN m,#d,f Opens specified file (*f*) for data transmission (*m*) to specified device (*d*). *m* may be I (Input) or O (Output). *d* may be #0 (Screen or Keyboard), #-1 (Cassette), or #-2 (Printer).
OPEN "O",-1,"DATA"

PAINT (x,y),c,b Paints graphic screen starting at point (x,y) with specified color *c* and stopping at border (*b*) of specified color.
PAINT (10,10),2,4

PCLEAR n Reserves *n* number of 1.5 K graphics memory pages.
PCLEAR 8

PCLS c Clears screen with specified color *c*. If color code is omitted, current background color is used. (See CLS for color codes.)
PCLS 3

PCOPY Copy graphics from source page to destination page.
PCOPY 5 TO 6

PLAY Plays music of specified note (A-G or 1-12), octave (O), volume (V), note-length (L), tempo (T), pause (P), and allows execution of substrings. Also sharps (# or +) and flats (-).
PLAY "L1;A#;PB;V10;T3;L2;B-;9;XA\$;"

PMODE mode, start-page Selects resolution and first memory page.
PMODE 4,1

POKE (location, value) Puts value (0-255) into specified memory location.
POKE 15872,255

PRESET Reset a point to background color.
PRESET (5,6)

PRINT Prints specified message or number on TV screen.
PRINT "HI"

PRINT # buffer, data list Prints *data list* to specified *buffer*. (See OPEN.) To separate items within data list, either commas or semi-colons may be used.
PRINT #1,"DATA" PRINT #-2,"EXP/DAT"

PRINT #-1 Writes data to cassette.
PRINT A\$

PRINT#-2 Prints an item or list of items on the printer.
PRINT#-2,CAP\$

PRINT TAB Moves the cursor to specified column position.
PRINT TAB(5)"NAME"

PRINT USING Prints numbers in specified format.

Formats numbers.
PRINT USING "####";62,2
. Decimal point.
PRINT USING "##.#";58.6
, Displays comma to left of every third character.
PRINT USING "####,";44.0
.. Fills leading spaces with asterisks.
PRINT USING "####,##";33.3
\$ Places \$ ahead of number.
PRINT USING "\$##,##";33.3
\$\$ Floating dollar sign.
PRINT USING "\$\$##,##";11.544
**\$ Floating dollar sign.
PRINT USING "**\$##,##";8,333
+ In first position, causes sign to be printed. In last position, causes sign to be printed after the number.
PRINT USING "+##,##";-216
Exponential format.
↑↑↑↑ PRINT USING "##.#↑↑↑↑";546
- Minus sign after negative numbers.
PRINT USING "##.#-";-534.7
/ Returns first string character.
PRINT USING "!";"YELLOW"
%spaces% String field; length of field is number of spaces plus 2.
PRINT USING "% %";"BLUE"

PRINT @ location Prints specified message at specified text screen location.

```
PRINT @256,"HI"  
PRINT @256,A$
```

PSET (x,y,c) Sets a specified point (x,y) to specified color c. If c is omitted, foreground is used.

```
PSET (5,6,3)
```

PUT (start)-(end), source, action Stores graphics from source onto start/end rectangle on the screen. (Array rectangle size must match GET rectangle size.)

```
PUT (3,2)-(5,6),V,PSET
```

PUT# buffer, record number Assigns a record number to the data in buffer. record number is optional; if omitted, BASIC will use the current record number. (See OPEN.)

```
PUT #2,3      PUT #1,4
```

READ Reads the next item in DATA line and assigns it to specified variable.

```
READ A$  
READ C,B
```

REM Allows insertion of comment in program line. Everything after REM is ignored by Computer.

```
REM THIS IS IGNORED  
10?X$ :REM IGNORE
```

RENAME old filename:d TO new filename:d Renames a file. filename must include an extension. If d is not specified, BASIC uses Drive 0 and will not search other drives.

```
RENAME "MFILE/DAT:1" TO "BFILE/DAT:1"
```

RENUM newline, startline, increment Allows program line renumbering.

```
RENUM 1000,5,100
```

RESET (x,y) Resets a point.

```
RESET (14,15)
```

RSET fieldname = data Right justifies the data within specified fieldname. If data is larger than the field, the right-hand characters will be truncated.

```
RSET M$="SOAP"
```

RESTORE Sets the Computer's pointer back to first item on the first DATA line.

```
RESTORE
```

RETURN Returns the Computer from subroutine to the BASIC word following GOSUB.

```
RETURN
```

RUN Executes a program.

```
RUN
```

RUN filename,R Loads specified disk file (filename) and runs it. ,R optional; if used, all Open files will remain open.

```
RUN "FILE"      RUN "PRDG/BAS"R
```

SAVE filename,A Saves specified file (filename) on diskette. If an extension is not assigned to filename, BASIC will assign the extension/BAS. ,A is optional; if used, program is saved in ASCII format.

```
SAVE "PRDG/BAS"      SAVE "TEST:1",A
```

SAVEM filename, start address, end address, execution address Saves specified machine-language file (filename) on diskette which begins at start address and ends at end address. execution address specifies the address at which program will begin execution. If filename is not assigned an extension, BASIC will assign the extension/BIN.

```
SAVEM "FILE/BIN:1" ,5200,5800,5300
```

SCREEN screen-type, color-set Selects either graphics (1) or text (0) screen and color-set (0 or 1).

```
SCREEN 1,1
```

SET (x,y,c) Sets a dot at specified text screen location to specified color.

```
SET (14,13,3)
```

SKIPF Skips to next program on cassette tape, or to end of specified program.

```
SKIPF "PROGRAM"
```

SOUND tone, duration Sounds specified tone for specified duration.

```
SOUND 128,3
```

STOP Stops execution of a program.

```
STOP
```

TROFF Turns program tracer OFF.

```
TROFF
```

TRON Turns program tracer ON.

```
TRON
```

WRITE# buffer, data list Writes data to specified buffer. (See OPEN.) A comma is used to separate each item in the data list.

```
WRITE #1, A$, B$, C
```

UNLOAD d Closes all Open files on diskette in specified drive d.

```
UNLOAD 1      UNLOAD 2
```

VERIFY switch Verifies whatever you are currently writing on a diskette is not lost or altered. switch is ON or OFF.

```
VERIFY ON      VERIFY OFF
```


Notes:

Functions

Argument ranges are indicated by special symbols:

numeric: $(-10^{38}, +10^{38})$

x: (0-255)

y: (0-191)

location: (0-65535)

code: (0-255)

str: string argument

var: variable name

ABS (*numeric*) Computes absolute value.

Y=ABS(5)

ASC (*str*) Returns ASCII code of first character of specified string.

A=ASC(T\$)

ATN (*numeric*) Returns arctangent in radians.

Y=ATN(X/3)

CHR\$ (*code*) Returns character for ASCII, control, or graphics code.

?CHR\$(191)

P\$=CHR\$(T)

COS (*numeric*) Returns cosine of an angle given in radians.

Y = COS(7)

CVN (*string variable*) Converts data back to its original number after the data had been converted to a string using MKNS.

X=CVN(A\$)

EOF (*buffer*) Checks to see if end-of-file is encountered during a READ operation. Returns a 0 if there is more data to be read, a -1 if there is no more data in the file.

IF EOF(1) = -1 THEN CLOSE #1

EOF (*f*) Returns FALSE (0) if there is more data; TRUE(-1) if end of file has been read. For cassette, $f = -1$ for keyboard, $f = 0$.

EOF(-1)

EOF(0)

EXP (*numeric*) Returns natural exponential of number (e^{number}).

Y = EXP(7)

FIX (*numeric*) Returns truncated (whole number) value.

Y=FIX(7.6)

FREE *d* Displays the number of free granules on specified Drive *d*. *d* is optional; if not omitted, Drive 0 (or Master Drive specified by DRIVE) is used.

PRINT FREE 1 ? FREE

HEX\$ (numeric) Computes hexadecimal value.

```
PRINT HEX$(30)
Y = HEX$(X/16)
```

INKEY\$ Checks the keyboard and returns the key being pressed (if any).

```
A$ = INKEY$
```

INT (numeric) Converts a number to an integer.

```
X = INT(5.2)
```

JOYSTK (j) Returns the horizontal or vertical coordinate (j) of the left or right joystick:

```
0 = horizontal, left joystick
1 = vertical, left joystick
2 = horizontal, right joystick
3 = vertical, right joystick
M = JOYSTK(0)
H = JOYSTK(1)
```

LEN (str) Returns the length of a string.

```
X = LEN(SEN$)
```

LOC (buffer) Returns the current record number of specified buffer.

```
PRINT LOC(#1)
```

LOF (buffer) Returns the number of the last (highest-numbered) record in specified buffer.

```
FOR X = 1 TO LOF(#1) Y = LOF(#5)
```

LOG (numeric) Returns natural logarithm.

```
Y = LOG(353)
```

MEM Finds the amount of free memory.

```
PRINT MEM
```

MID\$ (str,pos,length) Returns a substring of another string starting at pos. If length is omitted, the entire string right of position is returned.

```
F$ = MID$(A$,3)
?MID$(A$,3,2)
```

PEEK (location) Returns the contents of specified memory location.

```
A = PEEK(32076)
```

POINT (x,y) Tests whether specified graphics cell is on or off, x (horizontal) = 0-63; y (vertical) = 0-31. The value returned is -1 if the cell is in a text character; mode; 0 if it is off, or the color code if it is on. See CLS for color codes.

```
IF POINT(10,10) THEN PRINT "ON" ELSE
PRINT "OFF"
```

POS (device) Returns current print position. Device-2 = printer, 0 = display.

```
PRINT TAB(8) POS(0)
```

PPOINT (x,y) Tests whether specified graphics cell is on or off and returns color code of specified cell.

```
PPOINT(13,35)
```

RIGHT\$ (str,length) Returns right portion of string.

```
ZIP$ = RIGHT$(AD$,5)
```

SGN (numeric) Returns sign of specified numeric expression:

```
-1 if argument is negative
0 if argument is 0
+1 if argument is positive
X = SGN(A*B)
```

SIN (numeric) Returns sine of angle given in radians.

```
Y = SIN(5)
```

STRING\$ (length, code or string) Returns a string of characters (of specified length) specified by ASCII code or by the first character of the string.

```
?STRING$(5,"Z")
?STRING$(5,91)
```

STR\$ (numeric) Converts a numeric expression to a string.

```
S$ = STR$(X)
```

SQR (numeric) Returns the square root of a number.

```
Y = SQR(5+3)
```

TAN (numeric) Returns tangent of angle given in radians.

```
Y = TAN(45.7)
```

TIMER Returns contents or allows setting of timer (0-65535).

```
?TIMER
TIMER=0
```

USRn (numeric) Calls user's machine-language subroutine.

```
X = USR(Y)
```

VAL (str) Converts a string to a number.

```
A = VAL(B$)
```

VARPTR (var) Returns addresses of pointer to the specified variable.

```
Y = USR(VARPTR(X))
```

Notes:

ROM Subroutines

The Extended Color BASIC ROM contains many subroutines that can be called by a machine-language program; many of these can also be called by a Color BASIC program via the USR function. Each subroutine will be described in the following format:

NAME — *Entry address*
Operation Performed

The subroutine **NAME** is for reference only. It is not recognized by the Computer. *Entry address* is given in hexadecimal form; you must use an indirect jump to this address. For specific information on *Entry* and *Exit Conditions*, see the **Technical Information** section of your Extended Color BASIC manual or **Section IV** of the Color BASIC manual.

BLKIN = (A006)

Reads a block from Cassette

BLKOUT = (A009)

Writes a block to Cassette

CHROUT = (A002)

Outputs a character to device specified by the contents of 6F.

CSRDON = (A004)

Starts Cassette

DSKCON = (C004)

Read or write a disk sector.

JOYIN = (A00A)

Sample Joystick pots and stores values.

POLCAT = (A001)





Polls the keyboard for a character.

WRTLDR = (A7D8)

Turns the Cassette on and writes a leader.

Notes:



Control Keys

	Cancels last character typed; moves cursor back one space.
SHIFT 	Erases current line.
BREAK	Interrupts anything in progress and returns to command level.
CLEAR	Clears the screen.
ENTER	Signifies end of current line.
SPACEBAR	Enters a space (blank) character and moves cursor one space forward.
SHIFT 	Causes currently executing program to pause (press any key to continue).
SHIFT 	All-caps/upper-lowercase keyboard switch. (Lowercase displayed as reversed capitals.)

Special Characters

*	Abbreviation for REM.
\$	Makes variable string type.
:	Separates statements on the same line.
?	Same as PRINT.
,	PRINT punctuation; spaces over to the next 16-column PRINT zone.
;	PRINT punctuation; separates items in a PRINT list but does not add spaces when they are output.

Operators

Each operator or group of operators is precedent over the group below it.	
 -, +	Exponentiation Unary negative, positive
 +, -	Multiplication, division Addition and concatenation, subtraction
<, >, =, <=, >=, <>	Relational tests
NOT AND OR	

Notes:

Video Control Codes

Dec	Hex	PRINT CHR\$ (code)
8	08	Backspaces and erases current character.
13	0D	Line feed with carriage return.
32	20	Space

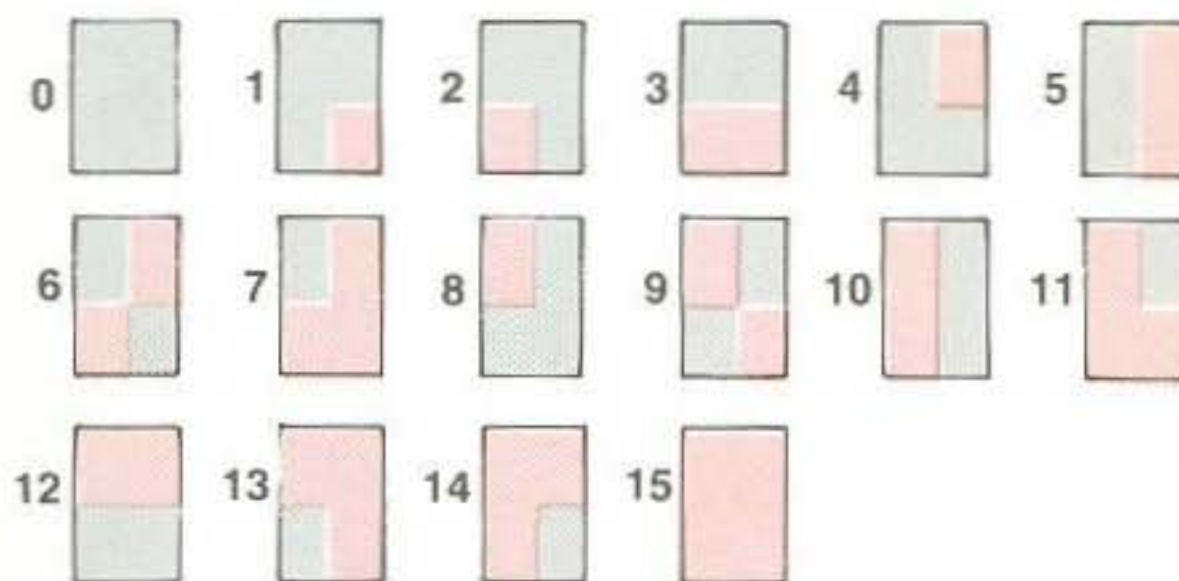
Color Codes

CODE	COLOR
0	Black
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

Graphic Character Codes

Given the *color* (1-8) and the *pattern* (0-15), this formula will generate the correct code:

$$\text{code} = 128 + 16 * (\text{color} - 1) + \text{pattern}$$



For example, to print *pattern* 9 in blue (*code* 3), type:

```
C = 128 + 16 * (3-1) + 9
? CHR$(C)
```


Notes:

ASCII Character Codes

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACEBAR	32	20
!	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
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F

P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
↑*	94	5E
↓*	10	0A
←*	8	08
→*	9	09
BREAK	03	03
CLEAR	12	0C
ENTER	13	0D

*If shifted, the code for these characters are as follows: **CLEAR** is 92 (hex 5C); ↑ is 95 (hex 5F); ↓ is 91 (hex 5B); ← is 21 (hex 15); and → is 93 (hex 5D).

These are the ASCII codes for lowercase letters. You can produce these characters by pressing **SHIFT** **Q** simultaneously to get into an upper-lowercase mode. The lowercase letters will appear on your screen in reversed colors (green with a black background).

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A

Extended Color BASIC Color-Set Table

PMODE #	Color Set	Two Color Combination	Four Color Combination
4	0	Black/Green	—
	1	Black/Buf	—
3	0	—	Green/Yellow/Blue/Red
	1	—	Buf/Cyan/Magen;a/Orange
2	0	Black/Green	—
	1	Black/Buf	—
1	0	—	Green/Yellow/Blue/Red
	1	—	Buf/Cyan/Magenta/Orange
0	0	Black/Green	—
	1	Black/Buf	—

Extended Color BASIC Graphics Screen Resolution Table

PMODE #	Grid Size	Color-Mode	Pages Used	Point Size
4	256x192	two-color	4	□
3	128x192	four-color	4	□
2	128x192	two-color	2	□
1	128x96	four-color	2	□
0	128x96	two-color	1	□

Musical Note/Number Codes

Number	Note
1	C
2	C#/D-
3	D/E-
4	E-/D#
5	E/F-
6	F/E#
7	F#/G-
8	G
9	G#/A-
10	A
11	A#/B-
12	B

Note: PLAY does not recognize the notation "B#" or "C-". Use the numbers 1 and 12 respectively or substitute C for B# and B for C-. An ?FC ERROR will occur if you try to use either of these notations.

Notes:

Editor Commands

Terms You'll Need To Know . . .

line Specifies a program line and may be any number between 0-63999. The following symbols may also be used: # (first line in the program), * (last line in the program), . (current line).

current line Specifies the last line inserted, edited, or printed.

startline Specifies the line where an operation begins. In most commands *startline* is optional; if omitted, the current line is used.

endline Specifies the line where an operation ends. In most commands, *endline* is optional; if omitted, the last line in the program is used.

range Specifies the line(s) in an operation. If more than one line is in the range, those lines must be specified by one of the following: : (separates *startline* from *endline*), ! (separates *startline* from the number of lines that follow *startline*).

increment Specifies the number of steps (line numbers) between successive program lines. On start-up, an increment of 10 is used. In most commands, *increment* is optional; if omitted, the last specified increment is used.

A filename/switch . . .

Assembles text program (specified by *filename*) into machine code. *filename* is a standard Color BASIC file specification and is optional; if omitted, NONAME is used. *switch* may be any of the following: /AO (Absolute Origin—applies only if /IM is set.); /IM (In Memory Assembly) /LP (Assembly listing on printer); /MO (Manual Origin—Applies only if /IM is set.) /NL (No listing printed); /NO (No object code generated.) /NS (No symbol table generated.) /SS (Short screen.) /WE (Wait on errors.) Unless the /IM switch is used, the program will be assembled on tape using the specified *filename*.

ASAMPLE / IM A

Cstartline, range, increment

Copies *range* to a new location that begins with *startline* using specified *increment*. *startline*, *range*, and *increment* must all be included.

C500,100:150,10

Drange

Deletes *range*. If *range* is omitted, current line is deleted.

D100 D100:150 D

Eline

Enters a line for editing. *line* is optional; if omitted, current line is used.

E100 E

Fstring

Finds specified string of characters. Search begins with the current line and ends each time the string is found. *string* is optional; if omitted, the last string defined is used.

FABC F

Hrange

Prints contents of *range* on the Printer. *range* is optional; if omitted, current line is used.

H100 H100:200 H

Istartline, increment

Inserts lines beginning at *startline* using the specified increment. *startline* and *increment* are optional; if omitted, current line and 10 (or previous increment) are used.

I150,5 I20 I,10

Lfilename

Loads specified file (*filename*) from cassette tape. *filename* is optional; if omitted, the next file is loaded.

L SAMPLE L

Nstartline, increment

Renums program beginning at *startline* and using the specified increment. *startline* and *increment* are optional; if omitted, current line and 10 (or previous increment) are used.

N100,50 N100 N

Prange

Displays contents of *range*.

P100:200 P100!5 P# P* P

Q

Return to BASIC.

Q

Rstartline, increment

Replaces startline and inserts lines using *increment*. *startline* and *increment* are optional; if omitted, current line and 10 (or previous increment) are used.

R100,10 R100 R

Trange

Prints contents of *range* on the Printer. (Line numbers are not printed.)

T100 T100:500

Vfilename

VTEST — Verify contents of *filename* on tape.

Assembler Switches

- /AO Absolute Origin.** The program is located as indicated in any ORG statements in the assembly-language listing. (Applies only if /IM is set.)
- /IM In Memory Assembly.** The program is located in memory, not on tape.
- /LP Line Printer.** Assembly is listed on the Line Printer.
- /MO Manual Origin.** The program is located beginning at USRORG, plus the values of any ORG statements. (Applies only if /IM is set.)
- /NL No listing printed.**
- /NO No object code generated.**
- /NS No symbol table generated.**
- /SS Short screen.**
- /WE Wait after each error** during assembly. Any keypress continues assembly. The errors will be listed at the end of the assembly.

Notes:

ZBUG Commands

Terms You'll Need To Know . . .

expression One or more numbers, symbols, or ASCII characters. If more than one are used, you may separate them with these operators: * (Multiplication) + (Addition) .DIV. (Division), - (Subtraction), .MOD. (Modulo), .EQU. (Equals), < (Shift), .NEQ. (Not Equal), .AND. (Local And), + (Positive), .XOR. (Exclusive Or), - (Negative), .OR. (Logical Or), .NOT. (Complement).

address A location in memory. This is a numeric expression between 0-16383 decimal for 16K Color Computers and between 0-32767 decimal for 32K Color Computers. *address* may be an expression using numbers and labels and symbols from the symbol table.

C Continues execution of the program after interruption at a breakpoint.

C

D Displays all breakpoints that have been set.

D

E Exits ZBUG and enters the Editor.

E

Gaddress

Executes the program which begins at *address*. *address* must be specified.

G15343

Lfilename

Loads machine-code file (specified by *filename*) from cassette tape. *filename* is optional; if omitted, the next file is loaded.

LSAMPLE

Pfilename start-address end-address

execution-address

Saves on tape the contents of memory from *start-address* to *end-address*. *execution-address* specifies the address where the program being saved begins execution.

PSAMPLE 14889 15338 15110

R

Displays the contents of all registers.

R

Tstart-address end-address

Displays memory locations from *start-address* *end-address*, inclusive.

T8001 9871

TH*start-address end-address*

Prints memory locations from *start-address* to *end-address*, inclusive.

TH6556 7734

U*source-address destination-address count*

Transfers the contents of *source-address* to *destination-address*. *count* specifies the number of program lines following *source-address* to transfer.

U1334 5559 10

V*filename*

Verifies data on the specified file (*filename*). *filename* is optional; if omitted, the next file on the tape is used.

VSAMPLE V

X*address*

Sets a breakpoint at specified *address*. *address* is optional; if omitted, the current location will be used.




X12982 X

Y*address*

Deletes breakpoint at the specified *address*. *address* is optional; if omitted, all breakpoints are deleted.

YB734 Y

Special Symbols

address/ register/	Opens address or register and displays its contents. If address or register is omitted, the last address opened will be re-opened. After the contents have been displayed, you may type new contents to change the contents
(ENTER)	to close and enter any change
(BREAK)	to close and not enter any change
	to open next address and enter any change
	to open preceding address
	to branch with instruction
:	to force numeric display mode
=	to force numeric and byte modes
:	to force flags
address,	Executes <i>address</i> . If <i>address</i> is omitted, the next instruction is executed. Single step through program.
expression =	Calculates expression and displays the results available in all 3 number systems or combinations of all three

Notes:

Examination Mode Commands

A	ASCII Mode	1 Byte
B	Byte Mode	1 Byte
M	Mnemonic Mode	1 or More Bytes
W	Word Mode	2 Bytes

Note: If Examination Mode is not specified, M (Mnemonic) Mode is used.

Display Mode Commands

H	Half Symbolic	Addresses displayed numerically— operands displayed symbolically
N	Numeric	Addresses and values displayed as numbers
S	Symbolic	Addresses and values displayed as labels and symbols.

Note: If Display Mode is not specified, S (Symbolic) is used.

Number Base Mode Commands

<i>Obase</i>	Output
<i>ibase</i>	Input

Note: *base* can be 8, 10, or 16 and specifies the numbering system needed. *base* is optional; if omitted 16 is used.

Notes:

6809 Instructions (Including Pseudo-Operation Instructions)

ABX

Add accumulator B into index register X.
Adds the 8-bit unsigned value in accumulator B into index register X.

ADC

Add with Carry into Register.
Adds the contents of the C (carry) bit and the memory byte into an 8-bit accumulator.

ADD

(8-bit) Add Memory into Register.
Adds the memory byte into an 8-bit accumulator.

ADD

(16-bit) Add Memory into Register.
Adds the 16-bit memory value into the 16-bit accumulator.

AND

Logical AND Memory into Register.
Performs the logical AND operation between the contents of an accumulator and the contents of a memory location. The result is stored in the accumulator.

AND CC

Logical AND Immediate Memory into Condition Code Register.
Performs a logical AND between the condition code register and the immediate byte specified in the instruction and places the result in the condition code register.

ASL

Arithmetic Shift Left.
Shifts all bits of the operand one place to the left. Bit zero is loaded with a zero. Bit seven is shifted into the C (carry) bit.

ASR

Arithmetic Shift Right.
Shifts all bits of the operand one place to the right. Bit seven is held constant. Bit zero is shifted into the C (carry) bit.

BCC

Branch on Carry Clear.
Tests the state of the C (carry) bit and causes a branch if it is clear.

BCS

Branch on Carry Set.
Tests the state of the C (carry) bit and causes a branch if it is set.

BEQ

Branch if Equal.

Tests the state of the Z (zero) bit and causes a branch if it is set. When used after a subtract or compare operation, this instruction will branch if the compared values, signed or unsigned, were exactly the same.

BGE

Branch on Greater Than or Equal to Zero.

Causes a branch if the N (negative) bit and the V (overflow) bit are either both set or both clear. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than or equal to the memory operand.

BGT

Branch on Greater.

Causes a branch if the N (negative) bit and V (overflow) bit are either both set or both clear and the Z (zero) bit is clear. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was greater than the memory operand.

BHI

Branch if Higher.

Causes a branch if the previous operation caused neither a carry nor a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was higher than the memory operand.

BHS

Branch if Higher or Same.

Tests the state of the C (carry) bit and causes a branch if it is clear. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was higher than or the same as the memory operand.

BIT

Bit Test.

Performs a logical AND of the contents of accumulator A or B and the contents of a memory location and modifies the condition codes accordingly. The contents of accumulator A or B and the memory location are not affected.

BLE

Branch on Less Than or Equal to Zero.

Causes a branch if the exclusive OR of the N (negative) and V (overflow) bits is 1 or if the Z (zero) bit is set. When used after a subtract or compare operation on twos complement values, this instruction will branch if the register was less than or equal to the memory operand.

BLO

Branch on Lower.

Tests the state of the C (carry) bit and causes a branch if it is set. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was lower than the memory operand.

BLS

Branch on Lower or Same.

Causes a branch if the previous operation caused either a carry or a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was lower than or the same as the memory operand.

BLT

Branch on Less Than Zero.

Causes a branch if either, but not both, of the N (negative) or V (overflow) bits are set. When used after a subtract or compare operation on twos complement binary values, this instruction will branch if the register was less than the memory operand.

BMI

Branch on Minus.

Tests the state of the N (negative) bit and causes a branch if set.

BNE

Branch not Equal.

Tests the state of the Z (zero) bit and causes a branch if it is clear. When used after a subtract or compare operation on any binary values, this instruction will branch if the register is, or would be, not equal to the memory operand.

BPL

Branch on Plus.

Tests the state of the N (negative) bit and causes a branch if it is clear.

BRA

Branch Always.

Causes an unconditional branch.

BRN

Branch Never.

Does not cause a branch. This instruction is essentially a no operation, but has a bit pattern logically related to Branch Always.

BSR

Branch to Subroutine.

The program counter is pushed onto the stack. The program counter is then loaded with the sum of the program counter and the offset.

BVC

Branch on Overflow Clear.

Tests the state of the V (overflow) bit and causes a branch if it is clear. When used after an operation on twos complement binary values, this instruction will branch if there was no overflow.

BVS

Branch on Overflow Set.

Tests the state of the V (overflow) bit and causes a branch if it is set. When used after an operation on twos complement binary values, this instruction will branch if there was an overflow.

CLR

Clear.
Accumulator A or B or a memory location is loaded with 00000000.

CMP

(8-bit) Compare Memory from Register.
Compares the contents of memory location to the contents of the specified register and sets the appropriate condition codes. Neither the memory location nor the specified register is modified.

CMP

(16-bit) Compare Memory from Register.
Compares the 16-bit contents of the concatenated memory locations to the contents of the specified register and sets the appropriate condition codes. Neither the memory locations nor the specified register is modified unless autoincrement or autodecrement are used.

COM

Complement.
Replaces the contents of a memory location or accumulator A or B with its logical complement. When operating on twos complement values, all signed branches are available.

CWAI

Clear CC bits and Wait for Interrupt.
This instruction ANDs an immediate byte with the condition code register which may clear the interrupt mask bits I and F, stacks the entire machine state on the hardware stack and then looks for an interrupt.

DAA

Decimal Addition Adjust.
The sequence of a single-byte add instruction on accumulator A and following decimal addition adjust instruction results in a BCD addition with an appropriate carry bit.

DEC

Decrement.
Subtracts one from the operand. The carry bit is not affected., thus allowing this instruction to be used as a loop counter in multiple-precision computations.

EOR

Exclusive OR.
The contents of a memory location is exclusive ORed into an 8-bit register.

EQU

Initializes data or addresses and sets up a label. EQU commands may be located anywhere in the program.

EXG

Exchange Registers.
Exchanges data between two designated registers.

FCB

Form Constant Byte.
Inserts one byte of data at this point in the program.

FCC

Form Constant Character.
Writes an ASCII string into memory, using the syntax: *label FCC delimiter string delimiter.*

INC

Increment.
Adds to the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations.

JMP

Jump.
Program control is transferred to the effective address.

JSR

Jump to Subroutine.
Program control is transferred to the effective address after storing the return address on the stack.

LD

(8-bit) Load Register from Memory.
Loads the contents of a memory location into the designated register.

LD

(16-bit) Load Register from Memory.
Loads the contents of the memory locations into the designated 16-bit register.

LEA

Load Effective Address.
Calculates the effective address from the indexed addressing mode and places the address in an indexable register.

LSL

Logical Shift Left.
Shifts all bits of accumulator A or B or a memory location one place to the left. Bit zero is loaded with a zero. Bit seven of accumulator A or B or the memory location is shifted into the C (carry) bit.

LSR

Logical Shift Right.
Performs a logical shift right on the operand. Shifts a zero into bit seven and bit zero into the C (carry) bit.

MUL

Multiply.
Multiply the unsigned binary numbers in the accumulators and place the result in both accumulators. Unsigned multiply allows multiple-precision operations.

NEG

Negate.
Replaces the operand with its twos complement. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

NOP

No Operation.
This instruction causes only the program counter to be incremented. No other register or memory locations are affected.

OR

Inclusive OR Memory into Register.
Performs an inclusive OR operation between the contents of accumulator A or B and the contents of memory location M and the result is stored in accumulator A or B.

ORCC

Inclusive OR Memory Immediate into Condition Code Register.
Performs an inclusive OR operation between the contents of the condition code registers and the immediate value, and the result is placed in the condition code register. This instruction may be used to set interrupt masks (disable interrupts) or any other bit(s).

ORG

Tells the assembler where to begin locating the op code in memory.

Example: `ORG $3F00`
This tells the assembler to locate the program op code beginning at address \$3F00. When the assembler arrives at the new ORG command, it will begin locating program code at the new address. An ORG statement must not have a label.

PSHS

Push Registers on the Hardware Stack.
All, some, or none of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer).

PSHU

Push Registers on the User Stack.
All, some, or none of the processor registers are pushed onto the user stack (with the exception of the user stack pointer).

PULS

Pull Registers from the Hardware Stack.
All, some, or none of the processor registers are pulled from hardware stack (with the exception of the hardware stack pointer).

PULU

Pull Registers from the User Stack.
All, some, or none of the processor registers are pulled from the user stack (with the exception of the hardware stack pointer).

RESTART

Restart Hardware Interrupt.
The processor is initialized (required after power-on) to start program execution. The starting address is located from the restart vector.

ROL

Rotate Left.
Rotates all bits of the operand one place left through the C (carry) bit.

ROR

Rotate Right.
Rotates all bits of the operand one place right through the C (carry) bit.

RTI

Return from Interrupt.
The saved machine state is recovered from the hardware stack and control is returned to the interrupted program.

RTS

Return from Subroutine.
Program control is returned from the subroutine to the calling program. The return address is pulled from the stack.

SBC

Subtract with Borrow.
Subtracts the contents of a memory location and the borrow (in the C bit) from the contents of the designated 8-bit register, and places the result in that register.

SETDP

Tells the assembler to set the direct page.
Example: `SETDP $20`
Tells the assembler to set the direct page to \$20.

SEX

Sign Extended.
This instruction transforms a twos complement 8-bit value in accumulator B into a twos complement 16-bit value in the D accumulator.

ST

(8-bit) Store Register into Memory.
Writes the contents of an 8-bit register into a memory location.

ST

(16-bit) Store Register into Memory.
Writes the contents of a 16-bit register into two consecutive memory locations.

SUB

(8-bit) Subtract Memory from Register.
Subtracts the value in a memory location from the contents of a designated 8-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

SUB

(16-bit) Subtract Memory from Register.
Subtracts the value in memory locations from the contents of a designated 16-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

SWI

Software Interrupt.
All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer), and control is transferred through the software interrupt vector.

SWI2

Software Interrupt 2.
All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer), and control is transferred through the software interrupt 2 vector.

SWI3

Software Interrupt 3.
All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer), and control is transferred through the software interrupt 3 vector.

SYNC

Synchronize to External Event.
Provides software synchronization with a hardware process. When executed, the processor enters a synchronizing state, stops processing instructions, and waits for an interrupt. When an interrupt occurs, the synchronizing state is cleared and processing continues.

TFR

Transfer Register to Register.
Transfers data between two designated registers.

TST

Test.
Set the N (negative) and Z (zero) bits according to the contents of a memory location, and clear the V (overflow) bit.

FIRQ

Fast Interrupt Request, Hardware Interrupt.
A FIRQ with the F (fast interrupt request mask) bit clear causes this interrupt sequence to occur at the end of the current instruction.

IRQ

Interrupt Request, Hardware Interrupt.
If the I (interrupt request mask) bit is clear, a low level on the IRQ input causes this interrupt sequence to occur at the end of the current instruction.

NMI

Non-Maskable Interrupt, Hardware Interrupt.
A negative edge on the NMI input causes all of the processor's registers (except the hardware stack pointer) to be pushed onto the hardware stack, starting at the end of the current instruction.

Editor Assembler Error Messages

- BAD BREAKPOINT (ZBUG)** Attempt to set a breakpoint that is: greater than 7; in ROM; at a SWI command; at an address where one is already set.
- BAD COMMAND (Editor)** Illegal command letter.
- BAD COMMAND (ZBUG)** Command used is not a ZBUG command.
- BAD LABEL (Assembler)** Symbol used is: an illegal symbol; not terminated with a space, tab, or carriage return; used with ORG or END which do not allow labels.
- BAD LINE NUMBER (Editor)** The line number used is not in the range of 1-63999.
- BAD MEMORY (Assembler)** Attempting to do an in-memory assembly which would: overwrite system memory (an address lower than hexadecimal 0600); overwrite the edit buffer or symbol table; go into the protected area set by USRORG; go over the top of RAM.
- BAD MEMORY (ZBUG)** Data did not store correctly on a memory modification.
- BAD OPCODE (Assembler)** Op code is either invalid or is not terminated with a space, tab or carriage return.
- BAD OPERAND (Assembler)** Syntax error in operand field.
- BAD PARAMETERS (Editor)** Syntax error in command line.
- BAD PARAMETERS (ZBUG)** Specified filename is more than eight characters.
- BAD RADIX (ZBUG)** Numbering system is not 10, 8 or 16.
- BUFFER FULL (Editor)** Not enough room in the Edit Buffer.
- BUFFER EMPTY (Editor)** Command requires text in the Edit Buffer and there isn't any.
- BYTE OVERFLOW (Assembler)** Field overflow in an immediate operand, an offset, a short branch, or an FCB pseudo op.
- DP ERROR (Assembler)** Direct Page Error. The high order byte of an operand where direct addressing has been forced does not match the value set by the most recent SETDP pseudo op.
- FM ERROR (Editor and ZBUG)** File Mode Error. File being loaded is not a TEXT file (if in the Editor) or a CODE file (if in ZBUG).
- I/O ERROR (Editor and ZBUG)** Input/Output Error. A checksum error was encountered while loading a file from a cassette tape.
- MISSING END (Assembler)** Last command is not END.

MISSING INFORMATION (Assembler) There is not a delimiter in an FCC pseudo op; a label on a SET or EQU pseudo op.

MISSING OPERAND (Assembler) One or more operands are missing from a command.

MULTIPLY DEFINED SYMBOL (Assembler) A label has been defined more than once.

NO ROOM BETWEEN LINES (Editor) Not enough room between lines to use the specified increment.

NO SUCH LINES (Editor) The specified line(s) do not exist.

REGISTER ERROR (Assembler) Registers have not been specified with a PSH/PUL instruction: A register has been specified more than once in a PSH/PUL instruction: A register mis-match with an EXG/TFR instruction.

SEARCH FAILS (Editor) The string specified in the F (Find) command could not be found in the edit buffer.

SYMBOL TABLE OVERFLOW (Assembler) Symbol table will extend into the protected area of memory: There is not enough room between BEGTMP and USRORG for the edit buffer and symbol table: At least 300 hexadecimal bytes must be allowed for BEGTMP.

UNDEFINED SYMBOL (Assembler) Symbol was not listed in the label field or defined with an EQU statement.

Memory Map

DECIMAL ADDRESS	CONTENTS	HEX ADDRESS
0-1023	System Use	0-3FF
255	Direct Page RAM	0FF
1023	Extended Page RAM	3FF
1024-1535	Text Screen Memory	400-5FF
	Graphic Screen Memory	
1536-3071	Page 1	600-BFF
3072-4607	Page 2	C00-11FF
4608-6143	Page 3	1200-17FF
6144-7679	Page 4	1800-1DFF
7680-9215	Page 5	1E00-23FF
9216-10751	Page 6	2400-29FF
10752-12287	Page 7	2A00-2FFF
12288-13823	Page 8	3000-35FF
	Program and Variable Storage	
13824-16383		3600-3FFF
32768-40959	Extended Color BASIC	8000-9FFF
40960-49151	Color BASIC	A000-BFFF
49152-65279	Program Pak™ Memory	C000-FEFF
65280-65535	Input/Output	FF00-FFFF

Notes:

Error Messages

Abbreviation	Explanation
/0	Division by 0
AE	File already exists
AO	File already OPEN
BR	Bad record number
BS	Subscript out of range
CN	Can't continue
DD	Redimensioned array
DF	Disk space full
DN	Device number error
DS	Direct statement in file
ER	Write or Input past end of record
FC	Illegal function call
FD	Bad file data
FM	Bad file mode
FN	Bad filename
FO	Field overflow
FS	File structure incorrect
ID	Illegal direct
IE	Input past end of file
I/O	Input/Output error
LS	String too long
NE	File not found
NF	NEXT without FOR
NO	File not open
OB	Out of buffer space
OD	Out of data
OM	Out of memory
OS	Out of string space
OV	Overflow
RG	RETURN without GOSUB
SE	Set to non-fielded string
SN	Syntax error
ST	String formula too complex
TM	Type mismatch
UL	Undefined line
VF	Verification error
WP	Write-protected diskette

Notes:

Line Printer Variables

Variable	Hexadecimal Address	Decimal Address	Initial Value	
			Hexadecimal	Decimal
BAUD Rate MSB	0095	149	00	0
BAUD Rate LSB	0096	150	57	87
Line Delay MSB	0097	151	00	0
Line Delay LSB	0098	152	01	1
Comma Field Width	0099	153	10	16
Last Comma Field	009A	154	70	112
Line Printer Width	009B	155	84	132
Line Printer Position	009C	156	00	00

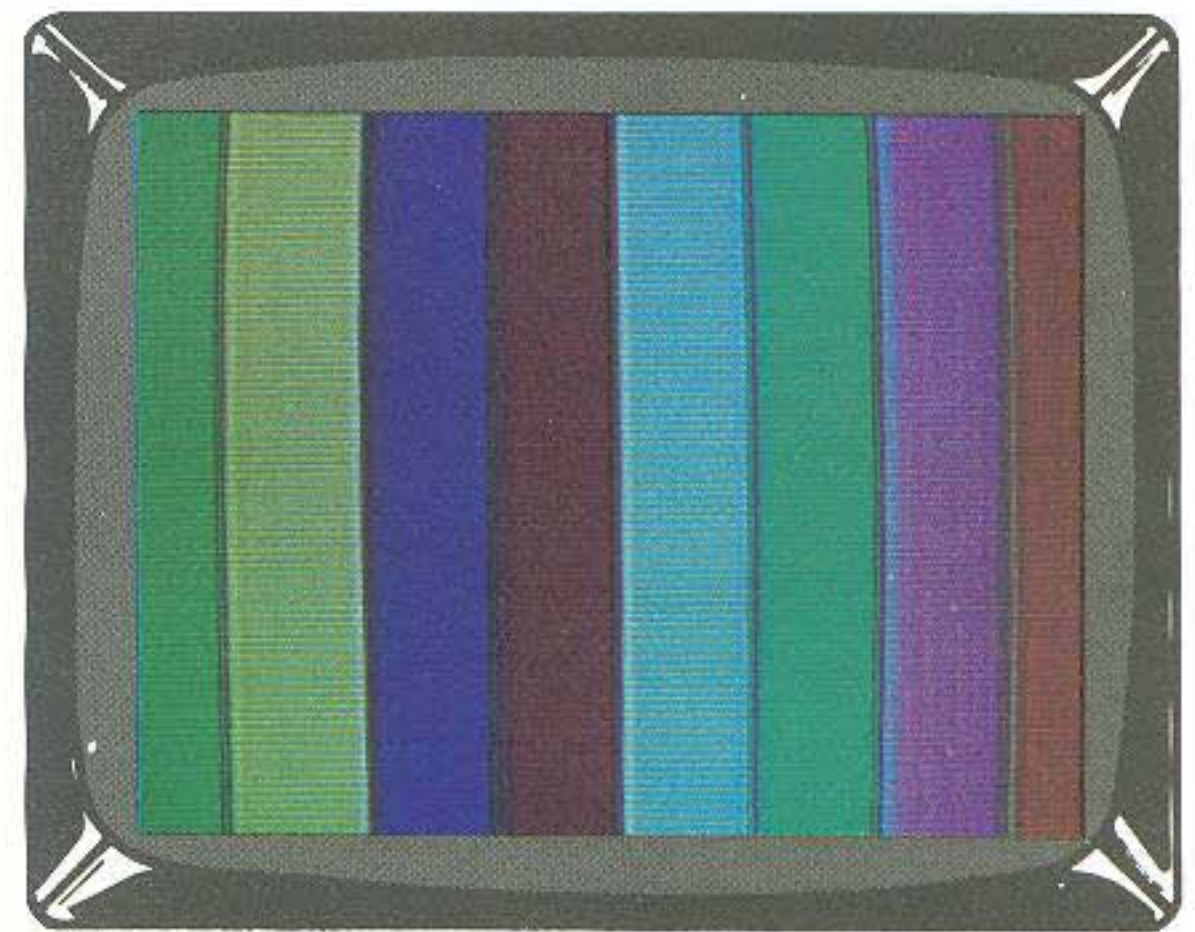
BAUD Rate	Decimal Value		Hexadecimal Value	
	MSB	LSB	MSB	LSB
120 BAUD	1	202	01	CA
300 BAUD	0	180	00	BE
600 BAUD	0	87	00	57
1200 BAUD	0	41	00	29
2400 BAUD	0	18	00	12

Line Delay	Decimal Value		Hexadecimal Value	
	MSB	LSB	MSB	LSB
.288 Seconds	64	0	40	00
.576 Seconds	128	0	80	00
1.15 Seconds	255	255	FF	FF

Line Width	Decimal Value		Hexadecimal Value	
	MSB	LSB	MSB	LSB
16 Characters/Line	16		10	
32 Characters/Line	32		20	
64 Characters/Line	64		40	
255 Characters/Line	255		FF	

Notes:

Color Adjustment Test



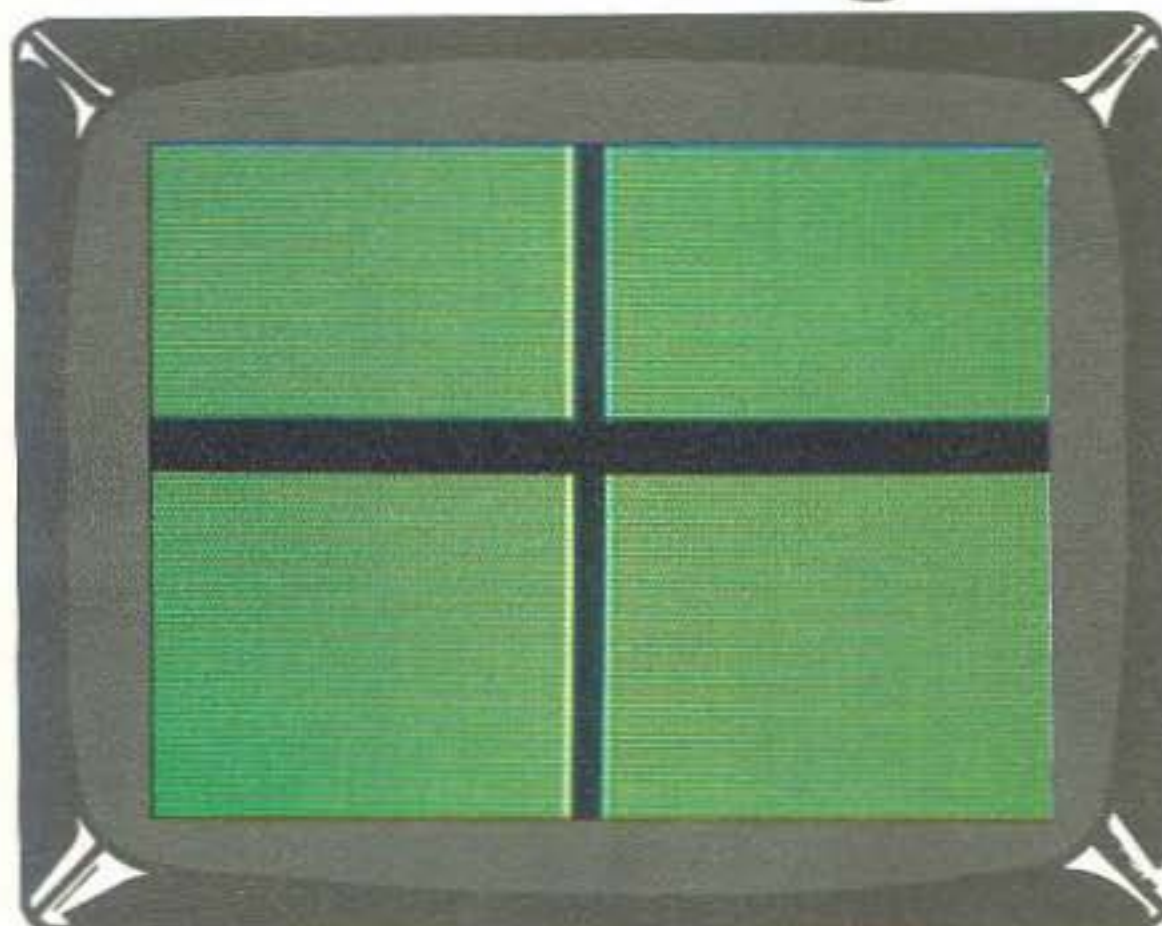
Color Adjustment Test Display

The following program will display the eight colors your TRS-80 Color Computer can produce. They are: green, yellow, blue, red, buff, cyan, magenta, and orange. The actual color tones produced by your set, and the degree of difference between tones will depend on *the quality and color adjustment of your television set* — not the Computer. By comparing the sample Color Test display with the colors on your screen, you can get a good idea of what the colors should be like.

```
5 FOR X = 0 TO 63
10 FOR Y = 0 TO 31
15 C = INT(X/8+1)
20 SET(X,Y,C)
25 NEXT Y,X
30 GOTO 30
```


Notes:

Video Centering Test



Video Centering Test Display

Since the Color Computer generates a rectangular image designed to fill most of your TV screen, it is important that your TV display be properly centered. Run the following program and use the horizontal — and vertical-centering controls to center the image as much as possible.

Don't worry if you can't get a perfectly centered image, or if you notice a slight distortion in certain areas of your TV screen. These minor variations depend on the condition of your TV set. If they are severe, we suggest you consult a qualified TV service technician.

```
10 CLS
15 FOR X = 0 TO 63
20 Y = 15
25 RESET (X,Y)
30 NEXT X
35 FOR Y = 0 TO 31
40 X = 31
45 RESET (X,Y)
50 NEXT Y
55 GOTO 55
```


Notes:

Specifications

AC Power Supply

Power Requirements	105-130 VAC, 60 Hz
Current Drain	0.18 Amps RMS

Microprocessor

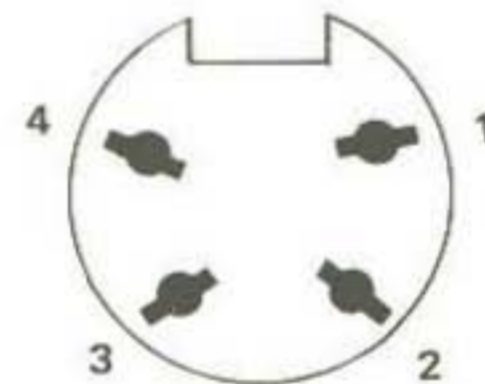
Type	6809E
Clock Rate	0.895 MHz

Serial Interface

Standard RS-232-C Signal		Pin #
CD	Carrier Detect (Status Input Line)	1
RD	Receive Data	2
GROUND	Zero Voltage Reference	3
TD	Transmit Data Out	4

RS-232-Pin Location

Looking from the outside at the RS-232-C jack on the COLOR COMPUTER.



Printer Software Requirements

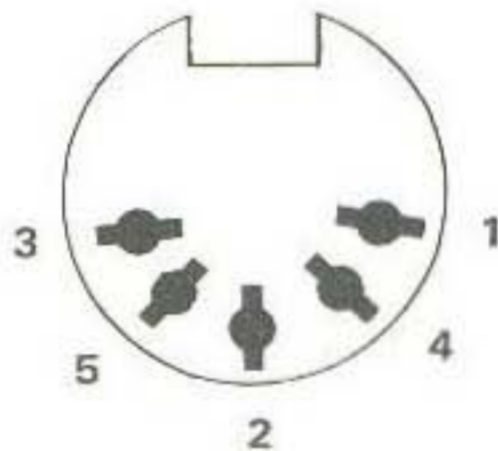
600 Baud
1 Start Bit (logical zero)
7 Data Bits (LSB first)
2 Stop Bits (logical one)
No Parity
132-Column Printer Width
Automatic Carriage Return at End of Line

Cassette Interface

Suggested Input Level for Playback from Recorder	1 to 5 Volts peak-to-peak at a minimum impedance of 220 Ohms
Typical Computer Output Level to Recorder	800 mV peak-to-peak at 1 K Ohms
Remote On/Off Switching Capability	0.5 A maximum at 6 VDC

Cassette Jack Pin Location

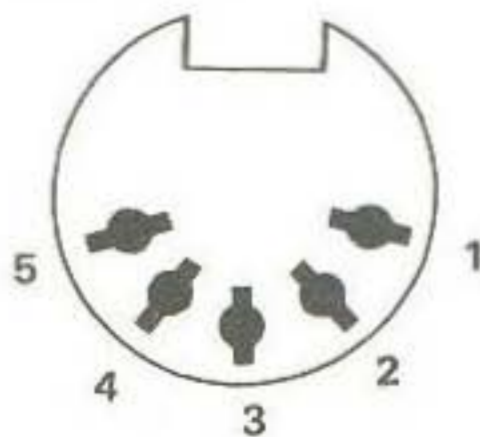
Looking at the outside of the jack on the Computer



1. Remote Control
2. Signal Ground
3. Remote Control
4. Input from Recorder's EARphone Jack
5. Output to Recorder's AUX or MIC Jack

Joystick Controller Jack Pin Location

Looking at the outside of the jack on the computer.




1. Comparator Input (Right-Left)
2. Comparator Input (Up-Down)
3. Ground
4. "Fire" button, High when open, Low when closed.
5. Vcc, current-limited +5VDC

Mini-Disk Specification


Type of disks	5 1/4" mini-diskettes Radio Shack Catalog Number 26-305 26-405 (package of three) or 26-406 (package of 10)
Disk Organization (Formatted disk)	Single-sided Double-density 35 tracks 18 sectors per track 256 data bytes per sector Directory on track 17
Operating Temperature	32 to 120 degrees Fahrenheit 0 to 49 degrees Centigrade
Memory Capacity Unformatted	218.8 kilobytes per disk 6.2 kilobytes per track
Soft sector (I/O sector/track)	179.1 kilobytes per disk 5.1 kilobytes per track
Data transmission speed	250 kilobits per second
Access Time Track to track Average Settling time	30 m sec. 463 m sec. 10 m sec.
Number of indexes	1
Weight of Disk Drive	3.8 kg.
Disk Controller	WD1793

Notes:



Definitions

**access**

The method in which information is read from or written to diskette; see direct access and sequential access.

**address**


A location in memory, usually specified as a two-byte hexadecimal number.

**array**



An organized set of elements which can be referenced in total or individually, using the array name and one or more subscripts.

**ASCII**


American Standard Code for Information Interchange. This method of coding is used to store textual data.

**ASCII format disk file**

Disk files in which each byte corresponds to one character of the original data.

**backup diskette**


An exact copy of the original: a "safe copy".

**BASIC**


Beginners' All-purpose Symbolic Instruction Code, the programming language which is stored in ROM in the TRS-80.

**baud**

Signaling speed in bits per second.

**binary**

Having two possible states, e.g., the binary digits 0 and 1.

**bit**

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

**buffer**

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

buffer field

A portion of the buffer which you define as the storage area for a buffer-field variable. Dividing a buffer into fields allows you to pass multiple values to and from disk storage.

byte

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

close

Terminate access to a disk file. Before re-accessing the file, you must reopen it.

data

Information that is passed to or from a program.

debug

To isolate and remove logical or syntax errors from a program.

decimal

Capable of assuming one of 10 states, e.g., the decimal digits 0, 1, . . . , 9.

default

An action or value which is supplied by the Computer 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.

destination

The device or address which receives the data during a data transfer operation.

direct access

See **random access**.

directory

A listing of the files which are contained on a disk.

disk drive or Mini Disk drive

The physical device which writes data onto diskettes and retrieves it.

diskette or disk

A magnetic recording medium for mass data storage.

drive number

An integer value from 0 to 3, specifying one of the Mini Disk drives.

end-of-file or EOF

A marker which indicates the end of a disk file, i.e., where the meaningful data ends and the unknown begins.

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.

field

A user-defined subdivision of a random access file-buffer, created and named with the **FIELD** statement.

file

An organized collection of related data.

file extension

An optional field in a file specification, consisting of a / followed by one alphabetic and up to two alphanumeric characters.

filename

A required field in a file specification, consisting of one alphabetic followed by up to eight alphanumeric characters. Filenames are assigned when a file is created or renamed.

format

To organize a new or magnetically erased diskette into tracks and sectors.

granule

The smallest unit of allocatable space on a disk, consisting of five sectors.

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.

input

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

machine language

The 6809 instruction set, usually specified in hexadecimal code. All higher-level languages must be translated into machine-language in order to be executed by the Computer.

octal

Capable of existing in one of eight states, for example, the octal digits are 0, 1, . . . , 7.

open

To prepare a file for access by assigning a sequential input, sequential output, or random I/O buffer to it.

physical record

The smallest amount of data which can be written to a disk file or read from it.

random access

Reading from a disk file or writing to it whereby there is no rule for predetermining the position from where the next item is to be obtained. Going "directly" to data.

random access memory or RAM

Semiconductor memory which can be addressed directly and either read from or written to.

read-only memory or ROM

Pre-programmed semiconductor memory which is directly addressable but can only be read, not written to.

sector

One-tenth of a track on a diskette, containing 256 bytes of storage; a "physical record".

sequential access

Reading from a disk file or writing to it "from start to finish", without being able to directly access a particular record in the file.

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".)

track

One of 35 concentric circles on the disk, each of which contains 18 sectors, or 256 bytes of storage. The tracks are not physical entities like grooves on a record; they are magnetic traces.

utility

A program or routine which serves a limited, specific purpose.

write-protect

To physically protect a disk from being written to by placing a tape over the write-protect notch.

Index

A	29
ABS	15
ABX	39
AC Power Supply	59
ADC	39
ADD	39
AND	39
AND CC	39
/AO	31
ASC	15
ASCII Character Codes	25
ASL	39
ASR	39
Assembler Switches	31
AUDIO	5
BACKUP	5
BCC	39
BCS	39
BEQ	40
BGE	40
BGT	40
BHI	40
BHS	40
BIT	40
BLE	40
BLKIN	19
BLKOUT	19
BLO	40
BLS	41
BLT	41
BMI	41
BNE	41
BPL	41
BRN	41
BSR	41
BVC	41
BVS	41
C	29, 33
Cassette Interface	60
Cassette Jack Pin Location	60
CHR\$	15
CHROUT	19
CIRCLE	5
CLEAR	5
CLOAD	5
CLOADM	5
CLOSE	5
CLR	42
CLS	5
CMP	42
COLOR	6
Color Adjustment Test	55
Color Codes	23
Color-Set Table	27
COM	42
CONT	6
Control Keys	21
COPY	6
COS	15

CSAVE	6
CSRDON	19
CVN	15
CWAI	42
D	29, 33
DAA	42
DATA	6
DEC	42
DEF FN	6
Definitions	47, 63
DEFUSR	6
DEL	6
DIM	6
DIR	6
Disk Extended Color BASIC	3
Display Mode Commands	37
DLOAD	7
DRAW	7
DRIVE	7
DSKCON	19
DSKIS	7
DSKINI	7
DSKOS	7
E	29, 33
EDIT	7
Editor Assembler Error Messages	47
Editor Commands	29
END	7
Entry Address	19
EOF	15
EOR	42
EQU	42
Error Messages	47, 51
Examination Mode Commands	37
EXEC	7
EXG	42
EXP	15
Extended Color BASIC	3
F	29
FCB	43
FCC	43
FIELD	8
Filename	3
FILES	8
FIRQ	46
FIX	15
FOR ... NEXT	8
FREE	15
Functions	15
G	33
GET	8
GOSUB	8
GOTO	8
Graphic Character Codes	23
Graphics Screen Resolution Table	27
H	30
HEX\$	16
I	30
IF ... THEN ... ELSE	8
/IM	31
INC	43
INKEY\$	16

INPUT	8
INSTR	8
INT	16
IRQ	46
JMP	43
JOYIN	19
Joystick Controller Jack Pin Location	60
JOYSTK	16
JSR	43
KILL	9
L	30, 33
LD	43
LEA	43
LEFT	16
LEN	16
LET	9
LIST	9
LLIST	9
LINE	9
LINE INPUT	9
Line Printer Variables	53
LOAD	9
LOADM	9
LOC	16
LOF	16
LOG	16
/LP	31
LSET	9
LSL	43
LSR	43
MEM	16
Memory Map	49
MERGE	9
Microprocessor	59
MIDS	10, 16
Mini-Disk Specification	61
MKNS	10
/MO	31
MOTOR	10
MUL	43
Musical Note/Number Codes	27
N	30
NEG	44
NEW	10
/NL	31
NMI	46
/NO	31
NOP	44
/NS	31
Number Base Mode Commands	37
ON ... GOSUB	10
ON ... GOTO	10
OPEN	10
Operators	21
OR	44
ORCC	44
ORG	44
P	30, 33
PAINT	10
PCLEAR	10
PCLS	10
PCOPY	10

PEEK	16
PLAY	10
PMODE	11
POINT	16
POKE	11
POLCAT	19
POS	16
PPOINT	17
PRESET	11
PRINT	11, 12
Printer Software Requirements	59
PRINT TAB	11
PRINT USING	11
PSET	12
PSHS	44
PULS	44
PULU	44
PUT	12
Q	30
R	30, 33
READ	12
REM	12
RENAME	12
RENUM	12
RESET	12
RESTART	45
RESTORE	12
RETURN	12
RIGHT\$	17
ROL	45
ROM Subroutines	19
ROR	45
RSET	12
RS-232-Pin Location	59
RTS	45
RUN	12
SAVE	13
SAVEM	13
SBC	45
SCREEN	13
SET	13
SETDP	45
Serial Interface	59
SEX	45
SGN	17
SIN	17
6809 Instructions	39
SKIPF	13
SOUND	13
Special Characters	21
Special Symbols	35
Specifications	59
/SS	31
ST	45
Start-Up	3
Statements	5
STOP	13
STRING\$	17
STR\$	17
SQR	17
SUB	45, 46
SWI	46

SWI2	46
SWI3	46
SYNC	46
T	30, 33
TAN	17
TFR	46
TH	34
TIMER	17
TROFF	13
TRON	13
TST	46
/WE	31
WRITE	13
U	34
UNLOAD	13
USR	17
V	30, 34
VAL	17
Variables	3, 15, 29, 33
VARPTR	17
VERIFY	13
Video Centering Test	57
Video Control Codes	23
WRTLDR	19
X	34
Y	34
ZBUG Commands	33

