MANUAL MANUAL MANUAL

The Compucolor 8001 CRT

Compucolor Corporation
P.O. Box 569
Norcross, Georgia 30071
Telephone 404/449-5879

FACTORY: (4Ø4) 449-5879

1922 -

Special note: the COMPUCOLOR uses 27Ø8 type EPROMS on its 24K ROM boards. If I desire to customize my unit, I can do so by means of inserting properly programmed 27Ø8's on that card.
For that, I'd need a decent 8Ø8Ø development system. There is no doubt that the CC8ØØ1 has potential for a powerful system with proper development.

DESIRABLE ADDITIONS: (not necessarily all at once.)

1. D.O.S. & EXTENDED BASIC WITH FILES.
2. RELOCATABLE MACRO-ASSEMBLER & DISASSEMBLER.
3. VARIABLE PRECISION/VARIABLE BASE ULTRA-CALCULATOR.
4. TEXT EDITOR/SELECTRIC I/O ROUTINES.
5. BASIC OR FORTRAN COMPILER. (VICE INTERPETER.)
6. 27Ø8 PROGRAMMER/2716 PROGRAMMER.
7. MORE POWERFUL PLOT SYSTEM INCLUDING 3-D DIAGRAM CAPABILITIES, MAYBE HIDDEN-LINE/PLANES, AND SUBSTITUTION-BY-ELEMENT FROM TABLES.
8. CURSOR BLINK-RATE AND CURSOR UPDATE FROM LIGHT-PEN OCCUR AT A RATE OF 2CPS... THIS COULD BE IMPROVED GREATLY BY INCREASING BLINK & UPDATE TO, SAY, 8CPS. Would this slow the 8Ø8Ø? (Yes, if these functions are serviced by interrupts; otherwise, No.)

BASIC 8001

T A B L E   O F   C O N T E N T S

BASIC 8001

INTRODUCTION

BASIC 8001 is a single-user, conversational programming language which uses simple English-type statements and familiar mathematical notations to perform an operation. BASIC 8001 is one of the simplest computer languages to learn and once learned has the facility of advanced techniques to perform more intricate manipulations or express a problem more efficiently.

BASIC 8001 is in incremental compiler which provides immediate translation and storage of user programs being input. This method decreases the response time of a RUN command and increases execution speed. BASIC 8001 has provision for alphanumeric character string, I/O and string variables, and allows user defined functions and assembly language subroutine calls from user BASIC 8001 programs.

BASIC 8001 can be run on any Intecolor 8001, Intecolor 8051 or Compucolor 8001 with a minimum of 8K of user workspace.

LOADING AND RUNNING BASIC 8001

BASIC 8001 is provided in ROM and runs in ROM. BASIC 8001 is initiated by typing the ESC key, then the W (BASIC) key. The dialogue described below is printed. This is a once-only dialogue and does not occur after an ESC key, and E key sequence. The READY message is printed after the ESC, E key sequence.

BASIC 8001 prints:

    BASIC 8001 V12/8/76 COPYRIGHT (C) 1976 BY CHARLES MUENCH

    MAXIMUM RAM ADDRESS?

The user then types the highest RAM address that he has available or wants to use and then keys a carriage return.

MIDDLE OF FIRST RAM CARD: 45055 = 65535-20480 = AFFF HEX
One extra RAM card is 49151 = 65535-16384 = BFFF hex.
Two extra RAM cards is 57343 = 65535- 8192 = DFFF hex.
Three extra RAM cards is 65529 = 65535-6 = FFF9 hex.

231 BYTES USED FOR INITIALIZATION OF BASIC 8001 STATUS.
247 BYTES USED AFTER DOING ANYTHING.

BASIC 8001 then prints the message,

    READY

and waits for a command or program line to be typed.

If BASIC 8001 has been initialized as above but has returned to the CRT O.S. (by CPU Reset Key), then BASIC 8001 can be recalled without disturbing existing programs by typing the ESC key, then the E key. BASIC 8001 will then print the message READY.

47103

If power fails, the CPU Reset key is hit or the unit is turned off, the unit returns to the CRT operating mode.

If the CPU Reset key or ESC delete keys are hit, the unit leaves BASIC 8001 and returns to the CRT operating mode. Any BASIC 8001 statement program is saved and can later be recalled if BASIC 8001 is re-entered by typing ESC, E.

BASIC 8001 has twenty-four (24) key word program statements, thirteen (13) editing and command statements, eighteen (18) mathematical functions, nine (9) string functions and eighteen (18) two-letter error messages. With these command and statement capabilities, BASIC 8001 is extremely simple to use and yet very versatile and powerful.

The next section provides an easy reference to BASIC 8001 capabilities. If the user is unfamiliar with BASIC 8001 language, then the remaining portion of this manual should be studied in sequence while having a terminal at your fingertips to run the example given. This manual should enable the user to become very proficient in BASIC 8001 when finished. Intelligent Systems Corporation and Compucolor Corporation have a number of BASIC 8001 programs on Floppy Tapes and are available at nominal prices. In addition, both companies will pay for BASIC 8001 programs that are provided on floppy tape when properly documented and accepted. Enjoy your self programming in BASIC 8001!

BASIC 8001

SUMMARY OF COMMANDS

1. BASIC 8001 STATEMENTS

The following summary of BASIC statements defines the general format for the statement and gives a brief explanation of its use.

DATA value list	Used in conjunction with READ to input data into an executing program.
DEF function (argument) = expression	Defines a user function to be used in the program.
DIM variable (n), variable (n,m), variable $(n), variable $(n,m)	Reserves space for lists and tables according to subscripts specified after variable name.
END	Placed at the physical end of the program to terminate program execution.
FOR variable=expression1 TO expression2 STEP expression3	Sets up a loop to be executed the specified number of times.
GOSUB line number	Used to transfer control to the first line of a subroutine.
GOTO line number	Used to unconditionally transfer control to other than the next sequential line in the program.
IF expression THEN / GOTO line number	Used to conditionally transfer control to the specified line of the program.
INPUT list	Used to input data from the terminal keyboard, promps with "?".
INPUT "string"; list	Used to input data without promp character.
LET variable = expression	Used to assign a value to the specified variable(s).
NEXT variable	Placed at the end of a FOR loop to return control to the FOR statement.
ON X GOSUB line number list	Call the Xth line number subroutine after GOSUB.
ON X GOTO line number list	Branch to the Xth line number after GOTO.

OUT I,X	Causes the X BYTE to be output to port I.
PLOT expression	Sends the one BYTE result of the expression to the 8001 CRT. The result must be between 0 and 255 binary.
POKE I,X	Causes the X BYTE to be placed in memory location $0 \leq$ I 32767. If I is negative then address is 65536 + I.
PRINT list	Used to output data to the terminal.
PRINT expression	Prints results of expression.
PRINT "string"	Prints a character string.
?	Equivalent to the word PRINT.
PRINT TAB(x)	Used to space to the specified column.
READ variable list	Used to assign the values listed in a DATA statement to the specified variables.
REM comment	Used to insert explanatory comments into a BASIC 8001 program.
RESTORE	Used to reset data block pointer so the same data can be used again.
RETURN	Used to return program control to the statement following the last executed GOSUB statement.
STOP	Used at the logical end of the program to terminate execution.
WAIT X,I,J	Causes the input port X to be read, exclusive OR'ed with BYTE J, and then AND'ed with BYTE I. The program will wait until the result is zero before continuing.

INPUT PORTS:

#1 - KEYBOARD, ASCII INPUT

#251 - LIGHT PEN TOUCH SWITCH

- X > 192 NO TOUCH
- 193 > X > 67 TOUCH, NO DETECT
- X < 3 TOUCH $\bar{c}$ DETECT

#252 - X-COORDINATE INPUT

#253 - Y-COORDINATE INPUT (INVERTED)

2. COMMANDS

The following key commands halt program execution, erase characters or delete lines.

Key	Explanation
CTRL/J or Line Feed	Terminates program execution. BASIC 8001 prints READY.
CTRL/M or RETURN	Must be typed to end every line typed in or to indicate the end of an INPUT.
:	A colon is used to separate multiple statements per line.
CTRL/K or ERASE LINE	Deletes the entire current line.
CTRL/L or ERASE PAGE	Erases the CRT screen, but does not change or disturb BASIC 8001 statements in any way.
CTRL/Z or CURSOR LEFT	Deletes the last character entered and echoes a cursor left.

The following commands list, load, save, erase and execute the program currently in core.

Command	Explanation
CLEAR	Sets the array and string buffers to nulls and zeroes.
CLEAR X Required for STRING data!	Sets space for string variable to X characters normally 50 characters.
LIST	Prints the user program currently in core on the list output device.
LIST line number	Prints the program from the line specified to the end.
LOAD I	Does a NEW and inputs the program on track #I from the READER input device.
LOAD ? I	Does not do a NEW but inputs and compares the program on track #I with what is existing in RAM Memory.
RUN	Executes the program in the buffer area.
RUN line number	Executes the program starting at line number specified.
SAVE I	Outputs the program in core to track #I of the WRITE output device.

SAVE I: LOAD?I (Very useful)

NEW	Erases the entire storage area.
CONT	Continues execution after CTRL/J is typed or after a STOP statement.

The following functions perform standard mathematical operations in BASIC 8001.

Name	Explanation
ABS(x)	Returns the absolute value of x.
ATN(x)	Returns the arctangent of x as an angle in radians in the range + or - pi/2.
CALL(x) THIS ~~GOTO~~ JMP MUST BE POKED IN AT -24575 (LO BYTE) and -24574 (Hi BYTE.)	Call the user machine language routine at location 0A000 HEX. A000 = -24576 = JMP A001 = -24575 = Lo A002 = -24574 = HI
COS(x)	Returns the cosine of x radians.
EXP(x)	Returns the value of e^x where e=2.71828.
FRE(x) DOES NOT INCLUDE FRE(x$) BYTES!	Returns number of free BYTEs not in use.
INT(x)	Returns the greatest integer less than or equal to x.
INP(x)	Returns a BYTE from input port $0 \leq x \leq 255$.
LOG(x)	Returns the natural logarithm of x.
PEEK(x) Same as POKE locations.	Returns a BYTE from memory address $0 \leq x \leq 32767$ or if X is negative the memory address is 65536- x.
POS(x)	Returns a value 0 to 79 current cursor position.
RND(x) Repeats after 1995 numbers.	Returns a random number between 0 and 1.
SGN(x)	Returns a value indicating the sign of x.
SIN(x)	Returns the sine of x radians.
SPC(x) DISTRUCTIVE TAB(x)	Causes x spaces to be generated.
SQR(x)	Returns the square root of x.
TAB(x)	Causes the cursor to tab to column number x when used in a print statement.
TAN(x)	Returns the tangent of x radians.

The string functions are:

Name	Explanation
ASC(x$)	Returns as a decimal number the seven-bit internal code for the first character of string (x$).
CHR$(x)	Generates a one-character string having the ASCII value of x.
FRE(x$)	Returns number of free string BYTES.
LEFT$(x$,I)	Returns left most I characters of string (x$).
LEN(x$)	Returns the number of characters in the string (x$).
MID$(x$,I,J)	Returns J characters of string (x$) starting at position I.
RIGHT$(x$,I)	Returns right most I characters of string (x$).
STR$(x)	Returns the string which represents the numeric value of x.
VAL(x$)	Returns the number represented by the string (x$).
CLEAR X	Reserves X bytes for string data. Default value is 5ϕ bytes. No single input can exceed 96 bytes.

ERROR MESSAGES

After an error occurs, BASIC 8001 returns to command level and types READY. Variable values and the program text remain intact, but the program cannot be continued and all GOSUB and FOR context is lost.

When an error occurs in a direct statement, no line number is printed.

Format of error messages:

Direct Statement	XX ERROR
Indirect Statement	XX ERROR IN YYYYY

In both of the above examples, "XX" will be the error code. The "YYYYY" will be the line number where the error occurred for the indirect statement.

The following are the possible error codes and their meanings:

ERROR CODE	MEANING
BS	Bad Subscript. An attempt was made to reference a matrix element which is outside the dimension of the matrix. This error can occur if the wrong number of dimensions are used in a matrix reference; for instance, LET A (1,1,1)=Z when A has been dimensioned DIM A(2,2).
DD	Double Dimension. After a matrix was dimensioned, another dimension statement for the same matrix was encountered. This error often occurs if a matrix has been given the default dimension 10 because a statement like A(I)=3 is encountered and then later in the program a DIM A(100) is found.
CF	Call Function error. The parameter passed to a math or string function was out of range. CF errors can occur due to:

a) a negative matrix subscript (LET A(-1)=0)

b) an unreasonably large matrix subscript (>32767)

c) LOG-negative or zero argument

d) SQR-negative argument

e) A B with A negative and B not an integer.

f) A CALL (X) before the address of the machine language subroutine has been patched in (see Pg.7)

g) calls to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC or ON...GOTO with an improper argument.

ID	Illegal Direct. You cannot use an INPUT or DEF statement as a direct command.
NF	NEXT without FOR. The variable in a NEXT statement corresponds to no previously executed FOR statement.
OD	Out of Data. A READ statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.
OM	Out of Memory. Program too large, too many variables, too many FOR loops, too many GOSUB's, too complicated an expression or any combination of the above.
OV	Overflow. The result of a calculation was too large to be represented in BASIC's number format. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.
SN	Syntax error. Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.
RG	RETURN without GOSUB. A RETURN statement was encountered without a previous GOSUB statement being executed.
US	Undefined Statement. An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist.
/0	Division by Zero.
CN	Continue error. Attempt to continue a program when none exists, an error occurred, or after a new line was typed into the program.
LS	Long String. Attempt was made by use of the concatenation operator to create a string more than 255 characters long.
OS	Out of String Space. Save your program on paper tape or cassette, reload BASIC and allocate more string space or use smaller strings or less string variables. ALLOCATE STRING SPACE WITH CLEAR X. See Pg. 5!
ST	String Temporaries. A string expression was too complex. Break it into two or more shorter ones.
TM	Type Mismatch. The left hand side of an assignment statement was a numeric variable and the right hand side was a string, or vice versa; or, a function which expected a string argument was given a numeric one or vice versa.

UF	Undefined Function. Reference was made to a user defined function which had never been defined.

BASIC 8001 ARITHMETIC

I. NUMBERS

BASIC treats all numbers (real and integer) as decimal numbers---that is, it accepts any decimal number and assumes a decimal point after an integer. The advantage of treating all numbers as decimal numbers is that any number or symbol can be used in any mathematical expression without regard to its type. Numbers used must be in the approximate range $10^{-38} < N < 10^{+38}$.

In addition to integer and real formats, a third format is recognized and accepted by BASIC 8001. This format is called exponential or E-type notation, and in this format, a number is expressed as a decimal number times some power of 10. The form is:

xxEn

where E represents "times 10 to the power of"; thus the number is read "xx times 10 to the power of n". For example:

$23.4E2=23.4*10^2 = 2340$

Data may be input in any one or all three of these forms. Results of computations are output as decimals if they are within the range .01_n_999999; otherwise, they are output in E format. Numbers are stored up to 24 bits of significance. If a number with more than 24 bits is entered, it is truncated and stored as 24 bits. BASIC 8001 handles six significant digits in normal operation and prints 6 decimal digits as illustrated below:

Value Typed In	Value Output by BASIC 8001
.01	.01
.0099	9.90000E-03
999999	999999
1000000	1.00000E+06

BASIC automatically suppresses the printing of leading and trailing zeroes in integer and decimal numbers, and, as can be seen from the preceding examples, formats all exponential numbers in the form:

(sign) x.xxxxxE(+ or -)n

where x represents the number carried to six decimal places, E stands for "times 10 to the power of", and n represents the exponential value. For example:

-3.47021E+08 is equal to -347,021,000
 7.26000E-04 is equal to .00726

Floating point format is used when storing and calculating most numbers.

NOTE

Because core size limitations prohibit the storage of infinite binary numbers, some numbers cannot be expressed exactly. In BASIC 8001, accuracy is approximately 5-½ digits, and errors in the 6th digit can occur. For example, .999998 as a result of some functions may be equal to 1. Discrepancies of this type are magnified when such a number is used in mathematical operation.

II. VARIABLES

A variable in BASIC 8001 is an algebraic symbol representing a number, and is formed by a single letter, a letter optionally followed by a single digit or by double letters. For example:

NOTE: Variables may be a string of characters many long

Long variables are very useful for inherant documentation in a program.

Acceptable Variables	Unacceptable Variables
ACCEPTABLE=8001	
I	2C-a digit cannot begin a variable.
B3	11-numbers alone cannot form a variable.
AB	
X	
DALE=69	

EXCLUDES ANYTHING WHICH RESEMBLES A BASIC COMMAND

ONLY 2 LEFTMOST CHARACTERS ARE SIGNIFIGANT.

Subscripted and string variables are described in later sections. The user may assign values to variables either by indicating the values in a LET statement, or by inputting the values as data; these operations are discussed in another chapter.

The value assigned to a variable does not change until the next time a statement is encountered that contains a new value for that variable. All variables are set equal to zero (0) when the RUN command is issued. It is only necessary to assign a value to a variable when an initial value other than zero is required. However, good programming practice would be to set variables equal to 0 wherever necessary. This ensures that later changes or additions will not misinterpret values.

III. SUBSCRIPTED VARIABLES

In addition to the simple variables described in the preceding section, BASIC 8001 allows the use of subscripted variables. Subscripted variables provide additional computing capabilities for dealing with lists, tables, matrices, or any set of related variables. In BASIC 8001 variables are allowed from 1 to 31 subscripts.

The name of a subscripted variable is any acceptable BASIC 8001 variable name followed by one or more integer expressions in parentheses within the range 0-32767. For example, a list might be described as A(I) where I goes from 0 to 5 as shown below:

A(0),A(1),A(2),A(3),A(4),A(5)

This allows reference to each of the six elements in the list, and can be considered a one dimensional algebraic matrix as follows:

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)

A two-dimensional matrix B (I,J) can be defined in a similar manner:

B(0,0),B(0,1),B(0,2),. . .,B(OJ),. . .,B(I,J)

and graphically illustrated as follows:

B(0,0)	B(0,1)	B(0,2)	B(0,3)	B(0,J)
B(1,0)	B(1,1)	B(1,2)	B(1,3)	B(1,J)
B(2,0)	B(2,1)	B(2,2)	B(2,3)	B(2,J)
B(3,0)	B(3,1)	B(3,2)	B(3,3)	B(3,J)
	.			
	.			
	.			
	.			
B(I,0)	B(I,1)	B(I,2)	B(I,3)	B(I,J)

Subscripts used with subscripted variables throughout a program can be explicitly stated or be any legal expression. If the value of the expression is non-integer, the value is truncated so that the subscript is an integer.

It is possible to use the same variable name as both a subscripted and unsubscripted variable. Both A and A(I) are valid variables and can be used in the same program. The variable A has no relationship to any element of the matrix A(I). BASIC 8001 will accept the same variable name as both a singly and a doubly subscripted variable name in the same program.

Character strings may also be subscripted variable arrays, and may have the same variable name i.e., A$(I).

A Dimension (DIM) statement is used with subscripted variables to define the maximum number of elements in a matrix. ("Matrix" is the subscripted variable.) The DIM statement is discussed in a later paragraph.

If a subscripted variable is used without appearing in a DIM statement, it is assumed to be dimensioned to length 10 in each dimension (that is, having eleven elements in each dimension, 0 through 10). However, all matrices should be correctly dimensioned in a program.

IV. EXPRESSIONS

An expression is a group of symbols which can be evaluated by BASIC 8001. Expressions are composed of numbers, variables, functions, or a combination of the preceding separated by arithmetic or relational operators.

The following are examples of expressions acceptable to BASIC 8001:

Arithmetic Expressions	String Expressions
4	A$+B$+"ABC"
A7*(B^2+1)	

Not all kinds of expressions can be used in all statements, as is explained in the sections describing the individual statements.

V. ARITHMETIC OPERATIONS

BASIC 8001 performs addition, subtraction, multiplication, division and exponentiation. Formulas to be evaluated are represented in a format similar to standard mathematical notation. The five operators used in writing most formulas are:

Symbol Operator	Example	Meaning
OR		Logical and bitwise "OR"
AND		Logical and bitwise "AND"
NOT		Logical and bitwise "NOT"
+	A + B	Add B to A
-	A - B	Subtract B from A
*	A * B	Multiply A by B
/	A / B	Divide A by B
^	A ^ B	Exponentiation (Raise A to the Bth power)

Unary plus and minus are also allowed, e.g., the - in -A+B or the + in +X-Y. Unary plus is ignored. Unary minus is treated as a zero minus the variable, e.g., -A+B would be handled as 0-A+B.

VI. PRIORITY OF ARITHMETIC OPERATIONS

When more than one operation is to be performed in a single formula, as is most often the case, rules are observed as to the precedence of the operators.

In any given mathematical formula, BASIC 8001 performs the arithmetic operations in the following order of evaluation:

1. Parentheses receive top priority. Any expression within parentheses is evaluated before an unparenthesized expression.

2. In the absence of parentheses, the order of priority is:

    a. Exponentiation (proceeds from left to right).

    b. Unary minus.

    c. Multiplication and Division (of equal priority).

    d. Addition and Subtraction (of equal priority).

    e. Logical operators in the order NOT, AND, then OR.

3. If either 1 or 2 above does not clearly designate the order of priority, then the evaluation of expressions proceeds from left to right.

The expression A^B^C is evaluated from left to right as follows:

1. A^B = step 1

2. (result of step 1)^C = answer

The expression A/B*C is also evaluated from left to right since multiplication and division are of equal priority:

1. A/B = step 1

2. (result of step 1)*C = answer

The expression A+B*C^D is evaluated as:

1. C^D = step 1

2. (result of step 1)*B = step 2

3. (result of step 2)+A = answer

Parentheses may be nested, or enclosed by a second set (or more) of parentheses. In this case, the expression within the innermost parentheses is evaluated first, and then the next innermost, and so on, until all have been evaluated.

In the following example:

```
A=7*((B^2+4)/X)
```

The order of priority is:

```
1.  B^2                  = step 1
2.  (result of step 1)+4 = step 2
3.  (result of step 2)/X = step 3
4.  (result of step 3)*7 = A
```

Parentheses also prevent any confusion or doubt as to how the expression is evaluated. For example:

```
A*B^2/7+B/C*D^2
((A*B^2)/7+((B/C)*D^2)
```

Both of these formulas are executed in the same way, but the second is easier to understand.

Spaces may be used in a similar manner. Since the BASIC 8001 interpreter ignores spaces (except when enclosed in quotation marks), the two statements:

```
1Ø LET B = D^2 + 1
1ØLETB=D^2+1
```

are identical, but spaces in the first statement provide ease in reading. When the statement is subsequently printed, extra spaces are ignored.

VII. RELATIONAL OPERATORS

Relational operators allow comparison of two values and are used to compare arithmetic expressions or strings in an IF. . . THEN statement. The relational operators are:

Mathematical Symbol	BASIC 8001 Symbol	Example	Meaning
=	=	A = B	A is equal to B.
<	<	A < B	A is less than B.
≤	<= or =<	A<= B	A is less than or equal to B.
>	>	A > B	A is greater than B.
≥	>= or =>	A>= B	A is greater than or equal to B.
≠	<> or ><	A<> B	A is not equal to B.

The symbols =<, => ,>< are accepted by BASIC 8001 but are converted to <=, >=, and <> and are shown in that form in a listing.

All string variables initialize with a 5Ø-byte potential length. ~~It appears I cannot change this in any way, not by concatenation or anything.~~ Change with CLEAR X statement, see Pg. 5.

CLEAR X RESERVES X BYTES FOR STRING DATA

I. STRINGS

The previous section described the manipulation of numerical information only; however, BASIC 8001 also processes information in the form of character strings. A string, in this context, is a sequence of characters treated as a unit. A string can be composed of alphabetic, numeric, or alphanumeric characters. (An alphanumeric string is one which contains letters, numbers, spaces or any combination of characters.) A character string can be 255 characters long. Strings cannot be typed on more than one terminal line since a carriage return terminates the command.

II. STRING VARIABLES

Any variable name followed by a dollar sign ($) character indicates a string variable. For example:

```
A$
C7$
```

are simple string variables and can be used, for example, as follows:

```
LET A$="HELLO"
PRINT A$
```

Note that the string variable A$ is separate and distinct from the variable A.

In BASIC 8001, all control characters above control code F (or 6) are legal within Quotes (") except for the following:

Control Code K or 11 or erase line
Control Code L or 12 or erase page
Control Code M or 13 or return
Control Code Z or 26 or cursor left

III. SUBSCRIPTED STRING VARIABLES

Any list of matrix variable name followed by the $ character denotes the string form of that variable. For example:

```
V$(n)        M2$(n)
C$(m,n)      G1$(m,n)
```

where m and n indicate the position of the matrix element within the whole.

The same name can be used as a numeric variable and as a string variable in the same program with no restriction. A one- and a two-dimensional matrix can have the same name in the same program. For example:

A	A(n)	A(m,n)
A$	A$(m,n)	A$(m,n(

can all be used in the same program.

String lists and matrices are defined with the DIM statement as are numerical lists and matrices.

IV. STRING OPERATIONS

Concatenation

Concatenation puts one string after another without any intervening characters. It is specified by a plus sign (+) and works only with strings. The maximum length of a concantenated string is 255 characters.

For example:

```
1Ø READ A$, B$, C$
2Ø DATA "11", "33", "22"
3Ø LET D$ = A$+C$+B$
35 PRINT D$
4Ø END
RUN
112233
```

V. RELATIONAL OPERATIONS

When applied to string operands, the relational operators indicate alphabetic sequence. The comparison is done on the basis of the ASCII value associated with each character in the strings being compared. For example:

```
55 IF A$<B$ THEN 100
```

When line 55 is executed, the first characters of each string (A$ and B$) are compared, then the second characters of each string and so on until the character in A$ is less than the character in B$. Then execution continues at line 100. Essentially, the strings are compared for alphabetic order. The next page contains a list of the relational operators and their string interpretations.

In any string comparison, trailing blanks are ignored (i.e., "ABC" is equivalent to "ABC ").

FRE(X)=AVAILABLE BYTES OF PROGRAM MEMORY.
FRE(X$)=AVAILABLE BYTES OF STRING MEMORY.
TOTAL AVAILABLE MEMORY IS FRE(X)+FRE(X$)...can adjust & transfer by CLEAR X.

BASIC 8001

Relational Operators Used With
String Variables

Operator	Example	Meaning
=	A$ = B$	The strings A$ and B$ are alphabetically equal.
<	A$ < B$	The string A$ alphabetically precedes B$.
>	A$ > B$	The string A$ alphabetically follows B$.
<= or =<	A$ <= B$	The string A$ is equivalent to or precedes B$ in alphabetical sequence.
>= or =>	A$ >= B$	The string A$ is equivalent to or follows B$ in alphabetical sequence.
<> or ><	A$ <> B$	The strings A$ and B$ are not alphabetically equal.

BASIC 8001 IMMEDIATE MODE

I. USE OF IMMEDIATE MODE FOR STATEMENT EXECUTION

It is not necessary to write a complete program to use BASIC 8001. Most of the statements discussed in this manual can be included in a program for later execution or given on-line as commands, which are immediately executed by the 8080 CPU. This latter facility makes BASIC 8001 an extremely powerful calculator.

BASIC 8001 distinguishes between lines entered for later execution and those entered for immediate execution solely by the presence (or absence) of a line number. Statements which begin with line numbers are stored; statements without line numbers are executed immediately upon being entered to the system. Thus the line:

```
1Ø PRINT "THIS IS A COMPUCOLOR 8001"
```

produces no action at the console upon entry, while the statement:

```
PRINT "THIS IS A  COMPUCOLOR 8001"
```

causes the immediate output:

```
THIS IS A COMPUCOLOR 8001
```

II. PROGRAM DEBUGGING

Immediate mode operation is especially useful in two areas: program debugging and the performance of simple calculations in situations which do not occur with sufficient frequency or with sufficient complications to justify writing a program.

In order to facilitate debugging a program, STOP statements can be liberally placed throughout the program. Each STOP statement causes the program to halt, at which time the various data values can be examined and perhaps changed in immediate mode. The

```
GO TO xxxxx
```

command is used to continue program execution (where xxxxx is the number of the next program line to be executed). GOSUB and IF commands could also be used. The values assigned to variables when the RUN command was executed remain intact until a NEW, CLEAR or another RUN command is executed.

If the STOP occurs in the middle of a FOR loop, modifications cannot be made to the section of the program preceding the FOR.

When using immediate mode, nearly all the standard statements can be used to generate or print results.

If CTRL/J or linefeed is used to halt program execution, the GO TO XXXX or CONT command can be used to continue execution, since CTRL J or linefeed does print the number of the line where execution stopped. It is easy to know where to resume the program.

III. MULTIPLE STATEMENTS PER LINE

Multiple statements can be used on a single line in immediate mode. For example:

```
A=1:PRINT A
1
```

Program loops are allowed in immediate mode; thus a table of square roots can be produced as follows:

```
  FOR I=1 TO 1Ø: PRINT I, SQR (I):NEXT I

 1                  1
 2                  1.41421
 3                  1.732Ø5
 4                  2
 5                  2.236Ø7
 6                  2.44949
 7                  2.64575
 8                  2.82843
 9                  3
 1Ø                 3.16228
READY
```

IV. RESTRICTIONS ON IMMEDIATE MODE

The INPUT statement cannot be used in immediate mode and such use results in the following error message:

```
ID ERROR
READY
```

Certain commands, while not illegal, make no logical sense when used in immediate mode. Commands in this category are DEF, DIM and DATA.

Also since user functions are not defined until the program is executed, function references in immediate mode cause an error unless the program containing the definition was previously executed.

Thus, the following dialogue might result if a function was defined in a user program and then referenced in immediate mode.

```
1Ø DEF FNA(X) = X^2 + 2*X:REM SAVED STATEMENT
PRINT FNA(1):REM IMMEDIATE MODE

UF ERROR
READY
```

but if the sequence of statements is:

```
10 DEF FNA(X) = X^2+2*X:REM SAVED STATEMENT
RUN

READY

PRINT FNA(1)
3

READY
```

the immediate mode statement is executed.

BASIC 8001 STATEMENTS

A user program is composed of lines of statements containing instructions to BASIC 8001. Each line of the program begins with a line number that identifies that line as a statement and indicates the order of statement execution. Each statement starts with an English word specifying the type of operation to be performed. The statement lines are terminated with the RETURN key which is non-printing.

I. STATEMENT NUMBERS

An integer number is placed at the beginning of each line in a BASIC 8001 program. BASIC 8001 executes the statements in a program in numerically consecutive order regardless of the order in which they were typed. Statement numbers must be within the range 0 to 65529. When first writing a program, it is advisable to number lines in increments of five or ten to allow insertion of forgotten or additional lines when debugging the program.

All BASIC 8001 statements and computations must be written on a single line; they cannot be continued onto a following line. However, more than one statement may be written on a single line when each statement after the first is preceded by a colon (:). For example:

```
1Ø INPUT A,B,C
```

is a single statement line, whereas

```
2Ø LET X=11: PRINT X,Y,Z: IF X=A THEN 1Ø
```

is a multiple statement line containing three statements: LET, PRINT, and IF. Most statements may be used anywhere in a multiple statement line;, exceptions are noted in the discussion of each statement. Only the first statement on a line can (and must) have a line number. It should be remembered that program control cannot be transferred to a statement within a line, but only to the first statement of a line.

II. REMARK STATEMENT

It is often desirable to insert notes and messages within a user program. Such data as the name and purpose of the program, how to use it, how certain parts of the program work, and expected results at various points are useful things to have present in the program for ready reference by anyone using that program.

The REMARK or REM statement is used to insert remarks or comments into a program without these comments affecting execution. Remarks do, however, use core area which may be needed by an exceptionally long program.

The REMARK statement must be preceded by a line number and may be used anywhere in a multiple statement line. The message itself can contain any printing character on the keyboard. BASIC 8001 completely ignores anything on a line following the letters REM. (The line number of a REM statement can be used in a GOTO or GOSUB statement, see sections pertaining to destination of a jump in the program execution.) Typical REM statements are shown below:

```
10 REM- THIS PROGRAM COMPUTES THE
11 REM- ROOTS OF A QUADRATIC EQUATION
```

III. THE ASSIGNMENT STATEMENT - LET

The LET statement assigns a value to the specified variable(s). The general format of the LET statement is:

```
LET variable = expression
```

where variable is a numeric or string variable and expression is an arithmetic or string expression. All items in the statement must be either string or numeric; they cannot be mixed. The word LET is optional.

The LET statement does not indicate algebraic equality, but performs calculations within the expression (if any) and assigns the value to the variable.

The meaning of the equal (=) sign should be clarified. In algebraic notation, the formula X=X+1 is meaningless. However, in BASIC 8001 (and most computer languages), the equal sign designates replacement rather than equality. Thus, this formula is actually translated: "add one to the current value of X and store the new result back in the same variable X". Whatever value has previously been assigned to X will be combined with the value 1. An expression such as A=B+C instructs the computer to add the values of B and C and store the result in a third variable A. The variable A is not being evaluated in terms of any previously assigned value, but only in terms of B and C. Therefore, if A has been assigned any value prior to its use in this statement, the old value is lost; it is instead replaced by the value B+C.

Example: X=Y=Z=98 is not evaluated for X=98, Y=98, Z=98... rather, it is evaluated logically where we test if Y=Z=98 and assign result to X.

LET X=2	Assigns the value 2 to the variable X.
LET X=X+1+Y	Adds 1 to the current value of X then adds the value of Y to the result and assigns that value to X.

IV. THE DIMENSION STATEMENT - DIM

The DIMension statement is used to define the maximum number of elements in a matrix. The DIM statement is of the form:

```
DIM variable(n),   variable(n,m),   variable$(n),   variable$(n,m)
```

where variables specified are indicated with their maximum subscript value(s).

For example:

```
1Ø DIM X(5), Y(4,2), A(1Ø ,1Ø)
12 DIM A4(1ØØ), A$(25)
```

Only integer constants (such as 5 or 5070) can be used in DIM statements to define the size of a matrix. Variables cannot be used to specify the bounds of arrays. Any number of matrices can be defined in a single DIM statement as long as their representations are separated by commas.

The first element of every matrix is automatically assumed to have a sub-script of zero. Dimensioning A(6,10) sets up room for a matrix with 7 rows and 11 columns. This zero element is illustrated in the following program:

```
1Ø REM - MATRIX CHECK PROGRAM
2Ø DIM A(6,1Ø)
3Ø FOR I=Ø TO 6
4Ø LET A(I,Ø) = I
5Ø FOR J=Ø TO 1Ø
6Ø LET A(Ø,J) = J
7Ø PRINT A(I,J);
8Ø NEXT J:PRINT:NEXT I
9Ø END

RUN
Ø    1    2    3    4    5    6    7    8    9    1Ø
1    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø
2    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø
3    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø
4    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø
5    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø
6    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø

READY
```

Notice that a variable has a value of zero until it is assigned another value.

Whenever an array is dimensioned (n,m), the matrix is allocated m+1, n+1 elements. Core space can be conserved by using the 0th element of the matrix. For example, DIM A(5,9) dimensions a 6 x 10 matrix which would then be referenced beginning with the A(0,0) element.

The size and number of matrices which can be defined depend upon the amount of storage space available.

A DIM statement can be placed anywhere in a multiple statement line and can appear anywhere in the program. A matrix can only be dimensioned once. DIM statements need not appear prior to the first reference to an array, although DIM statements are generally among the first statements of a program to allow them to be easily found if any alterations are later required.

All arrays specified in DIM statements are allocated space when the RUN command is executed.

V. PLOT STATEMENT

The PLOT Statement is used to output the 8 bit BYTE value of an expression to the CRT Screen. The general format of the PLOT Statement is:

```
10 PLOT expression
```

The expression can be any combination of variables which will evaluate to a positive value between 0 and 255.

The following example will plot a point on the CRT Screen at Location 80, 96 (X,Y):

```
10 X=80 : Y=96
20 PLOT 2 : REMARK THE 8001 PLOT MODE CODE
30 PLOT X : PLOT Y : REMARK PLOTS POINT AT 80, 96
40 PLOT 255 : REMARKS THE 8001 PLOT MODE ESCAPE CODE
```

As another example enter:

```
PLOT 65
A
READY
```

PLOT 65: PLOT66: PLOT67: PLOT68: PLOT69 : PLOT70: PLOT71
ABCDEFG
READY BASIC DOES NOT (L.F.)(C.R.) BETWEEN PLOTS.

It can be seen that (since 65 is the decimal ASCII value for A) PLOT 65 is the same as PRINT "A";

VI. PRINT STATEMENT

The PRINT statement is used to output data to the terminal. The general format of the PRINT statement is:

1) PRINT list

The list is optional and can contain expressions, text strings, or both.

When used without the list, the PRINT statement:

```
25 PRINT
```

DOES NOT ERASE, JUST SKIPS DOWN.

causes a blank line to be output on the 8001 CRT Screen (a carriage return/ line feed operation is performed).

2) PRINT Expression

PRINT statements can be used to perform calculations and print results. Any expression within the list is evaluated before a value is printed. For example:

```
10 LET A=1 : LET B=2: LET C=3+A
20 PRINT
30 PRINT A+B+C
RUN

7

READY
```

All numbers are printed with a preceding and following blank space.

The PRINT statement can be used anywhere in a multiple statement line. For example:

```
10 A=1: PRINT A: A=A+5: PRINT: PRINT A
```

prints the following on the terminal when executed:

```
1

6

READY
```

Notice that the terminal performs a carriage return/line feed at the end of each PRINT statement. Thus the first PRINT statement outputs a 1 and a carriage return/line feed; the second PRINT statement the blank line; and the third PRINT statement, a 6 and another carriage return/line feed.

3) PRINT Strings

The PRINT statement can be used to print a message or string of characters, either alone or together with the evaluation and printing of numeric values. Characters are indicated for printing by enclosing them in double quotation marks. For example:

```
10 PRINT "TIME'S UP"
20 PRINT "NEVERMORE"
RUN
TIME'S UP
NEVERMORE

READY
```

As another example, consider the following line:

```
40 PRINT "AVERAGE GRADE IS";X
```

which prints the following (where X is equal to 83.4):

```
AVERAGE GRADE IS 83.4
```

When a character string is printed, only the characters between the quotes appear; no leading or trailing spaces are added. Leading and trailing spaces can be added within the quotation marks using the keyboard space bar; spaces appear in the printout exactly as they are typed within the quotation marks.

When a comma separates a text string from another PRINT list item, the item is printed at the beginning of the next available print zone. Semicolons separating text strings from other items are ignored. Thus, the previous example could be expressed as:

```
40 PRINT "AVERAGE GRADE IS" X
```

and the same printout would result. A comma or semicolon appearing as the last item of a PRINT list always suppresses the carriage return/line feed operation.

BASIC 8001 does an automatic carriage return/line feed if a string is printing past column 80.

4) Use of "," and ";"

BASIC 8001 considers the 8001 CRT Screen to be divided into ten zones of eight spaces each. When an item in a PRINT statement is followed by a comma, the next value to be printed appears in the next available print zone. For example:

```
10 LET A=3: LET B=2
20 PRINT A,B,A+B,A*B,A-B,B-A
```

When the preceding lines are executed, the following is printed:

```
3       2       5       6       1       -1
```

Notice each character is 8 spaces from the next character.

Two commas together in a PRINT statement cause a print zone to be skipped. For example:

```
10 LET A=1: LET B=2
20 PRINT A,B,,A+B
RUN
1       2               3

READY
```

If the last item in a PRINT statement is followed by a comma, no carriage return/line feed is output, and the next value to be printed (by a later PRINT statement) appears in the next available print zone. For example:

```
1Ø A=1:B=2:C=3
2Ø PRINT A, :PRINT B: PRINT C
RUN
1             2
3

READY
```

If a tighter packing of printed values is desired, the semicolon character can be used in place of the comma. A semicolon causes no further spaces to be output other than the leading and trailing space automatically output with each number. A comma causes the print head to move at least one space to the next print zone or possibly perform a carriage return/line feed. The following example shows the effects of the semicolon and comma.

```
1Ø LET A=1/ B=2/ C=3
2Ø PRINT A;B;C;
3Ø PRINT A+1;B+1;C+1
4Ø PRINT A,B,C
RUN
1  2  3  2  3  4
1     2        3

READY
```

The following example demonstrates the use of the formatting characters , and ; with text strings:

```
12Ø PRINT "STUDENT NUMBER"X, "GRADE ="G;"AVE.  ="A;
13Ø PRINT "NO. IN CLASS ="N
```

could cause the following to be printed (assuming calculations were done prior to line 130):

```
STUDENT NUMBER 119Ø5Ø   GRADE = 87 AVE. = 85.44 NO. IN CLASS = 26
```

5) PRINT Statement - TAB Function

The TAB function is used in a PRINT statement to write spaces to the specified column on the output device. The columns on the output devices are numbered 1 to 80.

The form of the command is:

```
PRINT TAB(x)
```

where (x) is the column number in the range 0-255. (If X exceeds 80, however, every other consecutive line is tabbed until the number of spaces to be output is less than or equal to 80). If the column number specified is greater than 255 or negative, an error message is printed as follows:

```
CF ERROR
READY
```

If (x) is non-integer, only the integer portion of the number is used.

If the column number (x) specified is less than or equal to the current column number, the TAB function has no effect.

VII. INPUT STATEMENT

The INPUT statement is used when data is to be input from the terminal keyboard during program execution. The form of the statement is:

1) INPUT list

where list is a list of variable names separated by commas.

For example:

```
1Ø INPUT A,B,C
```

causes the computer to pause during execution, print a question mark, and wait for input of three numeric values separated by commas. The values are input to the computer by typing the RETURN key.

If too few values are entered, BASIC 8001 prints another ? to indicate that more data is needed. If too many values are typed, the excess data on that line is ignored and the message below is printed but program still continues. The values entered in response to the INPUT statement cannot be continued on another line and are terminated by the RETURN key. Values must be separated by commas, if more than one value is input on the same line.

When there are several values to be entered via the INPUT statement, it is helpful to print a message explaining the data needed. For example:

```
1Ø PRINT "YOUR AGE IS";
2Ø INPUT A
```

2) INPUT "string"; list

The INPUT statement can also contain quoted strings. The above example could be written:

```
10 INPUT "YOUR AGE IS?";A
```

Note that when a quoted string is included in a INPUT statement, the normal ? is not printed as a prompt character, and if desired, must be included as shown within the quotes above.

This feature allows BASIC 8001 to be programmed to handle fill-in-the-forms type of applications.

VIII. DATA STATEMENT

The DATA statement is used in conjunction with the READ statement to enter data into an executing program. One statement is never used without the other. The form of the statement is:

```
DATA value list
```

where the value list contains the numbers or strings to be assigned to the variables listed in a READ statement. Individual items in the value list are separated by commas; strings must be enclosed in quotation marks.

For example:

```
15Ø DATA 4,7.2,3,"ABC"
17Ø DATA 1,34E-3, 3.17311
```

The location of DATA statements is arbitrary as long as they appear in the correct order; however, it is good practice to collect all DATA statements near the end of the program.

When the RUN command is executed, BASIC 8001 searches for the first DATA statement and saves a pointer to its location. Each time a READ statement is encountered in the program, the next value in the data statement is assigned to the designated variable. If there are no more values in that DATA statement, BASIC 8001 looks for the next DATA statement.

IX. READ STATEMENT

A READ statement is used to assign the values listed in a DATA statement to the specified variables. The READ statement is of the form:

```
READ variable list
```

The items in the variable list may be simple variable names or string variable names and are separated by commas. For example:

```
1Ø READ A, B$, C(1)
2Ø DATA 12, "12",.12E2
```

Since data must be read before it can be used in a program, READ statements generally occur near the beginning of the program. A READ statement can be placed anywhere in a multiple statement line.

If there is no data available in the data table for the READ to store, the out of data message below is printed:

```
OD ERROR IN xxxxx
READY
```

Items in the data list in excess of those needed by the program's READ statements are ignored.

X. RESTORE STATEMENT

The RESTORE statement causes the program to reuse the data from the first DATA statement and is of the form:

```
RESTORE
```

For example:

```
30 RESTORE
```

causes the next READ statement following line 30 to begin reading data from the first DATA statement in the program, regardless of where the last value was found.

A further example of the use of RESTORE follows:

```
15 READ B,C,D
.
.
.
55 RESTORE
60 READ E,F,G
.
.
.
80 DATA 6,3,4,7,9,2
.
.
100 END
```

The READ statements in lines 15 and 60 both read the first three data values provided in line 80. (If the RESTORE statement had not been inserted before line 60, then the second READ would pick up data in line 80 starting with the fourth value.)

Since the values are being read as though for the first time, the same variable names may be used the second time through the data, if desired. To skip unwanted values, replacement, or dummy, variables may be inserted. For example:

```
1 REM - PROGRAM TO ILLUSTRATE USE OF RESTORE
20 READ N
25 PRINT "VALUES OF X ARE:"
30 FOR I=1 TO N
40 READ X
50 PRINT X,
60 NEXT I
70 RESTORE
185 PRINT
190 PRINT "SECOND LIST OF X VALUES"
200 PRINT "FOLLOWING RESTORE STATEMENT:"
210 FOR I=1 TO N
220 READ X
230 PRINT X,
240 NEXT I
```

```
250 DATA 4,1,2
251 DATA 3,4
300 END

RUN
VALUES OF X ARE:
  1        2        3        4
SECOND LIST OF X VALUES
FOLLOWING RESTORE STATEMENT:
  4        1        2        3
READY
```

The second time the data values are read, the first X picks up the value originally assigned to N in line 20, and as a result, BASIC prints:

```
4        1        2        3
```

To circumvent this, a dummy variable could be inserted to pick up and store the first value. This variable would not be represented in the PRINT statement, so the output would be the same each time through the list.

XI. GOTO STATEMENT

The GOTO statement is used when it is desired to unconditionally transfer to some line other than the next sequential line in the program. In other words, a GOTO statement causes an immediate jump to a specified line, out of the normal consecutive line number order of execution. The general format of the statement is as follows:

```
GOTO line number
```

The line number to which the program jumps can be either greater or less than the current line number. It is thus possible to jump forward or backward within a program.

For example,

```
10 LET A=2
20 GOTO 50
30 LET A=SQR(A+14)
50 PRINT A,A*A
RUN
```

causes the following to be printed:

```
2        4
```

When the program encounters line 20, control transfers to line 50; line 50 is executed, control then continues to the line following line 50. Line 30 is never executed. Any number of lines can be skipped in either direction.

When written as part of a multiple statement line, GOTO should always be the last statement on the line, since any statement following the GOTO on the same line is never executed. For example:

```
11Ø LET A=ATN(B2):PRINT A:GOTO 5Ø
```

XII. IF-THEN, IF-GOTO STATEMENTS

The IF-THEN statement is used to transfer conditionally from the normal consecutive order of statement numbers, depending upon the truth of some mathematical relation or relations. The basic format of the IF statement is as follows:

```
                                    THEN
IF expression rel.op. expression         line number
                                    GOTO
```

where expression is an arithmetic or string expression.

Expressions cannot be mixed; both must be string or both must be numeric. Numeric comparisons are handled as described in the ARITHMETIC Section. String comparisons are performed on the ASCII values of the strings as described in the STRING Section.

rel. op.	is one of the operators described in the ARITHMETIC Section.
line number	is the line of the program to which control is conditionally passed.

If the value of the expression is true, control passes to the line number specified.

If the value of the expression is false, control passes to the next statement in sequence.

Examples:

```
1Ø IF A=B THEN 2Ø:PRINT "A   B"
15 STOP
2Ø PRINT A+B

1Ø IF A<>1Ø GOTO 2Ø :PRINT A
15 STOP
2Ø D=A+B*C

1Ø IF A$<B$ THEN 2Ø:STOP
2Ø PRINT A$
```

XIII. FOR-NEXT STATEMENTS

FOR and NEXT statements define the beginning and end of a loop. (A loop is a set of instructions which are repeated over and over again, each time being modified in some way until a terminal condition is reached.) The FOR statement is of the form:

```
FOR variable = expression1 TO expression2 STEP expression3
```

where

variable	must be a nonsubscripted numeric variable.
expression	is an arithmetic expression which may be non-integer.

The variable is the index; expression1 is the initial value; expression2, the terminal value and expression3, the increment value.

For example:

```
15 FOR K=2 TO 2Ø STEP 2
```

causes the program execution of the designated loop as long as K is less than or equal to 20. Each time through the loop, K is incremented by 2, so the loop is executed a total of 10 times. When K=20, program control passes to the line following the associated NEXT statement.

The index variable must be unsubscripted, although a common use of such loops is to deal with subscripted variables using the control variable as the subscript of a previously defined variable. The expressions in the FOR statement can be any acceptable BASIC 8001 expression.

The NEXT statement signals the end of the loop which began with the FOR statement. The NEXT statement is of the form:

```
NEXT variable
```

where the variable is the same variable specified in the FOR statement. Together the FOR and NEXT statements define the boundaries of the program loop. When execution encounters the NEXT statement, the computer adds the STEP expression value to the variable and checks to see if the variable is still less than or equal to the terminal expression value. When the variable exceeds the terminal expression value, control falls through the loop to the statement following the NEXT statement. Note the variable is not necessary since when a NEXT statement is encountered it is assumed it is for the appropriate FOR loop variable.

If the STEP expression and the word STEP are omitted from the FOR statement, +1 is the assumed value. Since +1 is a common STEP value, that portion of the statement is frequently omitted.

The expressions within the FOR statement are evaluated once upon initial entry to the loop. The test for completion of the loop is made after each execution of the loop. (If the test fails initially, the loop is still executed once.)

The index variable can be modified within the loop. When control falls through the loop, the index variable retains the value used to fall through the loop.

The following is a demonstration of a simple FOR-NEXT loop. The loop is executed 10 times; the value of I is 11 when control leaves the loop; and +1 is the assumed STEP value:

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 PRINT I
```

The loop itself is lines 10 through 30. The numbers 1 through 10 are printed when the loop is executed. After I=10, control passes to line 40 which causes 11 to be printed. If line 10 had been:

```
10 FOR I = 10 TO 1 STEP -1
```

the value printed by line 40 would be Ø.

```
10 FOR I = 2 TO 44 STEP 2
20 LET I = 44
30 NEXT I
```

The above loop is only executed once since the value of I=44 has been reached and the termination condition is satisfied.

If the initial value of the variable is greater than the terminal value, the loop is still executed once. The loop set up by the statement:

```
10 FOR I = 20 TO 2 STEP 2
```

will be executed only once although a statement like the following will initialize execution of a loop properly:

```
10 FOR I=20 TO 2 STEP -2
```

For positive STEP values the loop is executed until the control variable is greater than its final value. For negative STEP values, the loop continues until the control variable is less than its final value.

FOR loops can be nested but not overlapped. The depth of nesting depends upon the amount of user storage space available (in other words, upon the size of the user program and the amount of RAM available). Nesting is a programming technique in which one or more loops are completely within another loop. The field of one loop (the numbered lines from the FOR statement to the corresponding NEXT statement, inclusive) must not cross the field of another loop.

ACCEPTABLE NESTING TECHNIQUES

Two Level Nesting

```
FOR I1 = 1 TO 10
FOR I2 = 1 TO 10
NEXT I2
FOR I3 = 1 TO 10
NEXT I3
NEXT I1
```

Three Level Nesting

```
FOR I1 = 1 TO 10
FOR I2 = 1 TO 10
FOR I3 = 1 TO 10
NEXT I3
FOR I4 = 1 TO 10
NEXT I4
NEXT I2
NEXT I1
```

UNACCEPTABLE NESTING TECHNIQUES

```
FOR I1 = 1 TO 10
FOR I2 = 1 TO 10
NEXT I1
NEXT I2
```

```
FOR I1 = 1 TO 10
FOR I2 = 1 TO 10
FOR I3 = 1 TO 10
NEXT I3
FOR I4 = 1 TO 10
NEXT I4
NEXT I1
NEXT I2
```

An example of nested FOR-NEXT loops is shown below:

```
5 DIM X(5,1Ø)
1Ø FOR A=1 TO 5
2Ø FOR B=2 TO 1Ø STEP 2
3Ø LET X(A,B)= A+B
4Ø NEXT B
5Ø NEXT A
55 PRINT X(5,1Ø)
```

When the above statements are executed, BASIC 8001 prints 15 when line 55 is processed.

It is possible to exit from a FOR-NEXT loop without the control variable reaching the termination value. A conditional or unconditional transfer can be used to leave a loop. Control can only transfer into a loop which had been left earlier without being completed, ensuring that termination and STEP values are assigned.

Both FOR and NEXT statements can appear anywhere in a multiple statement line. For example:

```
1Ø FOR I=1 TO 1Ø STEP 5:NEXT I: PRINT "I=";I
```

causes:

```
I=11
```

to be printed when executed.

XIV. GOSUB AND RETURN STATEMENTS

A subroutine is a section of code performing some operation required at more than one point in the program. Sometimes a complicated I/O operation for a volume of data, a mathematical evaluation which is too complex for a user-defined function, or any number of other processes may be best performed in a subroutine.

More than one subroutine can be used in a single program, in which case they can be placed one after another at the end of the program (in line number sequence). A useful practice is to assign distinctive line numbers to subroutines; for example, if the main program uses line numbers up to 199, use 200 and 300 as the first numbers of two subroutines.

Subroutines are usually placed physically at the end of a program before DATA statements, if any. The program begins execution and continues until it encounters a GOSUB statement of the form:

1) GOSUB line number

where the line number following the word GOSUB is that of the first line of the subroutine. Control then transfers to that line of the subroutine. For example:

```
50 GOSUB 200
```

Control is transferred to line 200 in the user program. The first line in the subroutine can be a remark or any executable statement.

Having reached the line containing a GOSUB statement, control transfers to the line indicated after GOSUB; the subroutine is processed until BASIC 8001 encounters a RETURN statement of the form:

2) RETURN

which causes control to return to the statement following the original GOSUB statement. A subroutine must always be exited via a RETURN statement.

Before transferring to the subroutine, BASIC 8001 internally records the next sequential statement to be processed after the GOSUB statement; the RETURN statement is a signal to transfer control to this statement. In this way, no matter how many subroutines there are or how many times they are called, BASIC 8001 always knows where to transfer control next. The following program demonstrates the use of GOSUB and RETURN.

```
1      REM - THIS PROGRAM ILLUSTRATES GOSUB AND RETURN
10     DEF FNA(X)= ABS(INT(X))
20     INPUT A,B,C
30     GOSUB 100
40     LET A=FNA(A)
```

```
5Ø    LET B=FNA(B)
6Ø    LET C=FNA(C)
7Ø    PRINT
8Ø    GOSUB 1ØØ
9Ø    STOP
1ØØ   REM - THIS SUBROUTINE PRINTS OUT THE SOLUTIONS
11Ø   REM - OF THE EQUATION:  AX^2 + BX + C = Ø
12Ø   PRINT "THE EQUATION IS  "A "*X^2 + " B"*X + "C
13Ø   LET D=B*B - 4*A*C
14Ø   IF D<>0 THEN 17Ø
15Ø   PRINT "ONLY ONE SOLUTION... X "; -B/(2*A)
16Ø   RETURN
17Ø   IF D<Ø THEN 2ØØ
18Ø   PRINT "TWO SOLUTIONS...X =";
185   PRINT (-B+SQR(D))/(2*A); ") AND ("; (-B-SQR(D))/(2*A)
19Ø   RETURN
2ØØ   PRINT "IMAGINARY SOLUTIONS ...X=(";
2Ø5   PRINT -B/(2*A) "," SQR(-D)/(2*A) ") AND (";
2Ø7   PRINT -B/(2*A) ","; -SQR(-D)/(2*A) ")"
21Ø   RETURN
9ØØ   END
```

Subroutines can be nested; that is, one subroutine can call another subroutine. If the execution of a subroutine encounters a RETURN statement, it returns control to the line following the GOSUB which called that subroutine. Therefore, a subroutine can call another subroutine, even itself. Subroutines can be entered at any point and can have more than one RETURN statement. It is possible to transfer to the beginning or any part of a subroutine; multiple entry points and RETURN's make a subroutine more versatile. Up to 20 levels of GOSUB nesting are allowed.

XV. END STATEMENT

The END statement is the last statement in a BASIC program and is of the form:

```
END
```

The line number of the END statement must be the largest line number in a given program, since any lines having line numbers greater than that of the END statement are not executed (although they are saved with the SAVE command).

The END statement is optional. When an END statement is executed, program execution stops and the READY message is printed.

XVI. STOP STATEMENT

The STOP statement can occur several times throughout a single program with conditional jumps determining the actual end of the program. The STOP statement is of the form:

```
90 STOP
```

and causes:

```
BREAK IN 90
READY
```

to be printed when executed.

This signals that the execution of a program has been terminated and BASIC 8001 is able to accept further input.

BASIC 8001 FUNCTIONS

ARITHMETIC FUNCTIONS

BASIC 8001 provides functions to perform certain standard mathematical operations such as square roots, logarithms, etc.

These functions have three or four letter call names followed by a parenthesized argument. They are pre-defined and may be used anywhere in a program.

Call Name	Function
ABS(x)	Returns the absolute value of x.
ATN(x)	Returns the arctangent of x as an angle in radians in range + or -pi/2.
CALL(x)	CALL the user, machine language program at location ØAØØØ Hex. AØØØ = JMP = -24576 AØØ1 = Lo = -24575 AØØ2 = Hi = -24574
COS(x)	Returns the cosine of x radians.
EXP(x)	Returns the value of e^x where e=2.71828.
FRE(x)	Returns number of free BYTES not in use.
INT(x)	Returns the greatest integer less than or equal to x, (INT(-.5)=-1).
INP(x)	Returns a BYTE from input port $0 \leq x \leq 255$.
LOG(x)	Returns the natural logarithm of x.
PEEK(x)	Returns a BYTE from memory address $0 \leq x \leq 32767$ or if x is negative the memory address is 65536+x.
POS(x)	Returns a value of current cursor positions between 0 and 79.
RND(x) NOT TOO GOOD...REPEATS after only 1995 numbers.	Returns a random number between 0 and 1.
SGN(x)	Returns a value indicating the sign of x.
SIN(x)	Returns the sine of x radians.
SPC(x)	Causes x spaces to be generated.
SQR(x)	Returns the square root of x.
TAB(x)	Causes the 8001 CRT to space over to column number x. Valid in PRINT statement only.
TAN(x)	Returns the tangent of x radians.

The argument x to the functions can be a constant, a variable, an expression, or another function. A square bracket cannot be used as the enclosing character for the argument x, e.g. SIN [x] is illegal.

Function calls, consisting of the function name followed by a parenthesized argument, can be used as expressions or as elements of expressions anywhere that expressions are legal.

Values produced by the functions SIN(x), COS(x), ATN(x), SQR(x), EXP(and LOG(x) have six significant digits.

I. Sine and Cosine Functions, SIN(x) and COS(x)

The sine and cosine functions require an argument angle expressed in radian measure. If the angle is stated in degrees, conversion to radians may be done using the identity:

<radians> = <degrees>*(pi/180)

In the following example program, 3.14159 is used as a nominal value for pi. P is set equal to this value at line 20. At line 40 the above relationship is used (in the expression within the LET statemer to convert the input value into radians.

```
1Ø REM - CONVERT ANGLE (X) TO RADIANS, AND
11 REM - FIND SIN AND COS
2Ø LET P = 3.14159
25 PRINT "DEGREES", "RADIANS", "SINE", "COSINE"
3Ø INPUT X
4Ø LET Y = X*P/180
6Ø PRINT X, Y, SIN(Y), COS(Y)
7Ø GOTO 30
RUN
DEGREES          RADIANS          SINE             COSINE
?Ø
 Ø               Ø                Ø                1
?1Ø
 1Ø               .174533          .173648          .9848Ø8
?2Ø
 2Ø               .349Ø66          .342Ø2           .939693
?3Ø
 3Ø               .523598          .5               .866Ø26
?36Ø
 36Ø             6.28318          -5.2431ØE-Ø6     1
?45
 45               .785398          .7Ø71Ø6          .7Ø71Ø7
?9Ø
 9Ø              1.57Ø8            1               1.12352E-Ø6
?RETURN
READY
```

II. Arctangent Function, ATN(x); Tangent Function, TAN(x)

The arctangent function returns a value in radian measure, in the range +pi/2 to -pi/2 corresponding to the value of a tangent supplied as the argument (X).

In the following program, input is an angle in degrees. Degrees are then converted to radians at line 40.

At line 70 the tangent value, Z, is supplied as argument to the ATN function to derive the value found in column 4 of the printout under the label ATN(X). Also in line 70 the radian value of the arctangent function is converted back to degrees and printed in the fifth column of the printout as a check against the input value shown in the first column.

```
1Ø LET P= 3.14159
2Ø PRINT "SUPPLY AN ANGLE IN DEGREES"
25 PRINT "ANGLE", "ANGLE", "TAN(X)", "ATAN(X)", "ATAN(X)"
26 PRINT "(DEGS)","(RADS)",,,"(DEGS)"
3Ø INPUT X
4Ø LET Y = X*P/18Ø
5Ø LET Z = TAN(Y)
7Ø PRINT X,Y,Z,ATN(Z),ATN(Z)*180/P
85 PRINT
9Ø GOTO 3Ø
RUN
SUPPLY AN ANGLE IN DEGREES
ANGLE   ANGLE     TAN(X)      ATAN(X)       ATAN(X)
(DEGS)  (RADS)                              (DEGS)
?Ø
 Ø       Ø         Ø           Ø             Ø

?45
 45     .785398   .999999     .785398       45

?1Ø
 10     .174533   .176327     .174533       1Ø
?(RETURN)
READY
```

III. Square Root Function, SQR(x)

This function derives the square root of any positive value as shown below.

```
1Ø INPUT X
2Ø LET X = SQR(X)
3Ø PRINT X
4Ø GOTO 1Ø
RUN
?16
 4
?1ØØ
 1Ø
?1ØØØ
```

```
 31.6228
?123456789
 11111.1
?17
 4.12311
?25E2
 5Ø
?197Ø
 44.3847
?(RETURN)
READY
```

IV. Exponential Function, EXP(x)

The exponential function raises the number e to the power **x.** EXP is the inverse of the LOG function. The relationship is

```
LOG(EXP(X)) = X
```

The following program prints the exponential equivalent of an input value. Note that the output values derived below are used as input to the LOG function.

```
1Ø INPUT X
2Ø PRINT EXP(X)
4Ø GOTO 1Ø

RUN
?4
 54.5981
?1Ø
 22Ø26.5
?9.421ØØ6
 12345
?4.6Ø517
 1ØØ
?25
 7.2ØØ49E+1Ø
?(RETURN)
READY
```

V. Logarithm Function, LOG(x)

The LOG function derives the logarithm to the base e of a given value. In the following program at line 20, the LOG function is used to convert an input value to its logarithmic equivalent.

```
1Ø INPUT X
2Ø PRINT LOG(X)
3Ø GOTO 1Ø

RUN
?54.59815
 4
?22Ø26.47
 1Ø
```

```
?12345
 9.42101
?100
 4.60517
?.720049E11
 25
?(RETURN)
READY
```

Logarithms to the base e may easily be converted to any other base using the following formula:

$$\log_a N = \frac{\log_e N}{\log_e a}$$

where a represents the desired base. The following program illustrates conversion to the base 10.

```
 1 REM - CONVERT BASE E LOG TO BASE 10 LOG.
 5 PRINT "VALUE","BASE E LOG","BASE 10 LOG"
15 INPUT X
17 PRINT X,
20 PRINT LOG(X),
40 PRINT LOG(X)/LOG(10)
50 GOTO 15
60 END
RUN
VALUE              BASE E LOG           BASE 10 LOG
?4
 4                 1.38629               .60206
?250
 250               5.52146              2.39794
?5
 5                 1.60944               .69897
?60
 60                4.09434              1.77815
?100
 100               4.60517              2
?(RETURN)
READY
```

An attempt to do a LOG(0) or LOG of a negative number causes the CF error message.

VI. Absolute Function, ABS(x)

The ABS function returns an absolute value for any argument value. Absolute value is always positive. In the following program, various input values are converted to their absolute values and printed.

```
1Ø INPUT X
2Ø LET X = ABS(X)
3Ø PRINT X
4Ø GOTO 1Ø
RUN
?-35.7
 35.7
?2
 2
?25E1Ø
 2.5ØØØØE+11
?1Ø5555567
 1.Ø5556E+Ø8
?1Ø.12345
 10.1234
?-44.555566668899
 44.5556
?(RETURN)
READY
```

VII. Integer Function, INT(x)

The integer function returns the value of the greatest integer not greater than x. For example:

```
PRINT INT(34.67)
34

PRINT INT(-5.1)
-6
```

The INT of a negative number is a negative number with the same or larger absolute value, i.e., the same or smaller algebraic value. For example:

```
PRINT INT(-23.45)
-24

PRINT INT(-14.39)
-15

PRINT INT(-11)
-11
```

The INT function can be used to round numbers to the nearest integer, using INT(X+.5). For example:

```
PRINT INT(34.67+.5)
  35

PRINT INT(-5.1+.5)
 -5
```

INT(x) can also be used to round to any given decimal place or integral power of 10, by using the following expression as an argument:

```
(X*10↑D+.5)/10 D
```

where D is an integer supplied by the user.

```
1Ø REM - INT FUNCTION EXAMPLE
15 PRINT
2Ø PRINT "NUMBER TO BE ROUNDED:"
25 INPUT A
4Ø PRINT "NO. OF DECIMAL PLACES:"
45 INPUT D
6Ø LET B = INT(A*10^D + .5)/1Ø^D
7Ø PRINT "A ROUNDED = " B
8Ø GOTO 15
9Ø END
RUN

NUMBER TO BE ROUNDED:
?55.65842
NO. OF DECIMAL PLACES:
?2
A ROUNDED = 55.66

NUMBER TO BE ROUNDED:
?78.375
NO. OF DECIMAL PLACES:
?-2
A ROUNDED = 1ØØ

NUMBER TO BE ROUNDED:
?67.38
NO. OF DECIMAL PLACES:
?-1
A ROUNDED = 7Ø

NUMBER TO BE ROUNDED:
?(RETURN)
READY
```

VIII. Random Number Function, RND(x) Repeats after 1995 numbers.

The random number function produces a random number, or random number set, between 0 and 1. The numbers are reproducible in the same order after ESC, E key if X>0 for later checking of a program. The argument (x) is not used and can be any number (it cannot be a string expression); it serves only to standardize all BASIC 8001 function representations. The form RND is not legal. For example:

```
10 REM - RANDOM NUMBER EXAMPLE.
25 PRINT "RANDOM NUMBERS:
30 FOR I = 1 TO 15
40 PRINT RND(1);
50 NEXT I
60 END
RUN
RANDOM NUMBERS:
 .100250.50438   .964813.0267824  .886627.388094  .636444.569123  .839019.720021
 .306121.209046  .285553.599886   .958221.744055  .179351.460434  .452117.433291
 .985412.27376   .522186.701146   .246246.590584  .777801.457448  .450592.30797

READY
```

To obtain random digits from 0 to 9, change line 40 to read:

```
40 PRINT INT(10*RND(1)),
```

and run the program again. This time the results will be printed as follows:

```
RUN
RANDOM NUMBERS:
8        9        8        9        5        5        5        9        8       7
5        4        4        1        5

READY
```

It is possible to generate random numbers over a given range. If the open range (A,B) is desired, use the expression:

```
(B-A)*RND(1)+A
```

to produce a random number in the range A<n<B.

The following program produces a random number set in the open range 4,6 (the extremes, 4 and 6, are never reached).

```
10 REM - RANDOM NUMBER SET IN OPEN RANGE 4,6.
20 FOR B = 1 TO 15
30 LET A = (6-4) * RND(1) +4
40 PRINT A,
50 NEXT B
60 END

RUN
 4.20054.59266  5.929624.20985  5.773255.54026  5.272884.76248  5.678045.25946
 4.612245.33046 4.571104.26695  5.916445.69965  4.358705.54721  4.904235.65021
 4.197085.09034 5.044374.82533  4.492495.61408  5.555604.41632  4.901185.01508

READY
```

NOTE: Negative arguments, i.e., RND(-x) will start a new random number sequence. While RND (Ø) will always generate the last random number. RND(-X) will always restart the same random number sequence!

IX. Sign Function, SGN(x)

The sign function returns the value 1 if x is a positive value, Ø if x is 0 and -1 if x is negative. For example:

```
PRINT SGN(3.42)
 1

PRINT SGN(-42)
-1

PRINT SGN(23-23)
 Ø
```

The following example program illustrates the use of the SGN function.

```
1Ø REM-SGN FUNCTION EXAMPLE.
2Ø READ A,B,C
25 PRINT "A = "A,"B = "B,"C = "C
3Ø PRINT "SGN(A) ="SGN(A), "SGN(B) ="SGN(B),
4Ø PRINT "SGN(C) ="SGN(C)
5Ø DATA -7.32, .44, Ø
6Ø END
RUN
A = -7.32   B = .44   C = Ø
SGN(A) =-1   SGN(B) =1   SGN(C) =Ø

READY
```

X. Call Statement

The CALL statement can be inserted anywhere in the BASIC 8001 program and has the form:

CALL (expression)

Where	expression	is the argument to the assembly language routine. The argument may be an expression. This may include values passed to the user routine.

The CALL statement causes a jump to location A000 HEX, which, unless modified by the user, contains a jump to the CF ERROR routine. The user must modify these three locations to go to his routines.

BASIC 8001 FUNCTIONS

USER DEFINED FUNCTIONS

In some programs it may be necessary to execute the same sequence of statements or mathematical formulas in several different places. BASIC 8001 allows definition of unique operations or expressions and the calling of these functions in the same way as the square root or trig functions.

These user-defined functions consist of a function name: the first two letters of which are FN followed by a third or a fourth letter. For example:

legal	illegal
FNA	FNA$
FNAA	FN2
FNA1	

Each function is defined once and the definition may appear anywhere in the program. The defining or DEF statement is formed as follows:

```
DEF FNa (argument) = expression (argument)
```

where a is a variable name. The argument may consist of a dummy variable and the number of arguments is limited to one variable. The expression may contain other program variables not among the argument variable. For example:

```
10 DEF FNA(S) = S^2
```

causes a later statement:

```
2Ø LET R = FNA(4)+1
```

to be evaluated as R=17. As another example:

```
5Ø DEF FNB(A) = A+X^2
6Ø Y=FNB(14)
```

causes the function to be evaluated with the current value of the variable X within the program.

The two following programs

Program #1:

```
1Ø DEF FNS(A) = A^A
2Ø FOR I=1 TO 5
3Ø PRINT I, FNS(I)
4Ø NEXT I
5Ø END
```

Program #2:

```
10 DEF FNS(X) = X^X
20 FOR I=1 TO 5
30 PRINT I, FNS(I)
40 NEXT I
50 END
```

cause the same output:

```
RUN
 1          1
 2          4
 3          27
 4          256
 5          3125

READY
```

The argument in the DEF statement can be seen to have no significance; it is strictly a dummy variable. (A DEF statement with no arguments is illegal.) The function itself can be defined in the DEF statement in terms of numbers, variables, other functions, or mathematical expressions. For example:

```
10 DEF FNA(X) = X^2+3*X+4
20 DEF FNB(X) = FNA(X)/2 + FNA(X)
30 DEF FNC(X) = SQR(X+4)+1
```

The statement in which the user-defined function appears can have that function combined with numbers, variables, other functions, or mathematical expressions. For example:

```
40 LET R = FNA(X+Y+Z)*N/(Y^2+D)
```

A user-defined function cannot have several arguments, as shown below:

```
25 DEF FNL(X,Y,Z) = SQR(X^2 + Y^2 + Z^2)
```

will cause an error

```
SN ERROR IN 25.
READY
```

When calling a user-defined function, the parenthesized arguments can be any legal expressions. The value of each expression is substituted for the corresponding function variable. For example:

```
10 DEF FNZ(X)=X^2
20 LET A=2
30 PRINT FNZ(2+A)
```

line 30 causes 16 to be printed.

If the same function name is defined more than once, then the last definition will be used. The program below

```
10 DEF FNX(X)=X^2
20 DEF FNX(X)=X+X
30 LET A=5
40 PRINT FNX(A)
```

will cause 10 to be printed.

The function variable need not appear in the function expression as shown below:

```
10 DEF FNA (X) = 4 +2
20 LET R = FNA(10)+1
30 PRINT R
40 END
RUN
 7
```

BASIC 8001 FUNCTIONS

STRING FUNCTIONS

Like the intrinsic mathematical functions (e.g., SIN, LOG), BASIC 8001 contains various functions for use with character strings. These functions allow the program to concatenate two strings, access part of a string, determine the number of characters in a string, generate a character string corresponding to a given number or vice versa, search for a substring within a larger string, and perform other useful operations. The various functions available are summarized in the following table.

String Functions

Function code	Meaning
ASC(x$)	Returns the seven-bit internal code for the one-character string (x$) as a decimal number. If the argument contains more than one character, then the first character in the string is returned.
CHR$(x)	Generates a one-character string having the ASCII value of x where x is a number greater than or equal to 0 and less than or equal to 255. For example: CHR$(65) is equivalent to "A". Only one character can be generated.
FRE(x$)	Returns number of free string BY[illegible].
LEFT$(x$,I)	Returns left most I characters of string (x$).
LEN(x$)	Returns the number of characters in the string x$ (including trailing blanks). For example: PRINT LEN(A$) 26
MID$(x$,I,J)	Returns the string of characters in position I through J in x$.
RIGHT$(x$,I)	Returns right most I characters of string (x$).
STR$(x)	Returns the string which represents the numeric value of x as it would be printed by a PRINT statement but without a leading or trailing blank.

VAL(x$)	Returns the number represented by the string x$. If x$ does not represent a number, then Ø value is returned.

In the above examples, x$ and y$ represent any legal string expressions, and I and J represent any legal arithmetic expressions.

User-Defined String Functions

Character string functions cannot be written in the same way as numeric functions.

BASIC 8001 EDITING COMMANDS

BASIC 8001 provides several key commands which can be used to halt program execution, erase characters or delete lines. The below table provides an explanation of each of the key commands.

Key Commands

Key	Explanation
CTRL/J or LINEFEED or ↓	Interrupts execution of a command or program. BASIC 8001 prints the message BREAK IN XXX READY A control command is typed by holding down the CTRL key while typing the letter key.
CTRL/M or RETURN	Must be typed to end every line typed in or to indicate the end of an INPUT.
CTRL/K or ERASE LINE	Deletes the entire current line (provided the RETURN key has not been typed). BASIC 8001 displays: Erased line and CR.
CTRL/Z or CURSOR LEFT or ←	Deletes the last character typed and echoes as a cursor left on the terminal. Spaces as well as characters or control codes may be erased.
:	A colon is used to separate multiple statements per line.
CTRL/L or ERASE PAGE	Erases CRT screen but does not change any BASIC 8001 statements.

If the RETURN key has already been typed, a program line can be corrected by typing the appropriate line number and retyping the line correctly.

The line can be deleted by typing the RETURN key immediately after the line number; removing both the line number and line from the program.

If the line number of a line not needing correction is accidentally typed, the cursor left key (CTRL Z) may be used to delete the number(s); then the correct number can be typed. Assume the line:

```
10 IF A>5 GO TO 230
```

is correct. A line 15 is to be inserted, but:

```
10 LET
```

is typed by mistake. The correction is made as follows:

```
10 LET←←←←5 LET X=X-3
```

Line 10 remains unchanged, and line 15 is entered.

Following an attempt to run a program, error messages may be output on the terminal indicating illegal characters or formats, or other user errors in the program. Most errors can be corrected by typing the line number(s) and the correction(s) and then rerunning the program. As many changes or corrections as desired may be made before runs.

The following editing commands are entered in immediate mode and terminated by the RETURN key. These commands are used to erase a program in RAM, and list, punch or run a program.

I. NEW COMMAND

The NEW command clears current contents of the storage area set up by BASIC 8001. This deletes any commands, programs, arrays, strings or symbols currently stored by BASIC 8001.

NEW should be used before entering a new program from the terminal keyboard to be sure no old program lines will be mixed into the new program and to clear out the symbol table area.

Example:

```
NEW
READY
10 READ A
.
.
.
```

clears the storage area and inserts the program being input at the keyboard.

II. LIST COMMAND

The LIST command prints the user program currently in core on the terminal.

A part of a program may be listed by typing LIST followed by a line number. This causes that line and all following lines in the program to be listed.

Type CTRL/J or linefeed key to halt the listing. BASIC 8001 returns to the READY message when the current line is finished.

The lines listed may differ slightly from those entered because:

1. Certain characters while acceptable to BASIC 8001 are stored in a standard manner.

Character Typed	Character Stored
=<	<=
=>	>=
><	<>

2. Literals are stored to 24 bits of accuracy. Those with more than 24 bits are truncated to 24 bits.

3. Although literal storage is 24 bits, output is truncated to 6 decimal digits.

4. Literals are output in standard BASIC 8001 format, regardless of how they were input; for example,

```
1Ø LET X=3.Ø+1.ØØØØØØ1
2Ø PRINT X-E7
LIST
1Ø LET X=3+1
2Ø PRINT X-1.ØØØØØE+Ø7
```

5. Spaces in the input program are ignored, except within strings and REM statements. The LIST command prints the program with a space inserted to separate the key word and the line number. The listed program is therefore easier to read.

Example:

```
LIST 1ØØ
```

Lists line 100 and all remaining lines in the program.

III. SAVE COMMAND

The SAVE command outputs the program in RAM to the specified device. The form of the command is:

```
SAVE A
```

The format of the program output by the SAVE command is exactly the same as that stored in RAM memory. It may be recalled by the same file name using the LOAD command.

IV. RUN COMMAND

After the user program is entered into RAM, it can be executed by typing the command

```
RUN
```

and the RETURN key.

The program is scanned; arrays are created in core and then the program is executed. Any appropriate error messages are printed and when the END or STOP statement is encountered, execution halts and a message is printed.

After execution, the variables used in a program remain accessible for use in immediate mode until a NEW, CLEAR or another RUN command is executed.

V. CLEAR COMMAND

The CLEAR command clears the contents of the user array and string buffers. This command is generally used when a program has been executed and then edited. Before it is rerun, the array and string buffers are set to zeros and nulls by the CLEAR command to provide more core.

These buffers will be filled again when the RUN command is executed.

Example:

```
10 A=10
20 PRINT A
CLEAR

READY

RUN
10

READY
```

VI. CLEAR X COMMAND

The CLEAR X performs the same function as CLEAR without the argument, but the Argument X reserves X locations for string variables which are required in string calculations. Normally this is 50 locations unless changed by CLEAR X command.

VII. CONTINUE COMMAND

Continues program execution after a Control J or line feed is typed or a STOP statement is executed. You cannot continue after any error, after modifying your program or before your program has been run.

One of the main purposes of CONT is debugging. Suppose at some point after running your program, nothing is printed. This may be because your program is performing some time-consuming calculation, but it may be because you have fallen into an "infinite loop". An infinite loop is a series of BASIC 8001 statements from which there is no escape. The BASIC 8001 will keep executing a series of statements over and over until you intervene or until power to the unit is cut off. If you suspect your program is in an infinite loop, type in a Control J or line feed. The line number of the statement BASIC 8001 was executing will be typed out.

After BASIC 8001 has typed out READY, you can use PRINT to type out some of the values of your variables. After examining these values, you may become satisfied that your program is functioning correctly. You should then type in CONT to continue executing your program where it left off, or type a direct GOTO statement to resume execution of the program at a different line.

You could also use assignment (LET) statements to set some of your variables to different values. Remember, if you line feed or Control J your program and expect to continue it later, you must not get any errors or type in any program lines. If you do, you won't be able to continue, and get a "CN" (continue not) error. It is impossible to continue a direct command. CONT always resumes execution at the next statement to be executed in your program when Control J or line feed was typed.

VIII. LOAD I COMMAND

LOADS the program named I from the 8001 CPU operating system Reader Input port specified by the I/O BYTE at location 9F90 HEX, see the CPU O.S. Manual. A new command is automatically done before the LOAD I command is executed. When finished loading the READY command will appear as usual. If the unit can't find the file on the floppy tape, then an error message should appear.

IX. LOAD?I COMMAND

Does same as LOAD I except that a NEW command is not performed and BASIC 8001 does a word-by-word comparison of file I with the program already existing in RAM memory. If they are the same, then READY appears, else

```
VERIFY FAILURE
READY
```

will appear.

This should always be used after saving a program with the SAVE I command to ensure that it was saved correctly and can be reloaded without error.

USING ASSEMBLY LANGUAGE

ROUTINES WITH BASIC

BASIC 8001 has a facility which allows experienced 8080 assembly language programmers to interface their own assembly language routines to BASIC 8001. This facility permits the user to add functions to BASIC 8001 which can operate directly on special purpose peripheral devices. This section describes in some detail the internal characteristics of BASIC 8001 during the execution of a BASIC 8001 program, and is intended to serve as a programming guide for the creation of such user-coded assembly language functions. This material assumes the user is familiar with 8080 assembly language. For additional information on this subject, refer to an assembly language programming manual on the 8080 CPU.

The CALL statement is used to reference these assembly language routines from the BASIC 8001 program.

Example: To call Assembly Language program from BASIC and pass arguements.

BASIC Program: (Multiply X by 2, by shifting to the left) [No speed advantage, by test.]

```
10  INPUT X
20  A=CALL(X)
30  PRINT X,A        A=2X
40  GOTO 10
5  POKE -24575,0: POKE -24574,176
```

NOTE. CALLING AT LOCATION $25A2_{HEX}$ will get the value of X into the D,E register.

ASSEMBLY LANGUAGE PROGRAM:

```
CALL 25A2H   *Get X in D,E Register
MOV  A,E
RLC
MOV  B,A
XRA  A       *To return value must be in A,B register.
JMP  2C53H   *Then jump to location 2C53HEX.
```

(COLENOTE: then why clear A with XRA A? So not get 256*X added in?)

NOTE: This will return the contents of the A,B register to the variable "A" in line 20, and also return to BASIC 8001.

When Assembly Program is entered at a location, say $B000_{HEX}$, then must place a jump to $B000_{HEX}$ at location $A000_{HEX}$ as BASIC 8001 will jump to A000 whenever it encounters a call to Assembly Language (This jump MUST be POKE IN via BASIC)

ColeNote: these are informal "hints" from the COMPUCOLOR people, and they are not too accurate...take it for what it is worth. I corrected some obvious errors, but not all. Cannot yet verify their complete data structure. (D,E & A,B is incomplete)

ColeNote: BASIC 8001 initializes A000 through A0B3...BEWARE!

ColeNote: POKE 32687,X through POKE 32767,X is stack area...good scratch!

Additional comments:

<u>My test program:</u>

```
1Ø  POKE -24575,Ø:  POKE -24574,176
2Ø  INPUT X
3Ø  FOR I=1 TO 1ØØØ: A=CALL(X): NEXT: PRINT X;A
4Ø  FOR I=1 TO 1ØØØ: A=2*(X): NEXT: PRINT X;A
5Ø  GOTO 2Ø
99  END
```

Some programs

BØØØ	CALL $25A2_{HEX}$	CD A2 25
BØØ3	MOV B, E	43
BØØ4	XRA, A	AF
BØØ5	JMP $2C53_{HEX}$	C3 53 2C

BØØØ	CALL $25A2_{HEX}$
BØØ3	MOV A, E

The Compucolor 8001 CRT

T A B L E O F C O N T E N T S

PROPRIETARY STATEMENT

PART I

SPECIFICATIONS

Introduction

The Intecolor® 8001 is an eight color intelligent CRT data terminal designed as a replacement for teletypes and black and white CRT data terminals. It is a self-contained, desk top unit which offers, with the use of a modem, two-way data communications over common voice telephone lines or teletype compatible current loops. It can also be used in the stand alone mode as a complete desk top computer if equipped with the proper options.

Basic System Specification

Power:	105-125 volts, 60HZ, 250 watts Option 11: 205-250 volts, 50-60 HZ
Temperature:	+10°C to +40°C operating -30°C to +70°C storage
Humidity:	0 to 95% non-condensing
Package Size Desk Mount Version:	17 1/2" high 19 3/8" wide 22 1/2" long
Keyboard Dimensions:	3 1/4" high 14 1/16" wide x 5 1/2" deep
Weight:	85 pounds
Screen Size:	19" diagonal measure 186 sq. inch screen area 4x3 aspect ratio
Display Area:	120 sq. inches (12.0" wide x 10.0" high)
Character Format:	80 characters per line, 25 lines per page Option 16: 80 characters per line, 48 lines per page
Character Style:	64 ASCII Characters, 5x7 dot matrix within a 6x8 dot pattern Option 03: 32 Graphic Characters, 6x8 dot matrix Option 17: 64 Graphic Characters, 6x8 dot matrix

Standard Interface

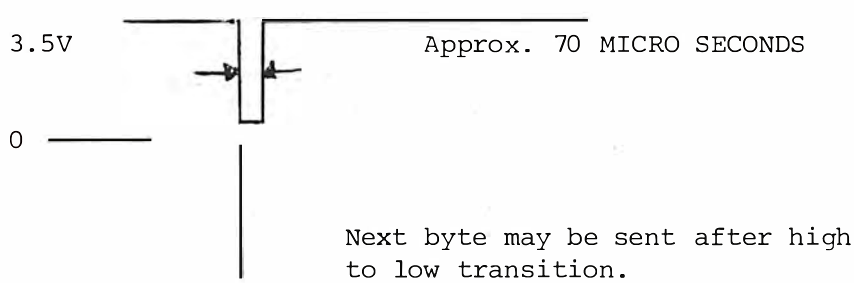
Standard I/O Ports

The standard Intecolor 8001 has two input ports. One port, J1, is an asynchronous serial RS 232C I/O, or if Option 07 is installed, a serial 20 ma current loop I/O. The other port, J2, accepts parallel input data from the keyboard and provides an 8 bit parallel output. The Intecolor® 8001 is furnished with a crystal clock and provides a keyboard selectable baud rate of normal 110, 150, 300, 1200, 2400, 4800, and 9600 baud, or a high speed option of 880, 1200, 2400, 9600, 19,200, 38,400, and 76,800 baud.

The serial input port is furnished without parity checking so that when in the Plot Mode, or CCI Mode, eight data bits can be received.

The signals for the standard RS 232C I/O ports are shown on page 3 and on J1 and J2 in Appendix A7.

Pin 2 of the Keyboard J2 connector signals the Data communications equipment that the terminal has received a byte and is processing the last byte received. The Unit's input port has a one byte buffer. So for maximum speed, the communications equipment can send the next byte as soon as it has detected the high to low transition on pin 2. The wave form is shown below:



Pin #	Signal Line	Nomenclature	Direction	Comments
1	Protective Ground	AA	NA	Connect to Chassis Ground and Pin 7 also
2	Transmitted Data	BA	From ISC to DCE*	"1" = Mark= -V "0" = Space= +V
3	Received Data	BB	From DCE* to ISC	"1" = Mark= -V "0" = Space= +V
4	Request to Send	CA	From ISC to DCE*	Conditions the DCE* for Transmission Always +V if terminal is on
5	Clear to Send	CB	From DCE* to ISC	Not required by ISC
7	Signal Ground	AB	NA	Connected to Pin 1 also
20	Data Terminal Ready	CD	From ISC to DCE*	Signals the DCE* that the data terminal is ready to transmit ON=+V=Ready OFF=-V=Not Ready

*DCE - Data Communication Equipment

RS232C INTERFACE

START-UP AND INITIALIZATION

Introduction

BEFORE ATTEMTPING TO OPERATE YOUR INTECOLOR® 8001, IT IS SUGGESTED THAT THIS SECTION BE READ AND UNDERSTOOD. The power switch (SW1) is located in the lower rear panel portion of the CRT case. Also located on this panel are the various input and output port connections. These are shown in Appendix A8. Connection diagrams are shown in Appendix A7.

Power

Plug the line cord into a 120VAC-60HZ outlet (230VAC-50-60 HZ with Option 11). When the power switch is pushed up the terminal is in the operating state. After the switch is turned on, a 60 second warm up period is required before operating the terminal. The unit will come up in the initialized state, S_O.

Initialized State - S_O

The unit will always come up in the initialized state-S_O when power is turned on after being off for at least 30 seconds.

In State S_O the following conditions are true:

A. Visible foreground color = white
B. Visible background color = black
C. Reverse field flag = "O"
D. Visible A7 bit = "O" (unless otherwise noted)
E. Plot Bit = "O"
F. Page Mode Operation (unless otherwise noted)
G. Terminal Mode = Local (unless otherwise noted)
H. Baud Rate = 9600 with one stop bit (unless otherwise noted)
I. Write left to right with visible cursor
J. Blind foreground color = red
K. Blind background color = black
L. Blind A7 Bit = "O"
M. Blind Plot Bit = "O"
N. Blind Cursor at home or top left corner of screen.

After the above conditions have been set, the cursor is moved to the home position which is the top left hand corner of the screen, and the position of the first character of the first line. The screen will clear by an Erase Page command which effectively makes all 2000 (3840 with 80 character x 48 line option) characters; spaces (20 HEX) which are white, non-blinking (Ø7 HEX). The unit is now ready to accept commands from the keyboard or the serial input if connected.

Convergence and Purity

The units convergence and purity may need adjusting when initially received. Allow at least a 30 minute warm before setting the final convergence. See Appendix C for convergence alignment.

SUMMARY OF CONTROL CODES
FOR INTECOLOR 8001

0 - NULL (control @) has no effect.

1 - PROTECT (control A) has no effect.

2 - PLOT (control B) enters graphic plot mode (see plot submodes).

3 - CURSOR XY (control C) enters X-Y cursor address mode for either visible cursor or blind cursor.

4 - FREE (control D) not used - has no effect.

5 - FREE (control E) not used - has no effect.

6 - CCI (control F) the next character which follows provides the 8 bit visible status word.

7 - BELL (control G) provides a 150 ms tone. [continuous tone.]

8 - HOME (control H) moves the cursor to top left corner of display.

9 - TAB (control I) causes cursor to advance to next column - the tab columns are every 8 characters.

10 - LINE FEED (control J) causes the cursor to move down one line.

11 - ERASE LINE (control K) causes the cursor to return to beginning of line and causes the complete line to be erased.

12 - ERASE PAGE (control L) causes the complete screen to be erased and the cursor moves to the home position.

13 - RETURN (control M) causes the cursor to move to the beginning of the line it presently is on.

14 - A7 ON (control N) turns the A7 bit flag on. PLOT BIG LETTERS!

15 - BLINK/A7 OFF (control O) turns the blink bit and A7 bit off.

16 - BLACK KEY (control P) sets either foreground or background to color black.

17 - RED KEY (control Q) sets either foreground or background to color red.

18 - GREEN KEY (control R) sets either foreground or background to color green.

19 - YELLOW KEY (control S) sets either foreground or background to color yellow.

20 - BLUE KEY (control T) sets either foreground or background to color blue.

21 - VIOLET KEY (control U) sets either foreground or background to color violet.

22 - CYAN KEY (control V) sets either foreground or background to color cyan.

23 - WHITE KEY (control W) sets either foreground or background to color white.

24 - XMIT (control X) causes data to be transmitted from the visible cursor to the end of page or until FF/00 is found in Refresh RAM.

25 - CURSOR RIGHT (control Y) causes the cursor to move right 1 position.

26 - CURSOR LEFT (control Z) causes the cursor to move left 1 position.

27 - ESC (control [ ) provides an entry to the escape code table- must be followed by one or more codes for proper operation.

28 - CURSOR UP (control \ ) causes the cursor to move up one line.

29 - FG ON/FLAG OFF (control]) sets the flag bit off.

30 - BG ON/FLAG ON (control ^) sets the flag bit on.

31 - BLINK ON (control _) sets the blink bit on.

SUMMARY OF ESCAPE CODES

FOR INTECOLOR 8001

	5 BIT CODE	LETTER	FUNCTION
	0	@	Visible cursor mode
	1	A	Blind cursor mode
*	2	B	Plot via color pad
	3	C	Transmit cursor X,Y position
	4	D	Not used
*	5	E	Re-entry to BASIC 8001
	6	F	Sets full duplex mode
	7	G	Not used
	8	H	Sets half duplex mode
	9	I	Not used
	10	J	Set write vertical mode
	11	K	Sets roll up and write left to right mode
	12	L	Sets local mode
	13	M	Not used
	14	N	Not used
*	15	O	Re-entry to the CPU operating system
*	16	P	Initializes and transfers control to the CPU operating system
*	17	Q	* Character insert mode
	18	R	Baud rate selection mode A7 on = 1 stop bit, A7 off = 2 stop bit
*	19	S	Transfer control to the 8080 assembler
*	20	T	Transfer control to the text editor
*	21	U	Insert one line
*	22	V	Delete one line
*	23	W	Initializes and transfers control to BASIC 8001

5 BIT CODE	LETTER	FUNCTION
24	X	Sets page mode and write left to right mode
25	Y	Test mode - fill page with next character
26	Z	Set write down on 45 degree mode
27	[	Not used
28	\	Sets write up on 45 degree mode
29	]	Set unit up for Block receive mode
30	^	Causes a jump to address 9FAØH
31	—	Transfer control to the CRT mode

* Must include certain option to be operational

SUMMARY OF GRAPHIC PLOT SUBMODES

FOR INTECOLOR 8001

RS-232 INPUT CODE	PLOT SUBMODE	NORMAL KEY-BOARD CODE	OPTIONAL FUNCTION KEYBOARD CODE
255	Plot Mode Escape	Control ?	F 15
254	Charactor Plot	Control >	F 14
253	X Point Plot	Control =	F 13
252	Y Point Plot	Control <	F 12
251	X-Y Incremental Point Plot	Control ;	F 11
250	X_0 of X Bar Graph	Control :	F 10
249	Y of X Bar Graph	Control 9	F 9
248	X max of X Bar Graph	Control 8	F 8
247	Incremental X Bar Graph	Control 7	F 7
246	Y_0 of Y Bar Graph	Control 6	F 6
245	X of Y Bar Graph	Control 5	F 5
244	Y max of Y Bar Graph	Control 4	F 4
243	Incremental Y Bar Graph	Control 3	F 3
242	X_0 Vector Plot	Control 2	F 2
241	Y_0 Vector Plot	Control 1	F 1
240	Incremental Vector Plot	Control Ø	F Ø

SUMMARY OF INCREMENTAL DIRECTION CODES
FOR INTECOLOR 8001

	Δ X1		Δ Y1		Δ X2		Δ Y2	
If BIT = 1	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Direction	+	-	+	-	+	-	+	-
Value	80	40	20	10	8	4	2	1

CRT REFRESH MEMORY LAYOUT

The 2000 [3840] * characters for display are stored in a 4096 [8192] word RAM memory beginning at 32,768 (8000 HEX) and ending at 36,767 (8F9F HEX) [40,447 (9DFF HEX)] . The first word is the zero character stored as the A_7 bit and then the 7 bit ASCII code (A_6 to A_0). The second word is the composite status for this character. It is composed of Plot Character Bit (A_7), Foreground Blink (A_6), Background color code (A_5, A_4, A_3), and Foreground color code (A_2, A_1, A_0).

Therefore, each screen character requires two 8 bit words in memory, (the screen character and the character's composite status). The RAM memory location 8FAO HEX [9FAØ] to 8FFF HEX [9FFF HEX] are used for scratch pad storages. Memory location 8FBO [9FBO HEX] and 8FB1 [9FB1] are the locations of the Cursor character position and line number respectively. With the Roll Mode (Option 15) memory location 8FB2 [9FB2] provides the number of lines that the home position has been shifted or rolled.

* [Indicates value for 48 Line System]

1A

REFRESH MEMORY WORD FOR ONE CHARACTER

EVEN

A7 bit	ASC II Code						
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
PLOT BITS							

ODD = (EVEN +1)

	BLINK BIT	B	G	R	FOREGROUND COLOR		
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
PLOT BIT		BACKGROUND COLOR			B	G	R

Secrets of the printing characters:

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ØØ	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
1Ø	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	#
2Ø		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3Ø	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4Ø	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5Ø	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6Ø	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7Ø	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	#
8Ø	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
9Ø	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	#
AØ		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
BØ	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
CØ	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
DØ	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
EØ	~	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
FØ	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	#

ØØ–5F: SMALL LETTERS

6Ø–7F: repeats ØØ to 1F, SMALL LETTERS.

8Ø–DF: BIG LETTERS...top half is printed on odd lines, bottom half on even lines. Same ASCII character prints both halves, differing only by line location.

EØ–FF: repeats 8Ø to 9F, BIG LETTERS.

Can get limited extra characters by plotting some letter tops with different letter bottoms...cannot top tops with tops.

PART II

Keyboard

The Intecolor 8001 has a detachable keyboard which presents the standard ASCII four level code. (See Appendix A-1 for keyboard layouts). The keyboard keys are optically encoded by means of phototransistors, a light source and shutters attached to the keys. There are no switches to wear out and the unit is RFI free. The Keyboard does not provide two key rollover.

CPU Reset

The CPU Reset key provides a reset signal to the 8080 CPU. Its primary function is to allow the operator to regain control of the terminal if the software the customer has installed gets hung in an endless loop. If the reset is operated properly the bell will issue a short beep upon the release of the key. If automatically forces the terminal to the S_o state. That is, just as if the power had been turned off and then back on. If additional RAM memory is installed this memory area is not cleared, but the scratch RAM area within the CRT Refresh RAM card is cleared.

Control Key

The control key must be held down while the proper alpha numeric key is depressed if a control function is desired. The control functions are either color coded or have its desired results engraved on top of the key. Those keys which have a name enclosed within a ( ) parentheses indicate that they are also standarized escape codes. The escape codes only require that the ESC key be depressed then the ( ) parentheses key desired.

Shift Key

The shift key must be held down while the proper alpha numeric key is depressed if a shifted function is desired. Note that both the control and shift key must be held down to generate certain codes from the keyboard using the alpha numeric keys. See Appendix A-2 for the keyboard code set.

DETAIL OF CONTROL CODES

All of the display commands can be entered either through the serial input port or the keyboard. The keyboard input port has the highest priority of all inputs or outputs. The eight bit Intecolor 8001 code set as shown in Appendix A-2 must be used for the serial input port. The display control commands are a subset of the 32 ASCII control code set, and a flow diagram of these commands is shown in Appendix A-3.

With some display commands, such as the Graphic Plot Mode, delays may be experienced at the higher baud rates. A chart for these delays is shown in Appendix A4.

The Intecolor® 8001 display commands has been expanded by an additional 32 commands via the ESC, character sequence as shown in Appendix 5. The terminal employs two input pointer flags, one for the keyboard and one for the RS232C input. Each flag may point to a different Mode of operation and thus the terminal can act differently from the keyboard as compared to the RS232 input. (See blind cursor operation Code 1 on page 19 .)

Code Ø   Null - (Control @)

Has no effect upon the display

Code 1   Protect - (Control A)

Not presently implemented so it has no effect upon the unit.

Code 2   Graphic Plot Mode - (Control B) - (Option 02)

The general Graphic Plot Mode is entered by a binary code 2 or a Control Code B. (See Appendix B). It should be noted that the XY Plot Mode is also entered at the same time. If a plot mode other than XY Point Plot is desired, the next word that follows should then be a binary code from 240 to 255. These codes represent the various plot submodes as shown in the summary of Graphic Plot Submodes.

An additional feature is available to allow a graphic plot to be erased by simply setting the Flag bit on before entering the plot mode. This causes an XOR function to exist when plotting. Therefore, if you plot the same point, bar or vector twice, the second time erases the original.

Once in the general Plot Mode, any of the plot submodes may be entered by sending the corresponding code to the terminal. When this code is received, a flag internal to the terminal, known as PLOFL, is set placing the terminal in the appropriate plot submode. It should be noted that in many of the plot submodes, PLOFL is automatically set to a different value upon completion of the operation of that submode causing the terminal to enter a new submode. This is done to make coding and operation of the terminal in the various plot functions easier for the operator. The various submodes and their interactions are explained in detail in Appendix B.

Code 3 <u>Cursor X-Y</u> (Control C)

The visible cursor may be positioned any where on the screen simply by sending a 3-word sequence beginning with 03. The next two words that follow determine that X character position (0-79) and Y line position (0-24) for 25 line unit or [0-47] for 48 line unit. Both X and Y values must be in binary form with the range indicated. The cursor home position (i.e., the top left hand corner) is position O, O while the bottom right hand corner is (79, 24) or [79, 47].

If the cursor is positioned at X = 80 binary (50 HEX) then the cursor will disappear. But if a character is typed it will be positioned at the beginning of the line specified by Y + 1, the cursor then reappears in character position 1. Any cursor command will automatically force the cursor to reappear at the proper position in relation to character position O, line Y + 1.

If the cursor X values is 81 binary (51 HEX) or larger then the CRT ignores this as the visible cursor X values and sends the unit into the blind cursor addressing mode. Once in the blind cursor X-Y addressing mode three (3) additional words must be sent. They are blind cursor X value, blind cursor Y value, and the blind status word. The blind X value must be in the range of 0-79 and the blind Y value must be in the range of 0-24 or [0-47]. The blind status word must be in the same format as required in the CCI mode (control F). See the next page,

The blind A7 bit will be set on by sending from 128 binary to 255 binary instead of 81 binary when going from the visible cursor X,Y mode to the blind cursor X,Y mode. The Blind A7 bit will be set off anytime a binary number between 81 and 127 is used to get into the blind X,Y mode.

It should be noted that the X and Y cursor values received are masked to 0-127 and 0-31 [0-63] respectively. Then, if the value is still out of range, the X value has 80 subtracted and the Y values has 25 [48] subtracted.

When exiting from the blind cursor X-Y mode the terminal is left in the blind cursor mode for what ever input device caused the mode to be entered. That is if after CPU reset is operated the keyboard causes the blind cursor XY to be addressed then the keyboard will be left in the blind cursor mode while the RS232 serial is still in the visible cursor mode.

Code 4 <u>EOT</u> - (Control D)

Has no effect upon the display

Code 5 <u>ENQ</u> - (Control E)

Has no effect upon the display

Code 6 <u>CCI</u> - (Control F)

When this code is received the system accepts the next eight bit word from the serial input as the new composite status for the characters which follow. See CRT Refresh Memory Section.

The first three bits represents the Foreground Color with $Red_F=A_0$, $Green_F=A_1$, and $Blue_F=A_2$. The next three bits represent the Background Color (optional) with $Red_B=A_3$, $Green_B=A_4$, and $Blue_B=A_5$. The next bit, A_6 is the Blink bit for the Foreground Color and the last bit, A_7 is Plot Character bit which causes the display to interpret the ASCII word as a 2x4 plot array.

Code 7 <u>Bell</u> - (Control G)

When this code is received a tone will sound for about 150 MS.
* Stays on until BASIC program reads an END, or INPUT...so a loop can be used to increase tone duration.

Code 8 <u>Home</u> - (Control H)

When this code is received the cursor moves to 0,0 or the top left hand corner of the screen.

Code 9 <u>Tab</u> - (Control I)

When this code is received the cursor moves horizontally to the next tab position. The tab positions are fixed and are at every eight positions from zero.

Code 10 <u>Line Feed</u> - (Control J)

When this code is received the cursor moves down one line. This is the only code used for cursor down.

Code 11 <u>Erase Line</u> - (Control K)

When this code is received a carriage return is initiated and the characters from the beginning to the end of the line are replaced with spaces and have the same color and status as the present visible CCI status. The cursor is always positioned at the beginning of the line.

Code 12 <u>Erase Page</u> - (Control L)

When this code is received the complete screen is replaced with spaces that have the same color and composite status as the present visible CCI status. The cursor always returns to the Home position. The blind cursor is also positioned at home.

Code 13 <u>Carriage Return</u> - (Control M)

When this code is received, the cursor returns to the beginning of the line that it presently is on.

Code 14 A_7 On - (Control N)

Upon receiving this code, the characters which are to be displayed have A_7 forced to a "1". This bit is used to allow 2X character sizes for 48 line units. This effectively doubles the number of displayable character types from 128 to 256.

Code 15 Blink - A_7 - OFF - (Control O)

When this code is received the characters which follow have A_7 set to "0" (i.e., opposite to A_7 On as above) and also have the Blink bit, A_6 of the composite status for the character set to "0" (i.e., the opposite of Blink-On per Code 31.)

Code 16 to 23 or Color Keys - There are eight color keys

			A_2	A_1	A_0
Black	(Control P)	Code 16	0	0	0
Red	(Control Q)	Code 17	0	0	1
Green	(Control R)	Code 18	0	1	0
Yellow	(Control S)	Code 19	0	1	1
Blue	(Control T)	Code 20	1	0	0
Magenta	(Control U)	Code 21	1	0	1
Cyan	(Control V)	Code 22	1	1	0
White	(Control W)	Code 23	1	1	1

When one of these eight codes is received then one of two things happens, depending upon the Flag bit. If the Flag is off then the key that is depressed will change the composite status to that Foreground Color code.

If the Flag is on, then the key that is depressed will change the composite status to the Background Color code. If Background Color option is used, then it will display that color. If Background Color option is not supplied, then no effect will be noticed.

Note that when the plot via color pad is selected, one of the eight color select keys will select one of the eight plot blocks. The plot option 2 is installed See Escape B section for details.

Code 24 Transmit - (Control X)

Whenever control X is received the terminal starts transmission from the visible cursor present position to the end of the screen, or until it detects a FF, ØØ Hex sequence in the Refresh memory.

The transmission sequence is terminated by a carriage return, either ØD Hex or 8D Hex at the customer option. It should be noted that there may be many ØD Hex or 8D Hex imbedded in the data transmission since these are legal words in the refresh memory.

The transmission sends each 8 bit word in memory in sequence. That sequence is the ASCII character, then the status of that character, followed by the next ASCII character and then its status until the FF, ØØ sequence is detected.

The best way to have this data sent back to the terminal is via the ESC] or block receive mode.

Code 25 Cursor Right - (Control Y)

Moves the cursor right one character without destroying any information.

Code 26 Cursor Left - (Control Z)

Moves the cursor left one character without destroying any information.

Code 27 Escape - (Control [ )

The Escape command effectively expands the control code set by 32 additional code capabilities. This requires at least a two code sequence (ESC, letter) which then performs a given function. At present only 26 of the 32 additional command capabilities have been enabled. These commands are given in the following table. (For Detail see the Escape Code Section).

SEE ESCAPE CODE TABLE - Page 17

ESCAPE CODE TABLE

OPTIONS	DECIMAL CODE	LETTER	FUNCTION
	0	@	Visible Cursor Operation
	1	A	Blind Cursor Operation
*	2	B	Plot Via Color Pad
	3	C	Transmit Cursor X,Y Position
	4	D	Not Used
*	5	E	Re-Entry Control to BASIC 8001 system
	6	F	Sets Unit to Full Duplex
	7	G	Not Used
	8	H	Sets Unit to Half Duplex
	9	I	Not Used
	10	J	Sets Unit to Write Vertical
*	11	K	Sets Unit to Roll up Mode & write Left to Right
	12	L	Sets Unit to Local Mode
	13	M	Not Used
	14	N	Not Used
*	15	O	Re-Entry control to the CPU Operating System
*	16	P	Initializing & Transfers Control to the CPU Operating System
*	17	Q	Allows Operation in Character Insert Mode
	18	R	Allows Selection of 1 of 7 Baud Rates
*	19	S	Transfers Control to the 8080 Assembler
*	20	T	Transfers Control to the Text Editor
*	21	U	Inserts one line (80 blanks)
*	22	V	Deletes one line (80 blanks)
*	23	W	Transfers Control to BASIC 8001 Software
	24	X	Sets Unit to Write Left to Right & Page Mode
	25	Y	Test Mode-Fills Screen with Next Character
	26	Z	Sets Unit to Write Down on 45 Degrees
	27	[(ESC)	Not Used
	28	\	Sets Unit to Write Up on 45 Degrees
	29	]	Sets Unit to Block Receive Mode
	30	^	Causes a Jump to Ram Address 9FAØH
	31	_	Transfers Control to the CRT Mode

The letters are presented for easy reference; i.e., (full duplex mode requires ESC, F sequence). It should be noted that the Escape control codes can be any 8 bit value so long as the 5 least significant bits are correct for the operation desired. The terminal simply masks off the undesired higher order bits. The Keyboard and RS232C input port has separate and independent Flags which determine some of the CRT modes. Therefore, the Keyboard may be in the character input mode while the RS232 input may be in the Plot mode or vise versa. The input port and the Keyboard can operate completely independently of each other. See Details of Escape Codes section for more information.

Code 28 Cursor Up - (Control \ )

Moves the cursor up one line without destroying any information. This is effectively the opposite of a Line Feed operation.

Code 29 Flag Off - (Control])

When this code is received the Reverse Field flag is set to "0". Effects the special character codes (96 to 127) and the color codes (16 to 23).

Code 30 Flag On - (Control ^)

When this code is received the Reverse Field flag is set to one. Effects the special character codes (96 to 127) ; the color codes (16 to 23); and the plot modes.

Code 31 Blink On - (Control _)

When this code is received the Blink bit A_6 of the composite status is set to a "1".

This bit is turned off when the Blink-Protect-Off key is operated (see Code 15).

Code 32 to 95 - Numbers and Letters

These provide the standard printing ASCII Upper Case characters, punctuations and numbers. See Appendix A-2 for code set of the the Intecolor® 8001.

Code 96 to 127 - Special Characters

These codes provide either 32 special characters (such as lower case ASCII characters) or 64 special characters. The 64 special characters are actually two groups of 32 special characters. A group is selected depending upon the condition of the Flag bit. If the flag bit is off then the codes are not changed when they are placed in the CRT refresh RAM. If the flag bit is on then these codes have 96 subtracted from them before they are replaced in the CRT refresh RAM. Therefore they are mapped into 0 to 31 within the CRT refresh memory.

DETAILS OF ESCAPE CODES

@ or Code 0 - Visible Cursor Mode

This is the terminal's normal mode of operation and it is also the startup state. A received character is placed at the visible cursor location. The cursor then advances to the right one position awaiting the next character. All normal cursor operations are applicable to placing the cursor at a different location.

A or Code 1 - Blind Cursor Mode

This optional mode provides for a dual cursor operation. That is, normally the host computer will operate in the blind cursor mode and the keyboard in the visible cursor mode. The two modes will not interact with each other. There is also a blind status which may be different than the visible status. The only blind cursor movements allowed are a subset of the cursor X-Y positioning. See Code 3 or control C. This mode allows operation without delay for ASCII TEXT at rates up to 38.4 K Baud.

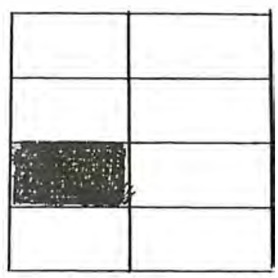
B or Code 2 - Character Plot Via Color Pad

When the plot option is installed then this plot mode will be available. It will normally be used via the color pad, but can be used without it. It provides a mix between the Plot Mode and the normal ASCII Character Mode. Instead of responding as described in Character Plot, this mode uses only eight codes to intensify each of the eight blocks within a character. These intensifying codes are the normal color select codes (Control P through W).

This option normally uses the color select pad on the keyboard. The pad is arranged as shown below.

Black	Blue
Red	Magenta
Green	Cyan
Yellow	White

Color Selection Pad



One Character Plot Aray

One Plot Block Selected by Green

Figure 2.5.6.1

From the above it is easy to see the one to one correspondence between the 2x4 color select pad and the 2x4 character plot blocks. Thus, this mode is designed especially for use by the keyboard to simplify the drawing of graphs or the correcting of graphs. Once in this mode a block at the top right hand corner of the cursor present position can be intensified by pushing the top right hand corner key in the color select pad, (in this case the blue key or Control T or Code 20). Once that plot block has been intensified, any other plot block at that same character location can also be intensified since the cursor does not automatically advance. If the blue key was to be pushed the second time, then the already intensified plot block will be extinguished. This effectively allows any one plot block to be erased. After all desired plot blocks have been either intensified or extinguished, the cursor may be conventionally moved without escaping from this special text and character plot mode. In fact, all of the control codes are effective while in this mode except the color select control codes, and any of the ASCII Text characters can be entered and displayed. Any code that requires a two key or more sequence (such as cursor X-Y, CCI, and ESC) will terminate the mode. It should be noted that the ASCII Character when entered and displayed advances the cursor as previously done in the visible mode, but the plot blocks (generated by the color pad) do not advance the cursor. Therefore, when a character position has been used to display plot blocks a cursor command must be given to advance the cursor to the next character position.

C or Code 3 - Transmit Cursor X,Y

When this code is selected the terminal sends out the following 7 word sequence:

03, X, Y, 06, Status, ASCII Character, CR.

The X and Y values represents the cursor position on the screen. The status is the status of the ASCII character at that cursor location. The CR may be either a ØD or an 8D HEX at the customer request.

This sequence of transmission is the same that the light pen would provide if the unit is so equipped.

E or Code 5 - Re-Entry to BASIC 8001

Return to BASIC 8001 without destroying the BASIC 8001 source program which is in Ram memory.

F or Code 6 - Full Duplex Mode

When this mode is selected then the Keyboard characters are only sent to the RS232C serial port. They are not processed by the terminal. Therefore, once the unit is put in the full duplex mode via the keyboard, then the only "normal" way the mode can be changed to local or half duplex is via the RS232C serial port. There are two other ways that have been provided to regain local control. One way is to operate the CPU Reset key on the Keyboard, which will initialize the terminal as if power has been just turned on. The other way is to operate the break key on the keyboard. When this is done a break of 150 MS will be transmitted on the RS232C serial port, and the terminal will be forced into the half duplex mode.

H or Code 8 - Half Duplex Mode

When this mode is selected then the keyboard characters are not only processed by the terminal but are also sent to the RS232C serial port.

J or Code 10 - Write Vertical Mode

This effects the visible cursor mode only and causes the terminal to enter characters vertically one below the other. All other cursor movements are possible via the cursor mode. After a character is entered the cursor is moved down one character awaiting the next character. Upon reaching the last line the next character will be on the top line, i.e. wrap around occurs.

K or Code 11 - Roll Mode (option 15) Write left to right

When this mode is selected the terminal will cause a page roll up when the last line has been filled. All 48 line units roll two lines at a time while 25 line units roll only one line. Note the plot mode and blind cursor mode only work in non-roll mode. This mode also sets the visible cursor to write left to right.

L or Code 12 - Local Mode

When this mode is selected then the keyboard characters are displayed on the terminal, but they are not sent to the RS232C serial port. In this mode the RS232C serial input port can receive data or change this mode. The terminal can be made to transmit out of the RS232C port, while in the local mode by typing Control X or ESC C.

O or Code 15 - Re-Entry to CPU Operating System Mode

Causes the same result as Code 16 below but does not reinitialize the I/O Byte or the second RS232C channel Baud rate.

P or Code 16 - Initialize CPU Operating System Mode

When this optional mode is selected the terminal enters into the CPU Operating System. It then obeys all the commands that are allowed in the CPU Operating System. See the CPU Operating System Manual.

Q or Code 17 - Character Insert Mode

Once in this mode the CRT acts exactly like the normal visible cursor system for all control commands except for those requiring a 2 or more character sequence (such as Cursor XY, CCI, and ESC). When any character is typed or received via the RS232C input, it is inserted within the line at the cursor present position and every character after the cursor to the end of the line is shifted right one character position. The last character on the line is lost forever. The cursor is also advanced one position. The above is true except for control codes, and "Delete" or (shift '__') keys (code 127).

When the "delete" key is depressed or code 127 is received via the RS232 input port then the character at the cursor present position is deleted and all characters to the end of the line are shifted left one character position. The last character on the line becomes a space. The cursor does not advance.

When the "ESC" key is depressed then the character insert-delete mode is terminated after the second character is selected. The terminal then normally returns to the visible character mode.

R or Code 18 - Baud Rate Selection Mode

When this mode is entered the unit then accepts the next character as one of seven baud rates. It does this by looking at only the first three bits. Therefore, any 8 bit character that has the desired 3 lower order bits will do. Normally the keyboard numbers 1 to 7 are used. The baud rates and the corresponding numbers are indicated in the table below:

Number	1	2	3	4	5	6	7
Normal Baud Rate	110	150	300	1200	2400	4800	9600
High Speed Baud Rate	880	1200	2400	9600	19,200	38,400	76,800

The unit is initialized with power up at normally 9600 baud, with one stop bit. This initialized baud rate can be specified by the customer at any of the fourteen above rates when ordered. It should be noted that only in certain modes (blind cursor mode) can the 38,400 Baud be used with delays. In no case can 76,800 Baud be used without delays. The unit may be ordered with either normal baud rates or with the High Speed Baud rates. The two different rate systems cannot be mixed.

The number of stop bits will be determined when the baud rate is set by the condition of the A7 flag. If A7 was on before the rate is selected, 1 stop bit is selected; if A7 was off before the rate is selected 2 stop bits are selected.

S or Code 19 - 8080 Assembler Mode

When this optional mode is selected the terminal enters into the 8080 Assembler Mode. It then obeys all the commands that are allowed in the 8080 Assembler. At present this option is not available.

T or Code 20 - Text Editor Mode

When this optional mode is selected the terminal enters into the Text Editor Mode. It then obeys all the commands that are allowed in the Text Editor. At present this option is not available.

U or Code 21 - Insert Line Mode

When this mode is selected the cursor moves to the beginning of the line it is presently on and this line and all lines to the end of the page is shifted down by one line. Then a new line of 80 spaces (or blanks) are inserted with the cursor remaining at the beginning of that new line.

Normally the cursor will be at the beginning of the line to be inserted when this mode is used. After a line has been inserted the terminal returns to the normal visible character mode.

V or Code 22 - Delete Line Mode

When this mode is selected the cursor moves to the beginning of the line it is presently on and this line is deleted. All lines to the end of the page are shifted up by one line. Then a new line of 80 spaces (or blanks) are inserted at the bottom of the page. The cursor will remain at the beginning of the line that had been deleted. After a line has been deleted the terminal returns to the normal visible character mode.

W or Code 23 - BASIC 8001 Language Mode

When this optional mode is selected the terminal enters into the BASIC 8001 Language mode. It then obeys all the commands that are allowed in Basic 8001. See the "BASIC 8001 Manual".

X or Code 24 - Page Mode Write Left to Right

When this mode is selected the terminal will not roll up when the last line has been filled, but will begin at home again. The terminal is also placed in the write left to right mode. This is the normal power up mode. This mode affects all modes that use the visible cursor. The blind cursor and plot modes will only operate in the page mode.

Y or Code 25 - TEST Mode

When this mode is selected the next character that follows causes the complete screen to be filled with that character. Note use ESC, Y, . for a convergence test pattern.

Z or Code 26 - Write Down 45 Mode

When this mode is selected the terminal will place the character at the present visible cursor and will then cause a cursor right followed by a line feed to occur. Therefore, the next character entered will be to the right and down one position from the previous character. When the bottom of the page is reached the next character will appear on the top of the screen, i.e., wrap around occurs.

[or ESC or Code 27 - No Effect Code

Performs a return to visible character mode.

\ or Code 28 - Write Up 45 Mode

When this mode is selected the terminal will place the character at the present visible cursor and will then cause a cursor right followed by a cursor up to occur. Therefore, the next character entered will be to the right and up one position from the previous characters. When the top of the page is reached the next character will appear on the bottom of the screen, i.e., wrap around occurs.

] or Code 29 - Block Receive Mode

Causes the unit to enter into the block receive mode. Uses the blind cursor to position the data. Looks for a (FF), (OO) HEX sequence to terminate back to the visible cursor mode. Note this is same format as when control (x) or page transmit is requested. Note page transmit starts at visible cursor and ends at end of page or when an (FF), (OO) HEX sequence is found.

^ or Code 30 - Jump to RAM 9FAØH

When this code is received the CRT O.S. branches to location 9FAØH. Therefore, the user must patch into RAM address 9FAØH a jump to his program.

— or Code 31 - Transfers Control to the CRT Operating System

When this code is received, the unit is forced to the CRT O.S. mode. If Option 34, the CPU O.S., is also installed, then a message will be printed saying:

YOU ARE NOW IN THE 8001 CRT MODE

DETAIL OF GRAPHIC PLOT SUBMODES

Code 2 Graphic Plot Mode - (Control B) - (Option 02)

The general Graphic Plot Mode is entered by a binary code 2 or a Control Code B. (See Appendix B). It should be noted that the XY Plot Mode is also entered at the same time. If a plot mode other than XY Point Plot is desired, the next word that follows should then be a binary code from 240 to 255. These codes represent the various plot submodes as shown in the summary of Graphic Plot Submodes.

An additional feature is available to allow a graphic plot to be erased by simply setting the Flag bit on before entering the plot mode. This causes an XOR function to exist when plotting. Therefore, if you plot the same point, bar or vector twice, the second time erases the original.

Once in the general Plot Mode, any of the plot submodes may be entered by sending the corresponding code to the terminal. When this code is received, a flag internal to the terminal, known as PLOFL, is set placing the terminal in the appropriate plot submode. It should be noted that in many of the plot submodes, PLOFL is automatically set to a different value upon completion of the operation of that submode causing the terminal to enter a new submode. This is done to make coding and operation of the terminal in the various plot functions easier for the operator. The various submodes and their interactions are explained in detail in Appendix B.

In addition to being able to enter the plot submodes from the general Plot Mode, any plot submode may be entered from any other plot submode with the exception of the Character Plot Mode.

Colors may be defined on a character by character basis only and the color of an individual plot block as well as all other intensified plot blocks within a character will be the most recent color defined when a new block is intensified in that character. To change a color, it is required that the Plot Mode or plot submode be terminated, the color changed, and the Plot Mode be re-entered.

The character grid is made up of 80 characters wide by 25 [48] lines high. The 0 reference point for all plotting is always the lower left corner. Each character is further broken up into 2 blocks wide by 4 blocks high which then causes the plot grid to be 160 blocks wide by 100 [192] blocks high. All plot submodes operate on this size grid and have the same reference point. Positive direction is considered up and to the right and negative direction is considered down and to the left.

All plot submodes and the general Plot Mode are terminated or exited by the binary code, 255. Whenever this code is received, the modes are terminated and must be re-entered as described above.

Appendix B-2 gives a convenient summary of the codes required to enter the Plot Mode and the various plot submodes as well as the status of PLOFL before and after each operation and the ranges of each operation.

<u>Plot Mode</u> - <u>Escape</u> - (255 binary)

This code is used to exit from the Plot Mode or any of the plot submodes. The control "?"or F15 is used to escape from the Plot Mode from the Keyboard.

<u>Character Plot</u> - (254 binary)

The Character Plot is entered by a 254 after the general Plot Mode, "2" or Control Code B, is entered. From the Keyboard use Control "**>**" or F14. It may also be entered directly from any of the other plot submodes. After entering the Character Plot, the next word will be treated as a plot character except for code 255 binary or (FF) hexadecimal (i.e. all eight bits are "1's"). See Appendix B-

The general Plot Mode and the Character Plot terminate upon receipt of a 255 code. The above procedure must be repeated after a 255 code terminates the Plot Mode and the plot submodes.

Other plot submodes may not be entered from the Character Plot. To enter other plot submodes, the Character Plot must be terminated, the general Plot Mode entered and the plot submode entered with its associated code.

The procedures for entering and exiting the Character Plot are shown below.

Function	Code
Plot Mode	2
Character Plot	254
Plot Character 1	0 to 254
.	
.	
.	
Plot Character n	0 to 254
Plot Escape	255

The Character Plot causes the 6 wide by 8 high dot matrix to be divided into 8 blocks organized 2 blocks wide by 4 blocks high. Each block consists of a sub-dot matrix of 3 dots wide by 2 dots high. Each block may be individually intensified by defining the bit (one of eight bits) associated with the block in the plot character. Bits may by "ORed" together for a combination of blocks in a plot character, creating a form of graphics for plotting data or drawing diagrams. Large characters may also be created by utilizing the blocks of several character positions to create a large 5x7 dot matrix.

X Point Plot - (binary 253)

The X Point Plot is automatically entered upon receipt of the general Plot Mode code, binary code 2, or Control Code B. It also may be entered directly from any of the other plot submodes except Character Plot. From the Keyboard use Control " = " or F13. After entering the X Point Plot, the next word defines the X value of the block that is desired to be plotted See Appendix B- The X value in this mode may range from binary 0 to 159 and all other values will cause 160 to be subtracted and the resultant value of X to be computed.

The X Point Plot may be terminated by code 255 which causes the general Plot Mode to be terminated also. Any of the other plo submodes may be entered directly from the X Point Plot by simply entering the appropriate plot submode codes which range from binary 240 to 254.

It should be noted that this mode does not cause a block to be intensified, but only causes the X value to be defined. Once the X value is sent, the terminal is automatically placed in the Y Point Plot mode. Thus, the next code sent will be the Y value, which may range from binary 0 to 99 [0-191]. Upon receipt of the Y value, a plot block will be intensified on the CRT screen at the X value and Y value intersection. The terminal is then automatically placed in the X Point Plot mode and the next word sent will be interpreted as an X value.

Therefore, once in the X Point Plot mode, new blocks may be defined by simply sending X values and Y values consecutively, without the necessity of re-entering the X or the Y Point Plot modes.

The procedures for entering and exiting the X Point Plot mode are shown below:

Function	Code
Plot Mode*	2
X_1 Value	0 to 159
Y_1 Value	0 to 99 (0-191)
.	
.	
.	
X_n Value	0 to 159
Y_n Value	0 to 99 (0-191)
Plot Escape	255
or	or
Plot Submode	240 to 254

* Automatically X Point Plot mode also

NOTE: SEND Code 253 between X,Y data sets if necessary for timing considerations. See Appendix A-4 for delays.

The X Point Plot in conjunction with the Y Point Plot allows any block on a 160 wide by 100 (192 for 48 Line) high block matrix to be positioned to and intensified. If the new block is within a character position that is a previously intensified ASCII character, then the ASCII character is replaced completely by the new block and its associated color.

Y Point Plot - (binary 252)

The Y Point Plot is entered by a binary 252 code after the general Plot Mode is entered. See Appendix B-From the Keyboard use Control "<" or F12. It may also be entered directly from any of the other plot submodes except Character Plot (binary 254). It is more commonly entered automatically from the X Point Plot mode. After entering the Y Point Plot, the next word defines the Y value of the block that is desired to be plotted and causes the block to be intensified in accordance with the Section on (binary 253). The Y value in this mode may range from binary 0 to 99 (0-191) and all larger values will cause 100 (192) to be subtracted from the new value of Y to be calculated.

Upon receipt of the Y value, the X Point Plot is automatically entered by the terminal. The X value of the next block to be plotted may then be sent as explained in the Section on (binary 253).

The Y Point Plot is terminated by Code 255 which causes the general Plot Mode to be terminated also. Any of the other plot submodes may be entered directly from the Y Point Plot by simply entering the appropriate plot submode codes which range from binary 240 to 254.

Therefore, once in the Y Point Plot mode, new points may be defined by simply sending X values and Y values consecutively without the necessity of re-entering the X or the Y Point Plot modes. The procedures for entering and exiting the Y Point Plot mode are shown below:

Function	Code
Plot Mode	2
Plot Submode	252
Y_1 Value*	0 to 99
X_2 Value	0 to 159
Y_2 Value	0 to 99
.	
.	
.	
X_n Value	0 to 159
Y_n Value	0 to 99
Plot Excape	255
or	or
Plot Submode	240 to 254

* Plots point using whatever previous X Value left in memory.

NOTE: Send Code 253 between X,Y data sets if necessary for timing considerations. See Appendix A-4 for Delays.

XY Incremental Point Plot - (binary 251.)

The XY Incremental Point Plot is entered by code 251 after the general Plot Mode is entered. From the Keyboard use Control ";" or F11. It may also be entered directly from any of the other plot submodes, except Character Plot. After entering the XY Incremental Point Plot mode, the next word defines the next two increments as shown in Figure below. This word may have a range from binary 0 to 239 since binary 240 to 255 is used for the plot submode codes.

<table>
<tr><td>b_7</td><td>b_6</td><td>b_5</td><td>b_4</td><td>b_3</td><td>b_2</td><td>b_1</td><td>b_0</td></tr>
<tr><td colspan="2">ΔX_1</td><td colspan="2">ΔY_1</td><td colspan="2">ΔX_2</td><td colspan="2">ΔY_2</td></tr>
<tr><td colspan="4">Plot
Block 1</td><td colspan="4">Plot
Block 2</td></tr>
</table>

b_{n+1}	b_n	
0	0	No Change
1	0	Positive Increment
0	1	Negative Increment
1	1	No Change

n= 0, 2, 4, 6

If b_o through b_3 are "0"s, then the plot block will not print but will increment one increment according to the coding of b_4 through b_7. This allows the user to easily "skip" a plot increment by plotting with an invisible block.

It should be noted that the XY Incremental Plot mode does not automatically transfer the terminal to any other plot submode upon receipt of an incremental change word,but remains in the XY Incremental Plot mode ready to receive another incremental change word. Therefore, a series of incremental movements may be made by sending consecutive incremental change words.

The XY Incremental Plot mode may be terminated by code 255 which causes the general Plot Mode to be terminated also. Any of the other plot submodes may be entered directly from the XY Incremental Point Plot by simply entering the appropriate plot submode codes which range from binary 240 to 254.

The procedures for entering and exiting the XY Incremental plot mode are shown below:

Function	Code
Plot Mode	2
or	or
Plot Submode	240 to 253
XY Incremental Point Plot	251
Incremental Change Word 1	0 to 239
.	
.	
.	
Incremental Change Word n	0 to 239
Plot Escape	255
or	or
Plot Submode	240 to 254

NOTE: Send code 251 between XY incremental point words if necessary for timing considerations. See Appendix A-4 for Delays.

X Bar Graph, X_o Value - (binary 250)

The X Bar Graph, X_o Value is entered by a binary 250 code after the general Plot mode is entered. From the Keyboard use Control ":" or F10. It may also be entered from any of the other plot submodes except Character Plot. After entering the X Bar Graph, X_o Value Mode, the next word sent defines the X_o Value or the left horizontal start block of the horizontal bar graph. The graph grid is referenced to the lower left hand corner of the face of the CRT. The X_o may range in value from 0 to 159 and all other values have 160 subtracted and the new value calculated for X_o.

Upon receipt of the X_o Value, the value of X_o is stored in memory and the terminal is automatically placed in the X Bar Graph, Y Value mode (binary 249). The terminal is now ready to receive the next eight bit word as the Y position of the bar graph. Upon receipt of the Y value, the terminal is then automatically placed in the X Bar Graph, X Max Value mode (binary 248). The terminal is now ready to receive the next eight bit word as the X Max Value. Upon receipt of the X Max Value, the bar is drawn on the CRT and the terminal is placed back into the X Bar Graph, Y Value mode (binary 251) ready to receive a new Y value to begin the bar graph drawing process over again as outlined above. This process is shown below and in Appendix B.

Function	Code
Plot Mode	2
or	or
Plot Submode	240 to 253
X Bar Graph, X_o Value	250
X_o Value Word 1	0 to 159
Y Value Word 1	0 to 99 (0-191)
X Max Value Word 1	0 to 159
Y Value Word 2	0 to 99 (0-191)
X Max Value Word 2	0 to 159
.	
.	
.	
Y Value Word n	0 to 99 (191)
X Max Word n	0 to 159
Plot Escape	255
or	or
Plot Submode	240 to 254

NOTE: Use Code 251 between Y value, X max Value data sets for timing considerations. Timing delays depends directly upon the length of the bar being intensified. See Appendix A-4 for delays both minimum and maximum.

As can be seen from the above process, once in the X Bar Graph, X_o mode, it is necessary to send only two words, Y and X Max, to completely define other bar graphs with the same X_o in the horizontal direction. As before, any of the submodes can be entered independently. After the first bar graph sequence, additional bar graphs can be described by a new Y position for the graph and a new X Max

Value for the graph. The bar is drawn after the X Max Value is received using the original value of X_o.

Any of the other plot submodes may be entered directly from the X Bar Graph, entering the appropriate plot submode codes which range from binary 240 to 254.

This mode allows bar graphs in any color or multiple colors to be drawn with a width as small as one plot block wide or areas under curves may be easily filled in.

<u>X Bar Graph, Y Value</u> - (binary 249)

The X Bar Graph, Y Value is entered by a binary 249 code after the general Plot Mode is entered. From the Keyboard use Control "9" or F9. It is more commonly entered from the X Bar Graph, X_o Value automatically, and may also be entered from any of the other plot submodes except Character Plot (binary 254). After entering the X Bar Graph, Y Value mode, the next word sent defines the Y or vertical position of the horizontal bar graph being drawn. The Y value may range from binary 0 to 99 (0 to 191) and all other values will have 100 (192) subtracted from it and the new value calculated for the Y value.

Upon receipt of the Y value word, the value of Y is stored in memory and the terminal is automatically placed in the X Bar Graph, X Max Value mode, as explained more completely in the Section on (binary 248).

Any of the other plot submodes may be entered directly from the X Bar Graph, Y Value mode by simply entering the appropriate plot submode codes which range from binary 240 to 254.

<u>X Bar Graph</u>, <u>X Max Value</u> - (binary 248)

The X Bar Graph, X Max Value is entered by a binary 248 code after the general Plot Mode is entered. From the Keyboard use Control "8" or F8. It is more commonly entered from the X Bar Graph, Y Value automatically, and may also be entered from any of the other plot submodes except Character Plot. After entering the X Bar Graph, X Max Value mode, the next word received defines the X Max horizontal point of the horizontal bar graph being drawn. The X Max Value may range from 0 to 159 and all other values will have 160 subtracted from it and the new value calculated for X Max Value.

Upon receipt of the X Max Value word, the bar graph is drawn in the predefined color on the face of the CRT according to the X_o and Y value stored in memory from previous operations. The terminal is then automatically placed in the X Bar Graph, Y Value mode, binary 249, for the beginning of a new bar graph as more completely explained in the Section on (binary 248).

Any of the other plot submodes may be entered directly from the X Bar Graph, X Max Value mode by simply entering the appropriate plot submode codes which range from binary 240 to 254.

X Incremental Bar Graph - (binary 247)

The X Incremental Bar Graph is entered by a binary 247 code after the general Plot Mode is entered. From the Keyboard use Control "7" or F7. It may also be entered from any of the other plot submodes except Character Plot. After entering the X Incremental Bar Graph mode, the next word sent defines the next two horizontal and vertical increments for two horizontal bar graphs. Thus, one may position a bar graph each side of the present location and add or subtract an increment to the bar graph previously defined. The coding and composition is the same as explained in the Section on (binary 251). An example is shown in Appendix B-6.

Y Bar Graph, Y_o Value - (binary 246)

The Y Bar Graph, Y_o Value is entered by a binary 246 code after the general Plot Mode is entered. From the Keyboard use Control "6" or F6. It may also be entered from any of the other plot submodes except Character Plot. After entering the Y Bar Graph, Y_o Value mode, the next word sent defines the Y_o or the vertical start point of the vertical bar graph being drawn. The range of the Y_o word is 0 to 99 (0-191) and all other values have 100 (192) subtracted and will have the new value calculated for Y_o Value.

All other operations are identical as explained in the Section on (binary 250), X Bar Graph, X_0 Value except that Y Bar Graph, X Value and Y Bar Graph, Y Max Value are applicable for drawing vertical bar graphs. An example is shown in Appendix B-5.

Y Bar Graph, X Value - (binary 245)

The Y Bar Graph, X Value is entered by a binary 245 code after the general Plot Mode is entered. From the Keyboard use Control "5" or F5. It is more commonly entered from the Y Bar Graph, Y_o Value automatically, and may also be entered from any of the other plot submodes except Character Plot. After entering the Y Bar Graph, X Value mode, the next word sent defines the X, or horizontal position of the vertical bar graph being drawn. The X Value may range from 0 to 159 and all other values will have 160 subtracted and will have the new value calculated for the X value.

All other operations are identical as explained in the Section on binary 249, X Bar Graph, Y Value except that Y Bar Graph, Y_0 Value and Y Bar Graph, Max Value are applicable for drawing vertical bar graphs. An example is shown in Appendix B-5.

Y Bar Graph, Y Max Value - (binary 244)

The Y Bar Graph, Y Max Value is entered by a binary 244 code after the general Plot Mode is entered. From the Keyboard use Control "4" or F4. It is more commonly entered from the Y Bar Graph, X

Value automatically, and also may be entered from any of the other plot submodes except Character Plot. After entering the Y Bar Graph, Y Max Value mode, the next word received defines the vertical Y Max point of the vertical bar graph being drawn. The Y Max Value may range from binary 0 to 99 (0-191) and all other values will have 100 (192) subtracted and will have the new value calculated for Y Max Value.

All other operations are identical as explained in the Section on (binary 248), X Bar Graph, X Value, except that Y Bar Graph, Y_0 Value and Y Bar Graph, X Value are applicable for drawing vertical bar graphs. An example is shown in Appendix B-5.

Y Incremental Bar Graph - (binary 243)

The Y Incremental Bar Graph is entered by a binary 243 code after the general Plot Mode is entered. From the Keyboard use Control "3" or F3. It may be entered from any of the plot submodes except Character Plot. After entering the Y Incremental Bar Graph mode, the next word sent defines the next two horizontal and vertical increments for two vertical bar graphs.

All other operations are identical as explained in the Section on (binary 247), X Incremental Bar Graph except for the mode being applicable for drawing vertical bar graphs. An example is shown in Appendix B-6.

Vector Mode X_0 Value - (binary 242)

The Vector Mode is entered by a binary 242 code after the general Plot Mode is entered. From the Keyboard use Control "2" or F2. It may be entered from any of the plot submodes except Character Plot. After entering the Vector Mode, X_0 Value, the next word defines the X_0 Value point of the vector being drawn.

The Vector Mode requires the two end points to be defined (i.e. X_0 Y_0 and X_1 Y_1). The X_1, Y_1 values should previously be defined by way of the X,Y Point Plot Mode.

Upon receipt of the X_0 Value the terminal is automatically placed in the Vector Y_0 Value Mode (binary 241). The terminal is now ready to receive the next eight bit word as the Y_0 Vector Value. Upon receipt of the Y_0 Value the terminal then determines the best straight line fit between X_0, Y_0 and X_1, Y_1 using the plot blocks. The terminal will then revert to the Vector Mode X_0 value (binary 242), ready to receive the new X_0 Value for another vector. The process is shown below and in Appendix B-7.

Function	Code
Plot Mode	2
or	
X point Plot submode	253
X_1 Vector point 1	0 to 159
Y_1 Vector point 1	0 to 99 (191)
X_o Vector plot submode	242
X_o Vector point 1	0 to 159
Y_o Vector point 1	0 to 99 (191)
.	
.	
.	
X_o Vector point N-1	0 to 159
Y_o Vector point N-1	0 to 99 (191)
X_o Vector point N	0 to 159
Y_o Vector point N	0 to 99 (191)
Plot Escape	255
or	
Plot Submode	240 to 254

NOTE: Send code 242 between Y_o vector point and X_0 vector point words if necessary for timing considerations. See Appendix A-4 for delays.

Vector Mode Y_o Value - (binary 241)

The Y_o vector is entered by binary 241 code after the general Plot Mode is entered. From the keyboard use Control "1" or F1. This mode is more commonly entered automatically from the X_o Vector mode. After entering the Y_o Vector mode, the next word defines the Y_o value of the vector being drawn. There is no restriction on Y_o with respect to Y_1 except it must be in the range of 0 to 99 (191). Upon receipt of the Y_o value a vector is drawn from X_1, Y_1 to X_o, Y_o, with the new X_1Y_1 now at the old X_oY_o. If the next vector has a X_1Y_1 value = X_oY_o old, then only the new X_oY_o need be sent. This would effectively draw a vector from the present X_o Y_o position to the new X_oY_o point. See Appendix B-7.

X_o Y_o - Incremental Vector Mode - (binary 240)

The X_o-Y_o incremental vector mode is entered by a binary 240 code after the general plot mode is entered. From the keyboard use control "Ø" or FØ. It may also be entered from any of the other plot submodes except Character Plot. After entering the incremental vector mode, the next word sent defines the increments in X_o, Y_o, X_1 and Y_1 point values for the vector from X_1Y_1 to X_oY_o. This word may have a range from binary 0 to 239 since binary 240 to 255 are used for the plot submode codes.

Referring to the section on (binary 251), XY Incremental Point Plot it can be seen that there is one two bit element available for each of the 4 points (i.e. X_o, Y_o, X_1 and Y_1). The ΔX_1, ΔY_1 refers to the increment in X_1, Y_1 of the vector and the ΔX_2, ΔY_2 refers to the increment in X_o, Y_o of the vector.

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
X_1+1	X_1-1	Y_1+1	Y_1-1	X_o+1	X_o-1	Y_o+1	Y_o-1

Therefore, if b_4 and b_5 are both 1 or both Ø then no increment will take place. If either half of the word is all zero then the corresponding X,Y will be changed but no vector will be drawn. This allows the user to easily "skip" points. The only time a vector will be drawn is when both halfs of the word are non zero.

The incremental vector plot mode does not automatically transfer control to any other mode. It remains in this incremental mode until terminated by a plot submode code. Therefore a series of incremental movements in both X_o, $Y_o$and X_1Y_1may be made by sending consecutive incremental change words.

The procedure for entering and exiting the XY Incremental plot mode are shown below:

Function	Code
Plot Mode	2
or	or
Plot Submode	240 to 253
Incremental Vector Plot Mode	240
Incremental change in X_1, Y_1, X_o, Y_o	
Word 1	0 to 239
.	
.	
.	
.	
Word N	0 to 239
Plot Escape	255
or	or
Plot Submode	240 to 254

NOTE: Send code 240 between incremental vector words if necessary for timing considerations. See Appendix A-4 for input Delay Times.

LIGHT PEN OPERATION (Option 28)

The Intecolor 8001 Light Pen is designed to move the cursor on the screen of the terminal by simply pointing to the desired location on the screen and touching with the forefinger the touch-sensitive end of the light pen. The touch sensitive end of the light pen acts as a switch or button.

To effect operation of the light pen, the pen is simply pointed to the desired location on the screen. Either the standard lense or the long range lense may be used in the same manner. When the desired location is reached, the forefinger is placed on the touch-sensitive end of the pen and held on the pen until the cursor on the screen resides at the location the pen is pointing to. As long as the finger is kept on the pen the cursor will follow the pen to any location.

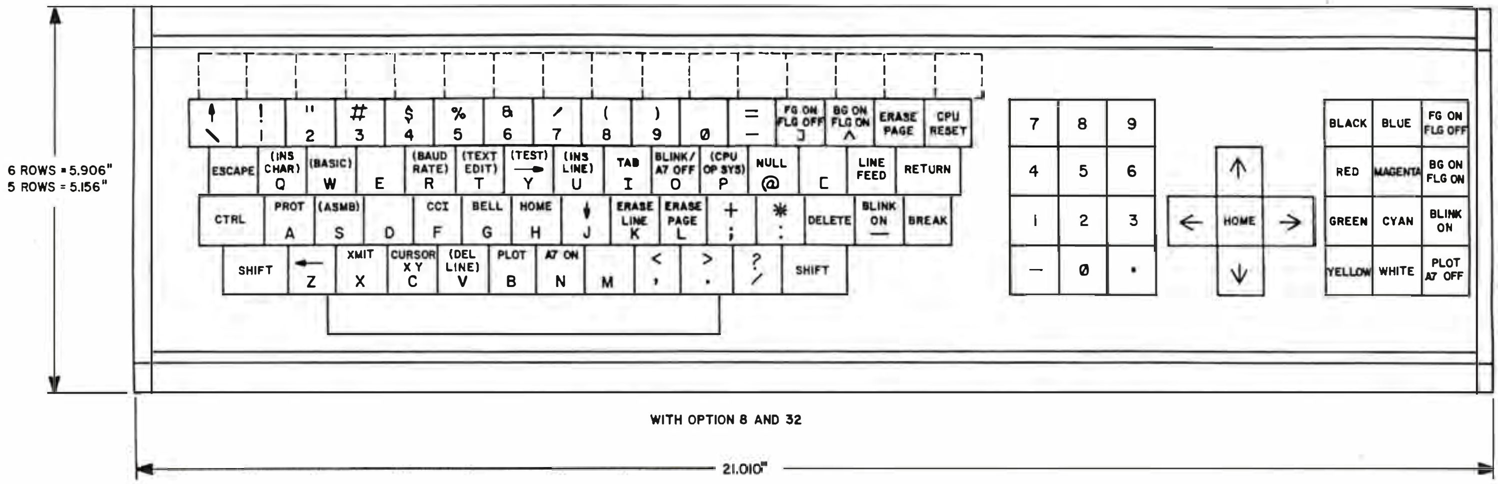
When the cursor is at the desired location, lift the forefinger from the tip of the pen and the following 7 word sequence will be transmitted to the J1 RS232 output port.

03	Cursor X-Y (See Code 3)
X	X Cursor Coordinate
Y	Y Cursor Coordinate
06	CCI (See Code 6)
Status	Status Character (See Appendix A-6)
ASCII or Special Character	
8D	Carriage Return

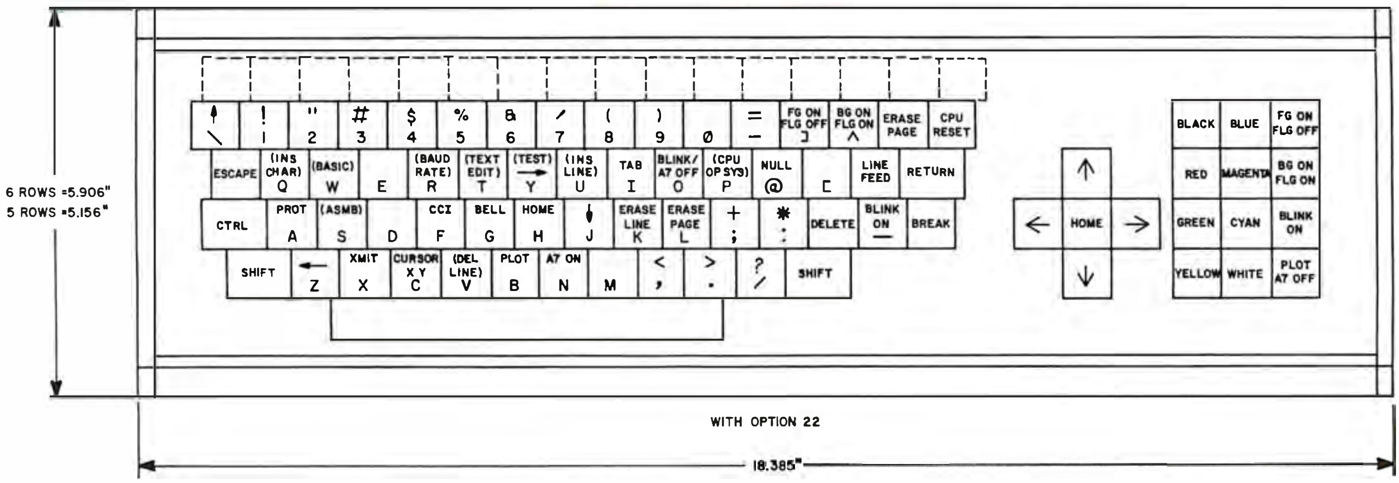
Notice that this sequence is not transmitted unless the finger first touches the end of the pen in the touch sensitive area and is effected when the finger is lifted from the end of the pen.

Note that a blue flood is normal operation and occurs every time the touch sensitive end of the pen is touched by the forefinger and will repeat at a 2cps rate until the finger is lifted.

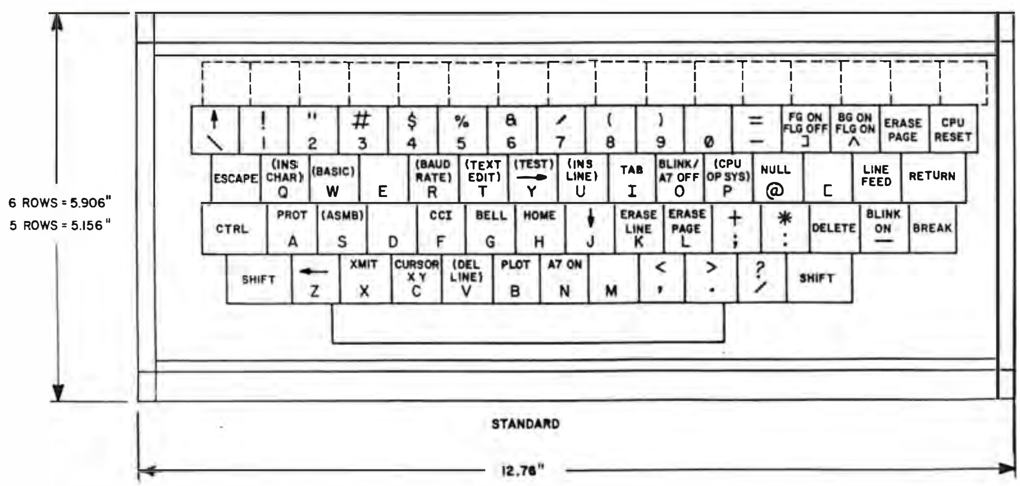
APPENDIX A



WITH OPTION 8 AND 32



WITH OPTION 22



STANDARD

NOTE : IF 16 FUNCTION KEYS ARE REQUIRED THEN THEN A SIXTH ROW OF KEYS ARE ADDED AS SHOWN DOTTED.

KEYBOARD LAYOUTS

THIS FUNCTION IS ASCII

C=CHR$(C):? C,C$

	C	C$
127	255	#
126	254	~
125	253	}
124	252	|
123	251	{
122	250	z (lower case)
↓	↓	↓
97	225	a (lower case)
96	224	`
95	223	_
94	222	^
93	221	]
92	220	\
91	219	[
90	218	Z (upper case)
↓	↓	↓
65	193	A (upper case)
64	192	@
63	191	?
62	190	>
61	189	=
60	188	<
59	187	;
58	186	:
57	185	9
↓	↓	↓
48	176	Ø
47	175	/
46	174	.
45	173	-
44	172	,
43	171	+
42	170	*
41	169	)
40	168	(
39	167	'
38	166	&
37	165	%
36	164	$
35	163	#
34	162	"
33	161	!
	160	?

	C	C$
31	159	BLINK ON
30	158	?
29	157	?
28	156	?SKIPS
27	155	HALT
26	154	CURSOR LEFT
25	153	?
24	152	?
23	151	PLOT WHITE
22	150	PLOT CYAN
21	149	PLOT MAGENTA
20	148	PLOT BLUE
19	147	PLOT YELLOW
18	146	PLOT GREEN
17	145	PLOT RED
16	144	PLOT BLACK
15	143	PLOT SMALL CHARACTERS
14	142	PLOT LARGE CHARACTERS
13	141	CURSOR LEFT
12	140	ERASE & HOME.
11	139	ERASE LINE, CURSOR LEFT.
10	138	CURSOR DOWN 1 LINE.
9	137	TAB (1)
8	136	CURSOR HOME
7	135	BEEP, BEEP!
6	134	RED ON RED
5	133	HALT
4	132	HALT
3	131	? DOWN 1/2 PAGE
2	130	CRASH
1	129	?
Ø	128	?
	127	#
	126	~
	↓	↓

Repeats all things like 128 greater.

KEYBOARD INPUTS: I=INP(1) IS NOT ASCII!

Ø=64
1=65
2=66
↓=↓
9=73

MUST SUB 16 OR ADD 112: Ø=176 ↓ ↓ 9=185

@=128
A=129
B=130
C=131
D=132
E=133
↓=↓
Z=202

MUST SUB 64 OR ADD 64: @=192, A=193 ↓ ↓ Z=218

WHITE =	215
CYAN =	214
MAGENTA =	213
BLUE =	212
YELLOW =	211
GREEN =	210
RED =	209
BLACK =	208

MUST SUB 92 OR SUB 64

` =160
a=161
b=162
c=163
↓ ↓
z=234

MUST SUB 64 OR ADD 64

STATUS 7	6	5	4	3	2	1	0
Plot	Blink	BB	BG	BR	FB	FG	FR

GREEN: to get ASCII from INP(1) non-standard keyboard.

					HEXADECIMAL	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
					A7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
					A6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
					A5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
HEXADECIMAL					A4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
	A3	A2	A1	A0		CONTROL @ TO O 143	CONTROL P TO _	SHIFT Ø TO ?				SHIFT * @ TO O	SHIFT * P TO _	CONT-SHIFT @ TO O	CONT-SHIFT P TO _	CONT-SHIFT Ø TO ?	CONT Ø TO ?	CONTROL F0-F15	SHIFT F0-F15	CONT-SHIFT F0-F15	F0-F15	OPTIONAL FUNCTION KEYS
0	0	0	0	0		0 NULL -16 144	16 BLACK 208	32 SPACE +64 96	48 Ø +112 64	64 @ +64 128	80 P +16 192	96 \/G0 +32 160	112 P/G16 -16 224	128 @ +64	144 P +16	160 SPACE +64	176 Ø +112	192 F0 P	208 F0	224 F0	240 F0	Xo-Yo INCREMENTAL VECTOR
1	0	0	0	1		1 PROTECT -16 145	17 RED 209	33 ! +64 97	49 1 +112 65	65 A +64 129	81 Q +16 193	97 a/G1 161	113 q/G17 225	129 A +64	145 Q +16	161 ! +64	177 1 +112	193 F1 Q	209 F1	225 F1	241 F1	Yo VECTOR PLOT
2	0	0	1	0		2 PLOT MODE -16 146	18 GREEN 210	34 " +64 98	50 2 +112 66	66 B +64 130	82 R +16 194	98 b/G2 162	114 r/G18 226	130 B +64	146 R +16	162 " +64	178 2 +112	194 F2 R	210 F2	226 F2	242 F2	Xo VECTOR PLOT
3	0	0	1	1		3 CURSOR X-Y MODE -16 147	19 YELLOW 211	35 # +64 99	51 3 +112 67	67 C +64 131	83 S +16 195	99 c/G3 163	115 s/G19 227	131 C +64	147 S +16	163 # +64	179 3 +112	195 F3 S	211 F3	227 F3	243 F3	INCREMENTAL Y BAR
4	0	1	0	0		4 FREE -16 148	20 BLUE 212	36 $ +64 100	52 4 +112 68	68 D +64 132	84 T +16 196	100 d/G4 164	116 t/G0 228	132 D +64	148 T +16	164 $ +64	180 4 +112	196 F4 T	212 F4	228 F4	244 F4	Ym-YBAR
5	0	1	0	1		5 FREE -16 149	21 MAGENTA 213	37 % +64 101	53 5 +112 69	69 E +64 133	85 U +16 197	101 e/G5 165	117 u/G21 229	133 E +64	149 U +16	165 % +64	181 5 +112	197 F5 U	213 F5	229 F5	245 F5	X-Y BAR
6	0	1	1	0		6 CCI -16 150	22 CYAN 214	38 & +64 102	54 6 +112 70	70 F +64 134	86 V +16 198	102 f/G6 166	118 v/G22 230	134 F +64	150 V +16	166 & +64	182 6 +112	198 F6 V	214 F6	230 F6	246 F6	Yo-Y BAR
7	0	1	1	1		7 BELL -16 151	23 WHITE 215	39 ' +64 103	55 7 +112 71	71 G +64 135	87 W +16 199	103 g/G7 167	119 w/G23 231	135 G +64	151 W +16	167 ' +64	183 7 +112	199 F7 W	215 F7	231 F7	247 F7	INCREMENTAL X BAR
8	1	0	0	0		8 HOME -16 152	24 TRANSMIT 216	40 (+64 104	56 8 +112 72	72 H +64 136	88 X +16 200	104 h/G8 168	120 x/G24 232	136 H +64	152 X +16	168 (+64	184 8 +112	200 F8 X	216 F8	232 F8	248 F8	Xm-X BAR
9	1	0	0	1		9 TAB -16 153	25 CURSOR RIGHT 217	41) +64 105	57 9 +112 73	73 I +64 137	89 Y +16 201	105 i/G9 169	121 y/G25 233	137 I +64	153 Y +16	169) +64	185 9 +112	201 F9 Y	217 F9	233 F9	249 F9	Y-X BAR
A	1	0	1	0		10 LINE FEED -16 154	26 CURSOR LEFT 218	42 * +64 106	58 : +112 74	74 J +64 138	90 Z +16 202	106 j/G10 170	122 z/G26 234	138 J +64	154 Z +16	170 * +64	186 : +112	202 F10 Z	218 F10	234 F10	250 F10	Xo-X BAR
B	1	0	1	1		11 ERASE LINE -16 155	27 ESC 219	43 + +64 107	59 ; +112 75	75 K +64 139	91 [+16 203	107 k/G11 171	123 {/G27 235	139 K +64	155 [+16	171 + +64	187 ; +112	203 F11 [	219 F11	235 F11	251 F11	X-Y INCREMENTAL PLOT
C	1	1	0	0		12 ERASE PAGE -16 156	28 CURSOR UP 220	44 , +96 176	60 < +80 108	76 L +64 140	92 \ +16 204	108 l/G12 172	124 |/G28 236	140 L +64	156 \ +16	172 , +96	188 < +80	204 F12 \	220 F12	236 F12	252 F12	Y-POINT PLOT
D	1	1	0	1		13 CR +16 157	29 FOREGND ON FLAG OFF 221	45 - +96 177	61 = +80 109	77 M +64 141	93] +16 205	109 m/G13 173	125 }/G29 237	141 M +64	157] +16	173 - +96	189 = +80	205 F13]	221 F13	237 F13	253 F13	X-POINT PLOT
E	1	1	1	0		14 A7 ON +16 158	30 BACKGND ON FLAG ON 222	46 . +16 178	62 > +80 110	78 N +64 142	94 ^ +16 206	110 n/G14 174	126 ~/G30 238	142 N +64	158 ^ +16	174 . +96	190 > +80	206 F14 ^	222 F14	238 F14	254 F14	CHAR PLOT
F	1	1	1	1		15 BLINK A7 OFF +16 159	31 BLINK ON 224	47 / +16 79	63 ? +80 111	79 O +64 143	95 _ +16 207	111 o/G15 175	127 #/G31 239	143 O +64	159 _ +16	175 / +96	191 ? +80	207 F15 -	223 F15	239 BREAK (F15)	255 F15	PLOT ESCAPE

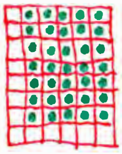
RED: values obtained from keyboard.

GREEN: ASCII correction constants.

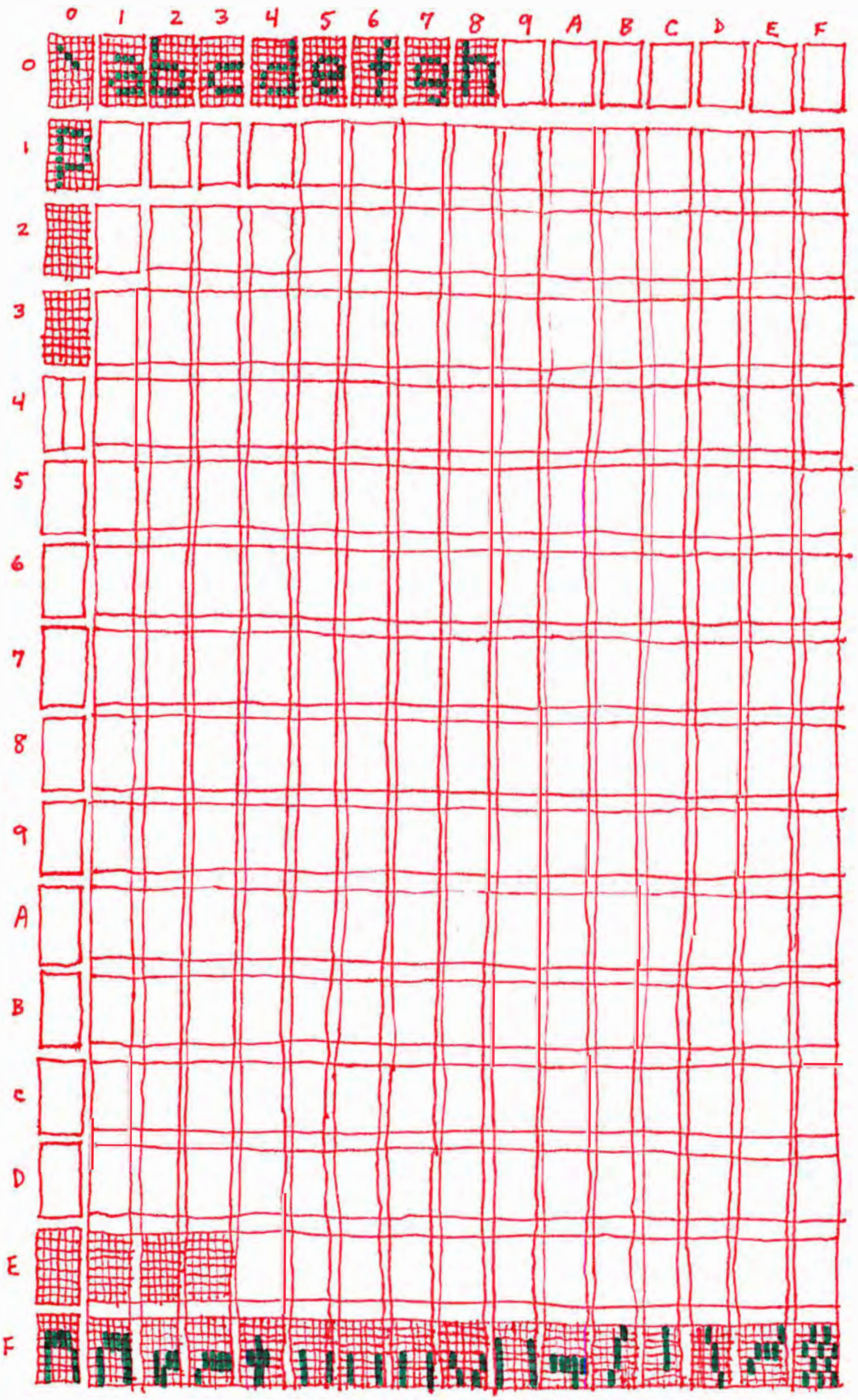
NOTE: THE TERMINAL ACCEPTS ALL 80 TO BF HEX CODES FROM THE KEYBOARD AND REASSIGNS THEM F0 TO FF HEX WHEN IN THE PLOT MODE, UNLESS THE OPTIONAL KEYS ARE INSTALLED. THEREFORE WITHOUT THE FUNCTION KEYS THE KEYBOARD CAN PLOT IN A RANGE OF 0 TO 175.

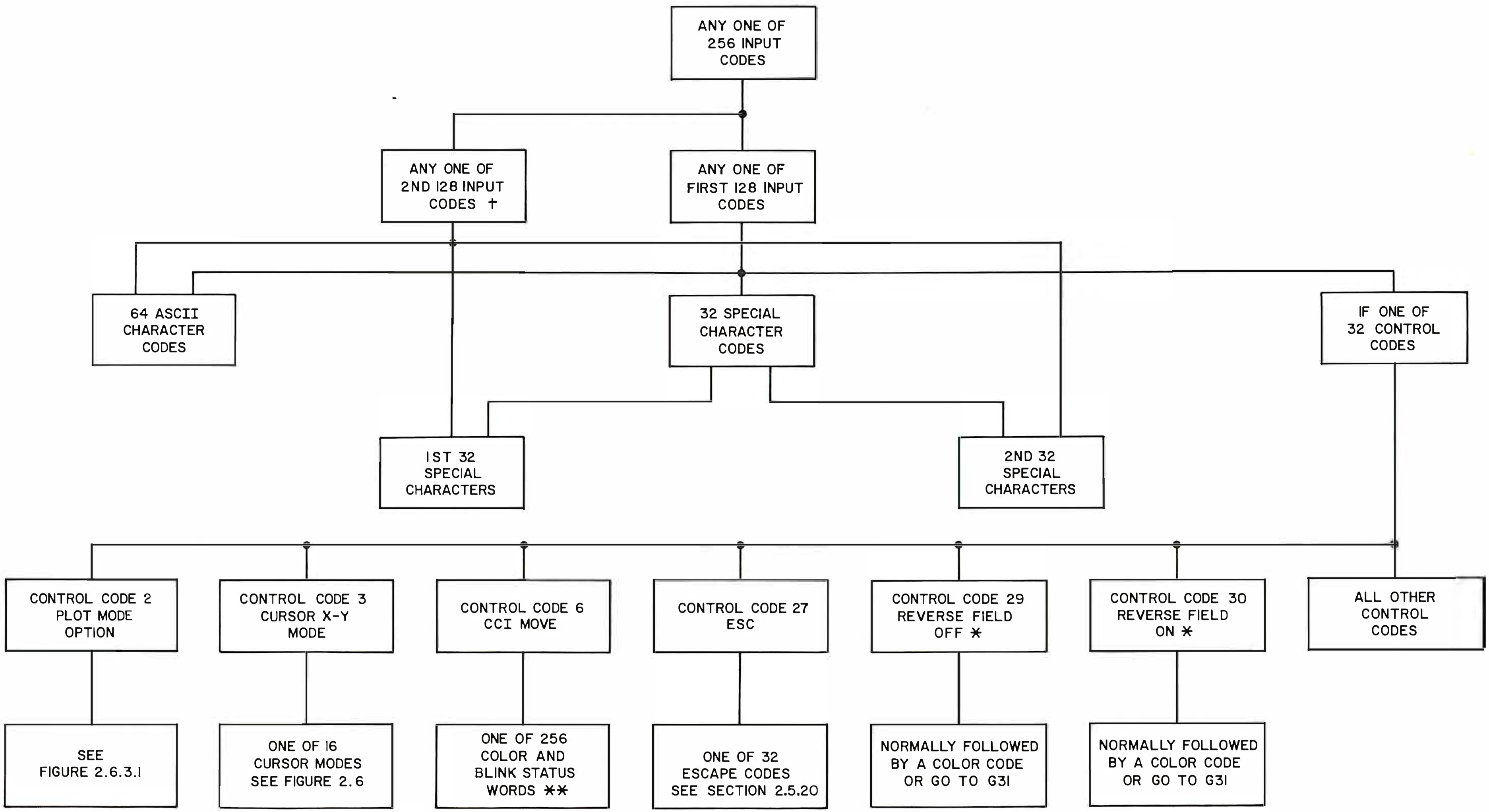
* COLUMNS 6 AND 7 WILL BE TRANSLATED TO COLUMNS 0 AND 1 RESPECTIVELY IN THE CRT REFRESH RAM IF THE FLAG ON HAS BEEN SET BEFORE ENTERING THESE CODES. THEY WILL THEN APPEAR AS THE SECOND GROUP OF 64 CHARACTERS IF THAT OPTION IS SUPPLIED.

INTECOLOR 8001 CODE SET



CHARACTER PLACEMENT GIVES LEFT & BOTTOM BOARDERS WITHIN THE 6x8 MATRIX.





† THIS IS VALID ONLY IF THE A7 BIT IS NOT MASKED OFF AT CUSTOMER REQUEST

* THIS CODE DOES NOT REQUIRE ANY FOLLOW-ON CODE

** SEE SECTION 2.6.7 FOR DEFINITION OF THE 8 BITS OF THE STATUS WORD

INPUT CODE FLOW DIAGRAM

Delay Times are in Milliseconds

Mode	Normal	High Speed Option
Blind Cursor Character Store	.278	.231
Most Control Codes	.46	.40
Erase Line	1.45	1.2
Erase Page	16 (30)*	14.1 (27)*
Visible Cursor Character Store Left-Right	.51	.430
2X Char	.59	.50
down @ 45^{o}	.75	.63
Insert 80 Characters	4.82	4.0
Delete 80 Characters	4.34	3.6
X,Y Point Plot	.40, .63	.33, .53
XY Increment 2 points	1.2	1.0
100 Element X Bar Graph	5.45	4.53
100 Element Y Bar Graph	3.28	2.73
100 Element Vector	34	28.3

*48L Delay time in ( )

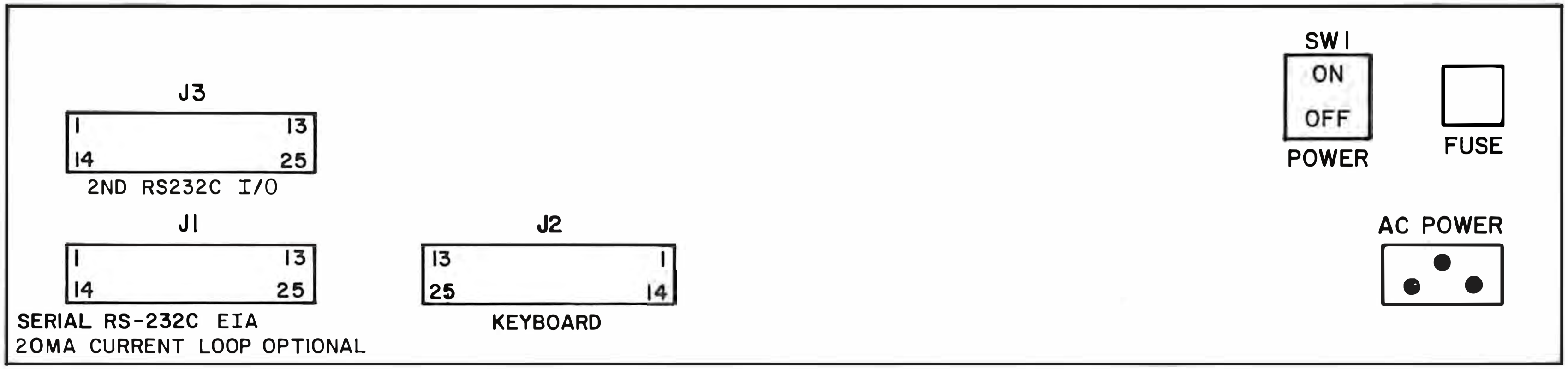
STANDARD INTECOLOR$_R$ 8001

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	1	RED FOREGROUND
0	0	0	0	0	0	1	0	GREEN FOREGROUND
0	0	0	0	0	1	0	0	BLUE FOREGROUND
0	1	0	0	0	0	0	0	FOREGROUND BLINK
1	0	0	0	0	0	0	0	PLOT CHARACTER

WITH BACKGROUND COLOR OPTION

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	1	RED FOREGROUND
0	0	0	0	0	0	1	0	GREEN FOREGROUND
0	0	0	0	0	1	0	0	BLUE FOREGROUND
0	0	0	0	1	0	0	0	RED BACKGROUND
0	0	0	1	0	0	0	0	GREEN BACKGROUND
0	0	1	0	0	0	0	0	BLUE BACKGROUND
0	1	0	0	0	0	0	0	FOREGROUND BLINK
1	0	0	0	0	0	0	0	PLOT CHARACTER

The above codes may be "ORed" for composite functions



NOTES: (1) J1-SERIAL RS-232C EIA AND CURRENT LOOP ARE NOT SIMULTANEOUSLY AVAILABLE IN RECEIVE MODE. PIN NUMBERS ARE FOR STANDARD EIA RS-232C, 25 PIN, CHASSIS MOUNT PLUG.

(2) MATING PLUGS & RECEPTACLES FOR REAR CHASSIS CONNECTORS:

J1,J3 (EIA, 25 PIN RECEPT.)	- AMP 205207-1	OR EQUIVALENT -	ISC P.N. 600040
J2 (25 PIN PLUG)	- AMP 205208-1	OR EQUIVALENT -	ISC P.N. 600052
AC POWER CORD	- BELDEN 17258B	OR EQUIVALENT -	ISC P.N. 110036
CRIMP PINS FOR PLUG CONNECTORS	- AMP 205201-5	OR EQUIVALENT -	ISC P.N. 600044
CRIMP SOCKETS FOR RECEPTACLE CONNECTORS	- AMP 205202-4	OR EQUIVALENT -	ISC P.N. 600046

( ) ALL CHASSIS MOUNT CONNECTOR PIN AND SOCKET NUMBERS ARE SHOWN AS VIEWED FROM THE REAR OF THE INTECOLOR® 8001 CHASSIS.

J1 AND J2 PIN ASSIGNMENTS

1. AA	1. AA	10. IN 0A Key	10. IN 0C
2. BA	2. BA	9. IN 1A Data	9. IN 1C
3. BB	3. BB	21. IN 2A Bits	21. IN 2C
4. CA	4. 470 ohms to +12V	8. IN 3A 1-4	8. IN 3C
5. CB	5.	20. IN 4A Control	20. IN 4C
6.	6.	7. IN 5A Shift	7. IN 5C
7. AB	7. AB	19. IN 6A Key Data B5	19. IN 6C
8.	8.	6. IN 7A Key Data B6	6. IN 7C
9. RX Response Control	9. RX Response Control	12. IN 4B Not	2. OUT $\overline{0}$C
10. RX Responce Control	10. RX Response Control	23. IN 5B Used	14. OUT $\overline{1}$C
11.	11. CLR+	11. IN 6B	3. OUT $\overline{2}$C
12. TTL TX	12. TTL TX	22. IN 7B Key Trigger	15. OUT $\overline{3}$C
13.	13. TX Isolator input	2. OUT $\overline{0}$A RX ACK	4. OUT $\overline{4}$C
14.	14.	14. OUT $\overline{1}$A	16. OUT $\overline{5}$C
15.	15.	3. OUT $\overline{2}$A	5. OUT $\overline{6}$C
16.	16.	15. OUT $\overline{3}$A	17. OUT $\overline{7}$C
17.	17.	4. OUT $\overline{4}$A	13. CPU RESET
18.	18. CLR-	16. OUT $\overline{5}$A	24. 2nd RS232 TX
19.	19.	5. OUT $\overline{6}$A Bell	11. 2nd TTL TX
20. CD	20. 470 ohms to +12V	17. OUT $\overline{7}$A -Key ACK	22. 2nd RS232 RX
21.	21. CLT+	13. CPU RESET	12. +12V
22.	22.	1. SN -Key Inturr.	23. -12V
23.	23.	25. +5V	25. +5V
24.	23.	18. GND	18. GND
25.	25. CLT-		1. SN - EXT Inturr.
	An external jumper is required from pin 12 to pin 13. A 2.2K ohm register is required from pin 3 to pin 4.		
STANDARD TTY ElA RS2326	OPTIONAL 20MA Current Loop		
J1 SERIAL INPUT/OUTPUT		J2 KEYBOARD	J3 OPTIONAL: PARALLEL INPUT/OUTPUT AND 2nd PS232C

I/O Connector Layout

APPENDIX B

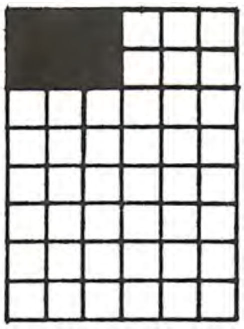
PLOT MODE
OR (2)
CONTROL CODE B
SETS PLOFL = 2

BINARY CODE =	255	254	253	252	251	250	249	248
TWO'S COMPLEMENT=	(-1)	(-2)	(-3)	(-4)	(-5)	(-6)	(-7)	(-8)
SETS PLOFL TO =	0	1	2	3	4	5	6	7
FUNCTION =	PLOT ESCAPE	CHARACTER PLOT	X-Y POINT PLOT	X-Y POINT PLOT	INCREMENTAL X-Y PLOT	X BAR GRAPH	X BAR GRAPH	X BAR GRAPH
NEXT WORD =	N/A	PLOT CHARACTER	X VALUE	Y VALUE	ΔX_1 ΔY_1 ΔX_2 ΔY_2	Xo VALUE	Y VALUE	X MAX VALUE
RANGE FROM =	N/A	0	0	0	0	0	0	0
TO =	N/A	254	159	100 [191]	239	159	100 [191]	159
CHANGE PLOFL TO =	N/A	1 (SAME)	3	2	4 (SAME)	6	7	8

BINARY CODE =	247	246	245	244	243	242	241	240
TWO'S COMPLEMENT=	(-9)	(-10)	(-11)	(-12)	(-13)	(-14)	(-15)	(-16)
SETS PLOFL TO =	8	9	10	11	12	13	14	15
FUNCTION =	X INCREMENTAL BAR GRAPH	Y BAR GRAPH	Y BAR GRAPH	Y BAR GRAPH	Y INCREMENTAL BAR GRAPH	VECTOR	VECTOR'	INCREMENTAL X-Y VECTOR
NEXT WORD =	X_1 Y_1 X_2 Y_2	Yo VALUE	X VALUE	Y MAX VALUE	ΔX_1 ΔY_1 ΔX_2 ΔY_2	Xo VALUE	Yo VALUE	ΔX_1 ΔY_1 ΔX_0 ΔY_0
RANGE FROM =	0	0	0	0	0	0	0	0
TO =	239	100 [191]	159	100 [191]	239	259	191	239
CHANGE PLOFL TO =	8 (SAME)	10	11	10	12 (SAME)	14	2	15

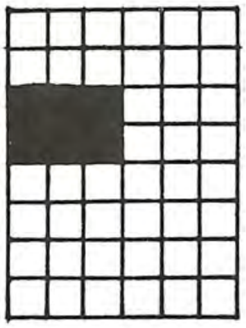
B-1

PLOT MODE FUNCTIONS



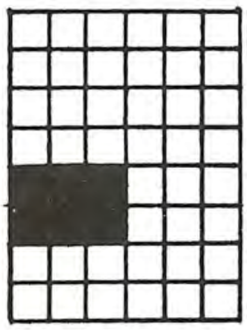
01 HEX

0 0 0 0 0 0 0 1



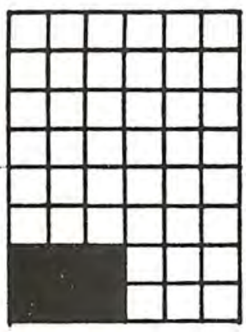
02 HEX

0 0 0 0 0 0 1 0



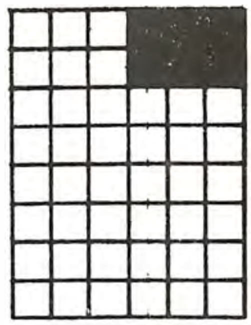
04 HEX

0 0 0 0 0 1 0 0



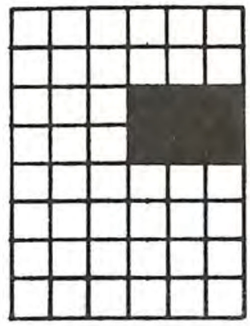
08 HEX

0 0 0 0 1 0 0 0



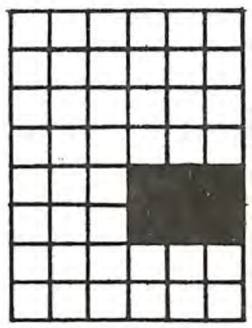
10 HEX

0 0 0 1 0 0 0 0



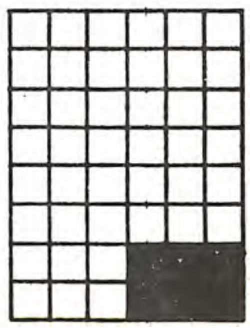
20 HEX

0 0 1 0 0 0 0 0



40 HEX

0 1 0 0 0 0 0 0

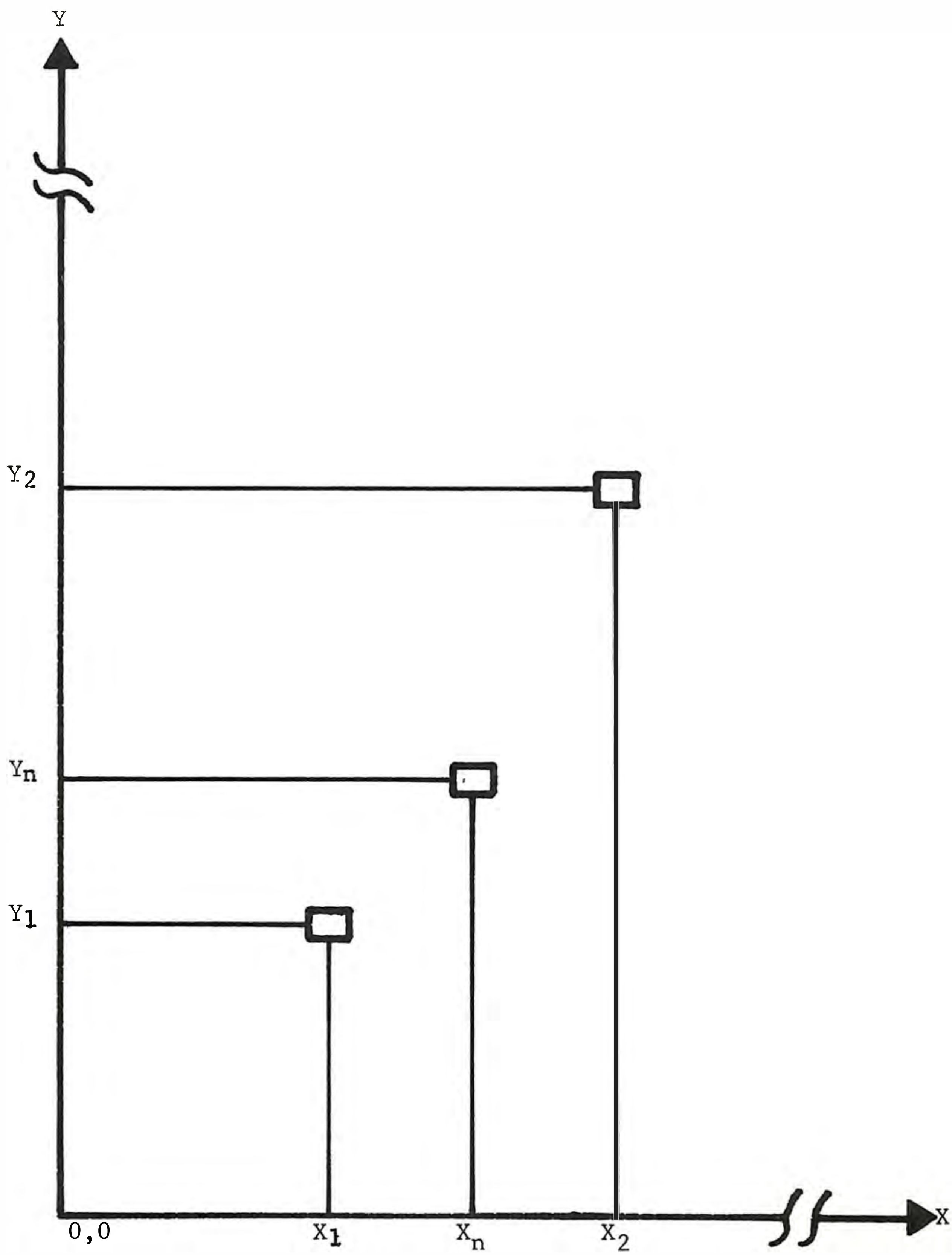


80 HEX

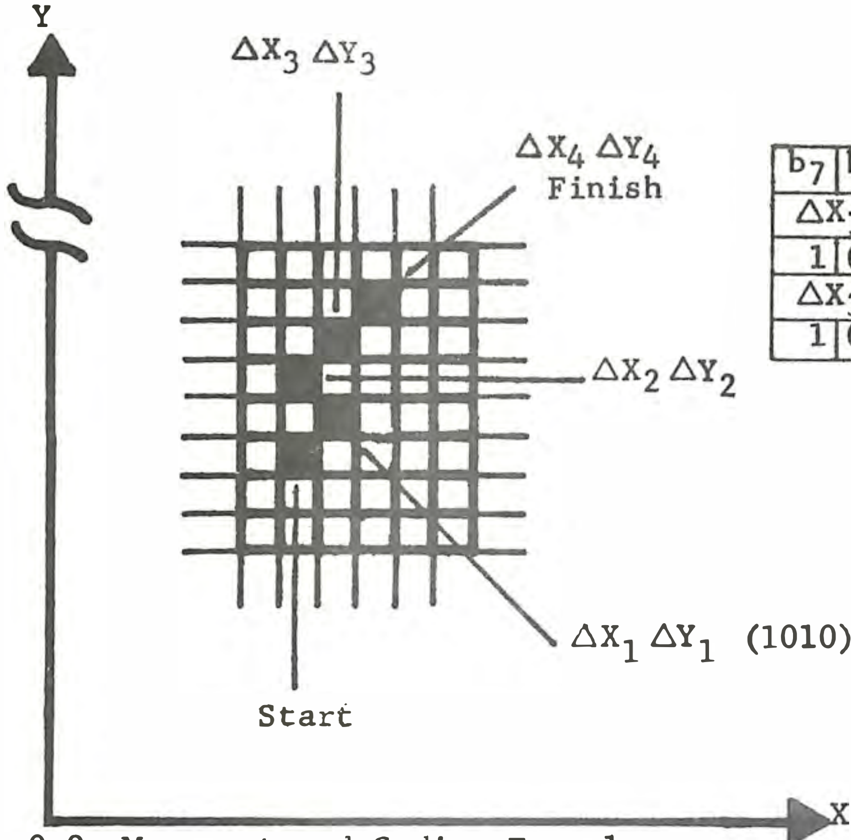
1 0 0 0 0 0 0 0

Note: Each of the above codes may be "ORed" for composite symbols.

PLOT MODE CHARACTERS AND CODES

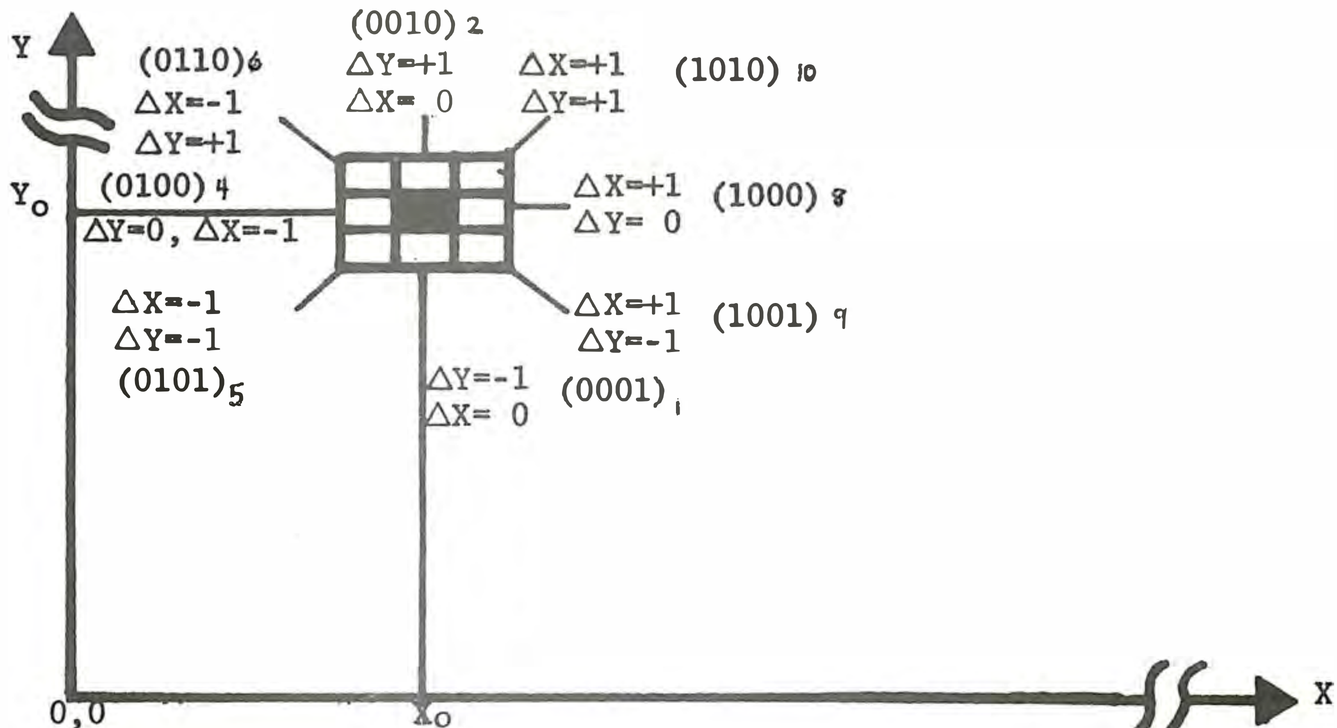


X POINT PLOT AND Y POINT PLOT



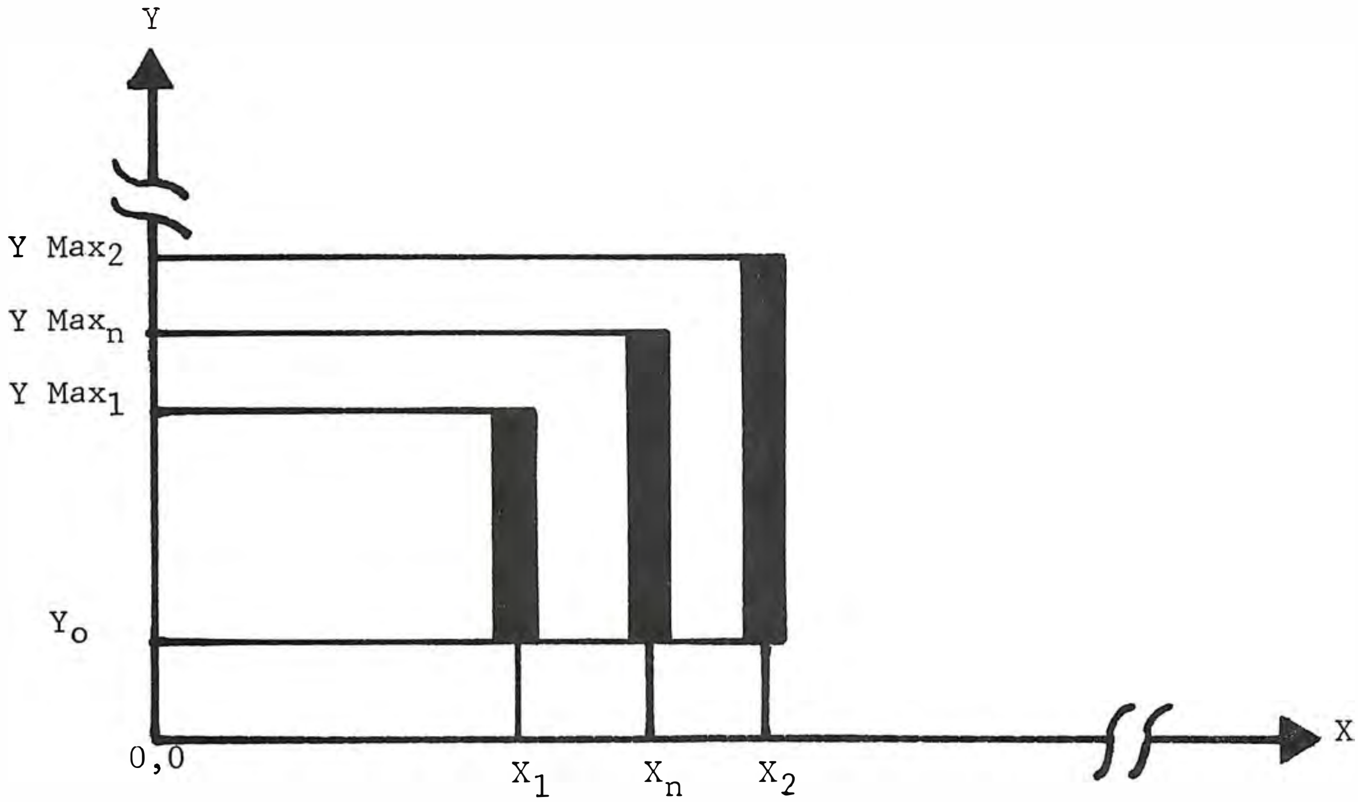
Movement and Coding Example

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
ΔX_1		ΔY_1		ΔX_2		ΔY_2	
1	0	1	0	0	1	1	0
ΔX_3		ΔY_3		ΔX_4		ΔY_4	
1	0	1	0	1	0	1	0

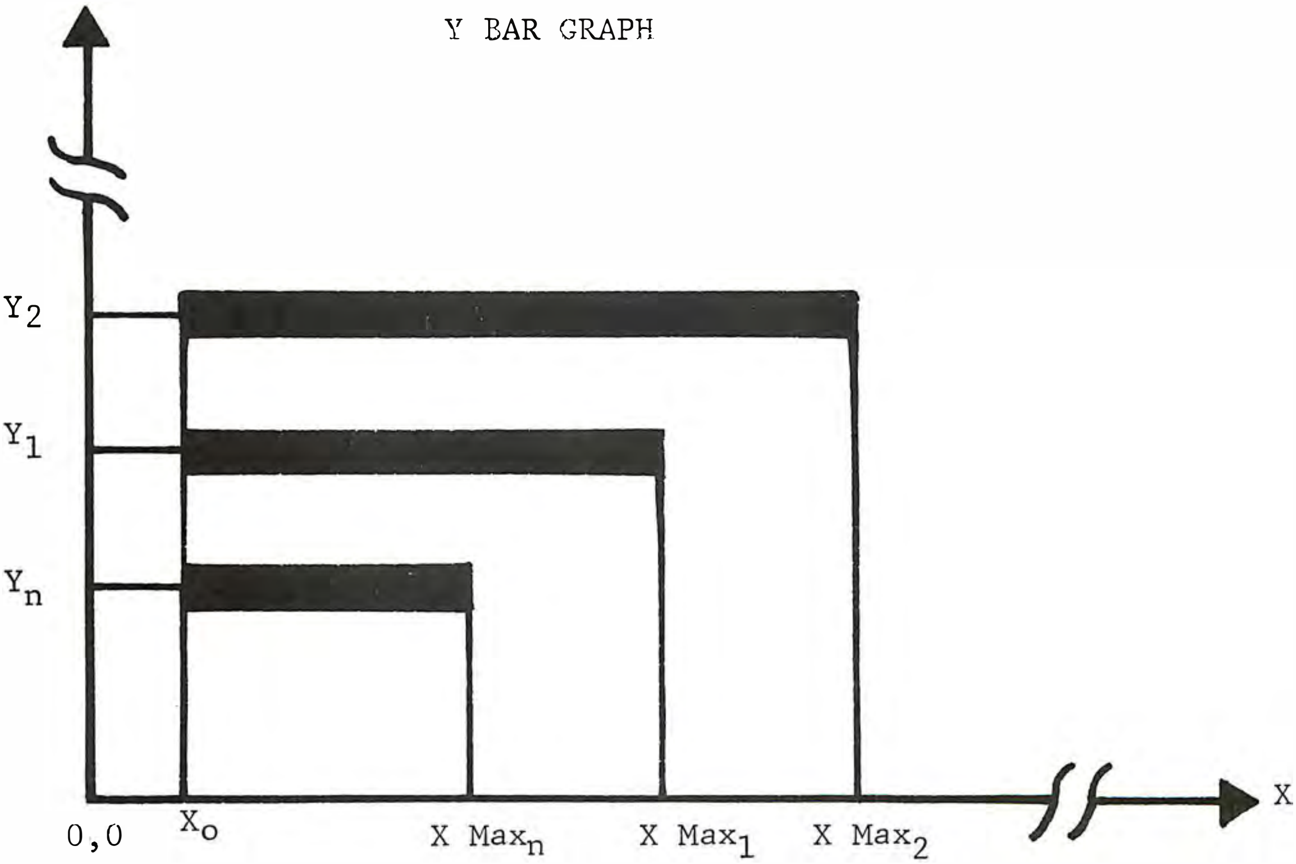


Movement Possibilities

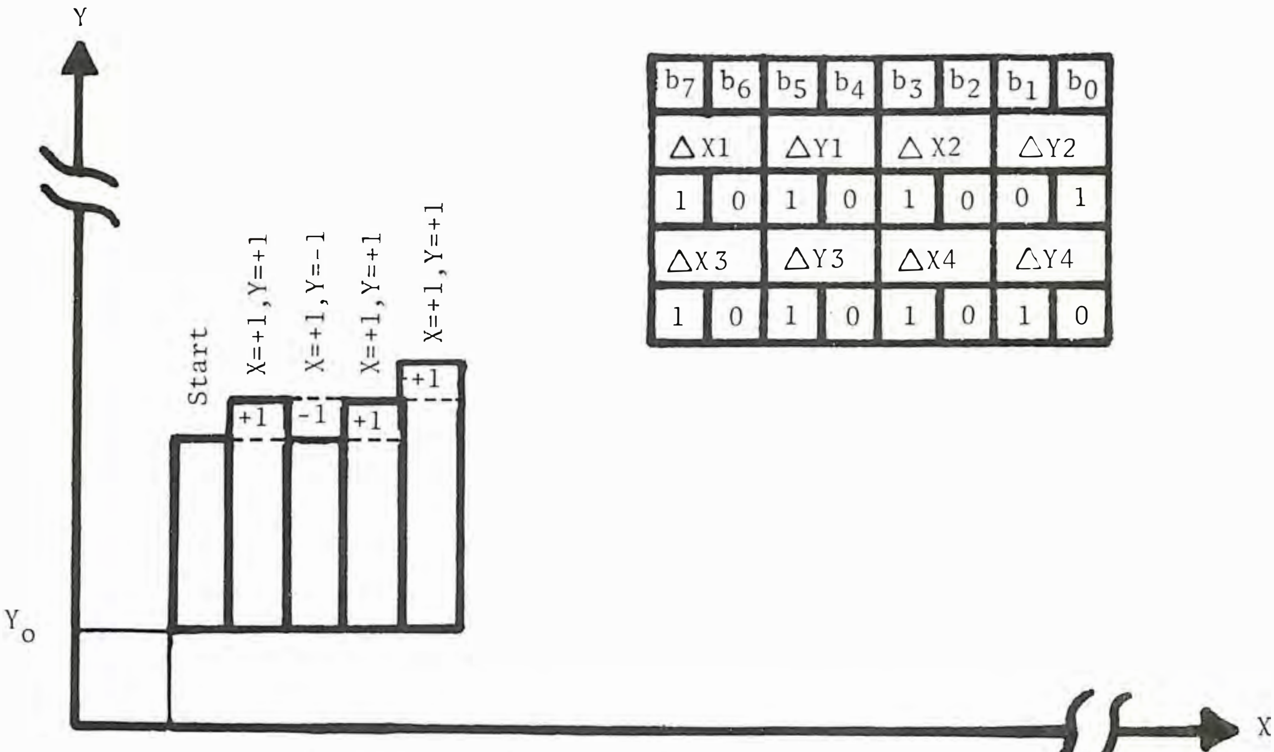
XY INCREMENTAL POINT PLOT MOVEMENTS



Y BAR GRAPH

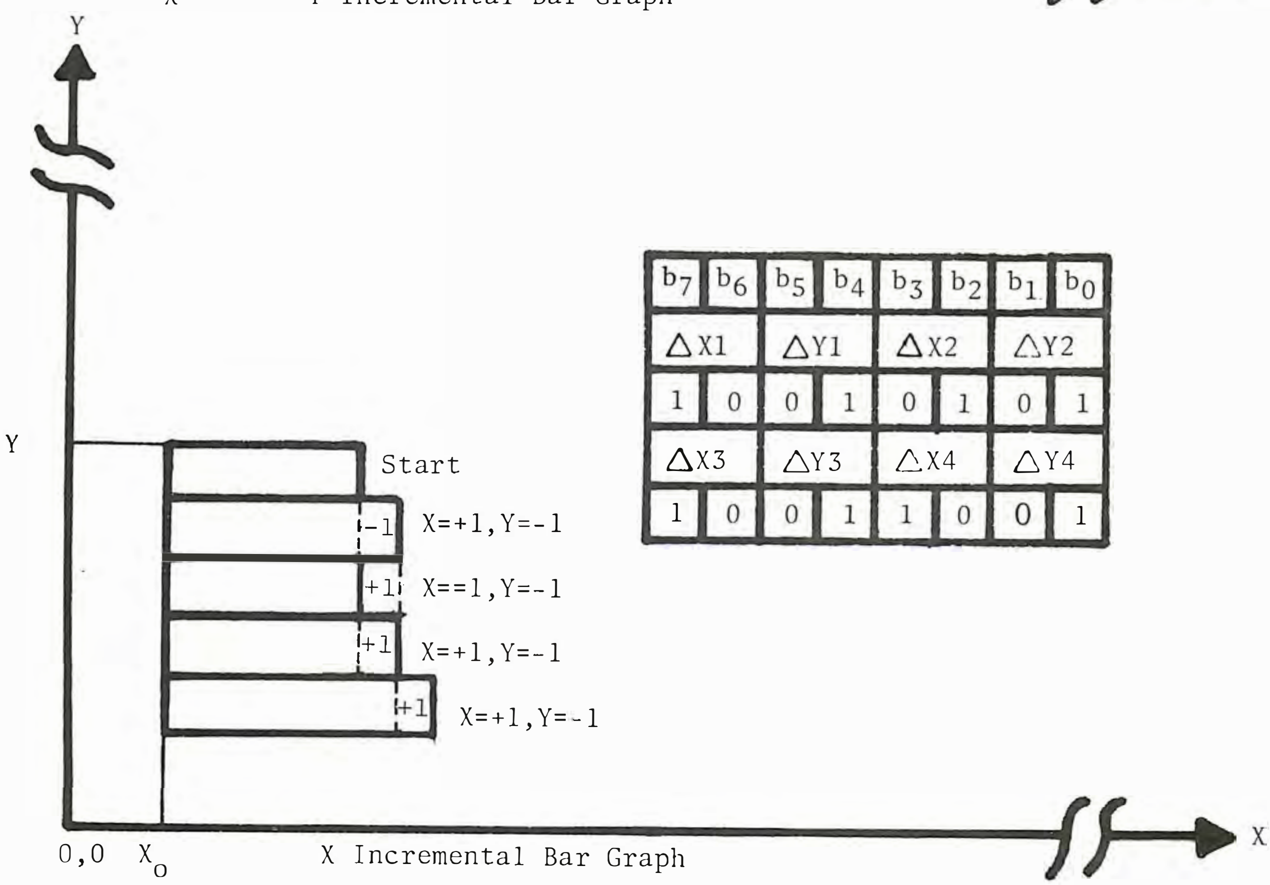


X AND Y BAR GRAPH MODES



b$_7$	b$_6$	b$_5$	b$_4$	b$_3$	b$_2$	b$_1$	b$_0$
△X1		△Y1		△X2		△Y2	
1	0	1	0	1	0	0	1
△X3		△Y3		△X4		△Y4	
1	0	1	0	1	0	1	0

Y Incremental Bar Graph

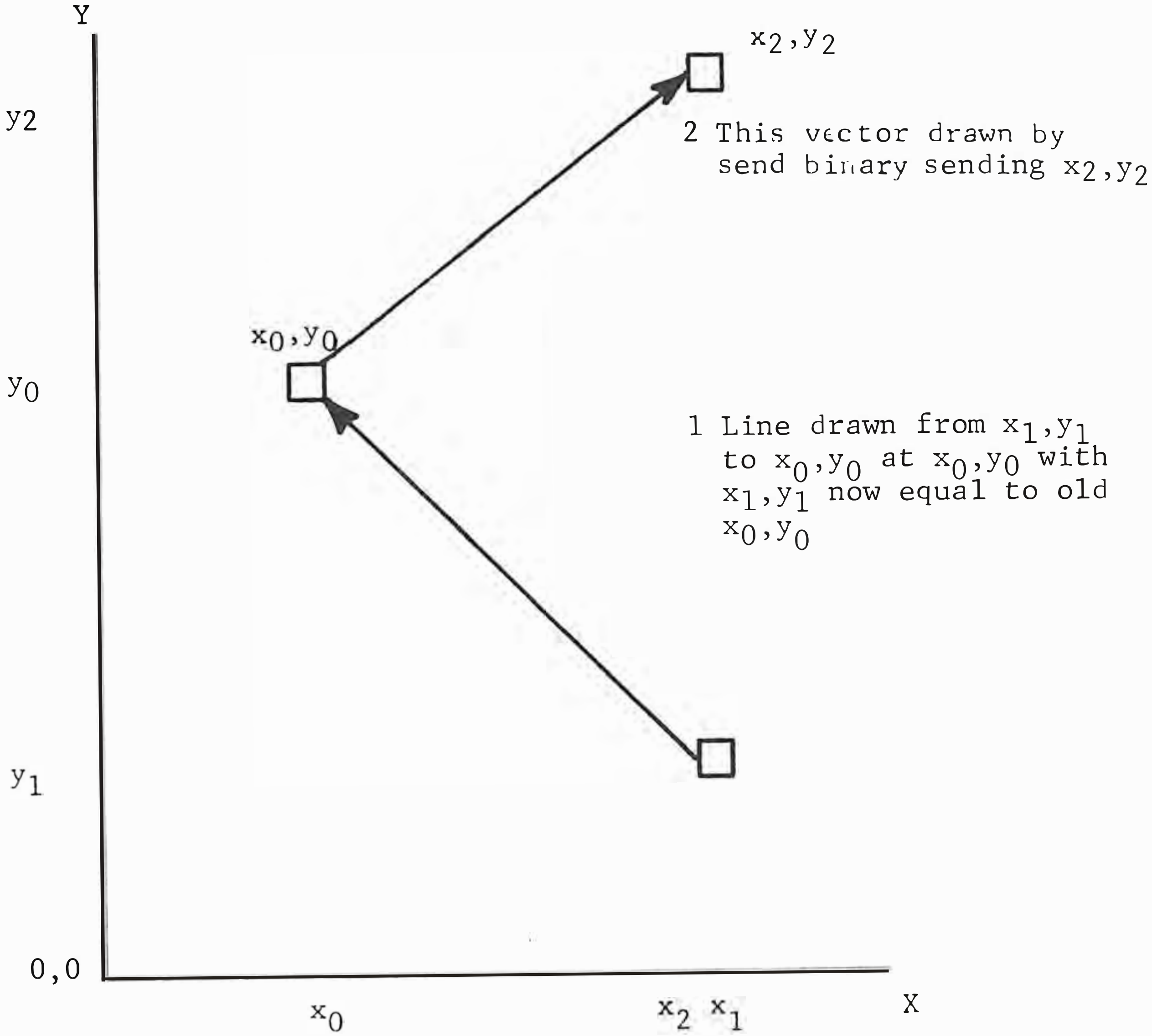


b$_7$	b$_6$	b$_5$	b$_4$	b$_3$	b$_2$	b$_1$	b$_0$
△X1		△Y1		△X2		△Y2	
1	0	0	1	0	1	0	1
△X3		△Y3		△X4		△Y4	
1	0	0	1	1	0	0	1

X Incremental Bar Graph

Appendix B6

X INCREMENTAL BAR GRAPH, Y INCREMENTAL BAR GRAPH



X_0 Y_0 Vector Plot Mode

APPENDIX C

Appendix C

TMS 5501 Multifunction Input/Output Controller

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TMS 5501 MULTIFUNCTION INPUT/OUTPUT CONTROLLER

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 5501 is a multifunction input/output circuit for use with TI's TMS 8080 CPU. It is fabricated with the same N-channel silicon-gate process as the TMS 8080 and has compatible timing, signal levels, and power supply requirements. The TMS 5501 provides a TMS 8080 microprocessor system with an asynchronous communications interface, data I/O buffers, interrupt control logic, and interval timers.

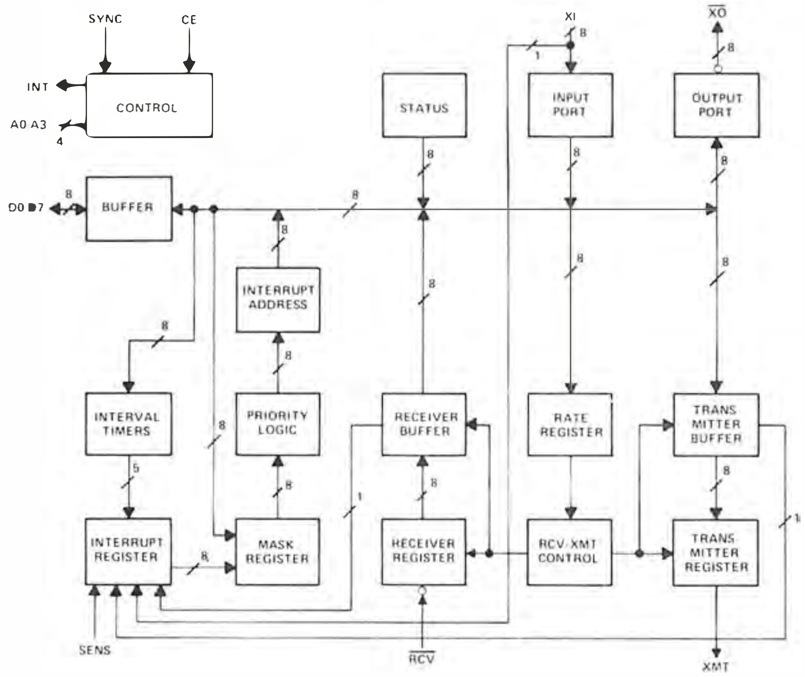


FIGURE 1—TMS 5501 BLOCK DIAGRAM

The I/O section of the TMS 5501 contains an eight-bit parallel input port and a separate eight-bit parallel output port with storage register. Five programmable interval timers provide time intervals from 64 μs to 16.32 ms.

The interrupt system allows the processor to effectively communicate with the interval timers, external signals, and the communications interface by providing TMS 8080-compatible interrupt logic with masking capability.

Data transfers between the TMS 5501 and the CPU are carried by the data bus and controlled by the interrupt, chip enable, sync, and address lines. The TMS 8080 uses four of its memory-address lines to select one of 14 commands to which the TMS 5501 will respond. These commands allow the CPU to:

---- read the receiver buffer
---- read the input port
---- read the interrupt address
---- read TMS 5501 status
---- issue discrete commands
---- load baud rate register
---- load the transmitter buffer
---- load the output port
---- load the mask register
---- load an interval timer

The commands are generated by executing memory referencing instructions such as MOV (register to memory) with the memory address being the TMS 5501 command. This provides a high degree of flexibility for I/O operations.by letting the systems programmer use a variety of instructions.

1.2 SUMMARY OF OPERATION

Addressing the TMS 5501

A convenient method for addressing the TMS 5501 is to tie the chip enable input to the highest order address line of the CPU's 16-bit address bus and the four TMS 5501 address inputs to the four lowest order bits of the bus. This, of course, limits the system to 32,768 words of memory but in many applications the full 65,536 word memory addressing capability of the TMS 8080 is not required.

Communications Functions

The communications section of the TMS 5501 is an asynchronous transmitter and receiver for serial communications and provides the following functions:

Programmable baud rate — A CPU command selects a baud rate of 110, 150, 300, 1200, 2400, 4800, or 9600 baud.

Incoming character detection — The receiver detects the start and stop bits of an incoming character and places the character in the receive buffer.

Character transmission — The transmitter generates start and stop bits for a character received from the CPU and shifts it out.

Status and command signals — Via the data bus, the TMS 5501 signals the status of: framing error and overrun error flags; data in the receiver and transmitter buffers; start and data bit detectors; and end-of-transmission (break) signals from external equipment. It also issues break signals to external equipment.

Data Interface

The TMS 5501 moves data between the CPU and external devices through its internal data bus, input port, and output port. When data is present on the bus that is to be sent to an external device, a Load Output Port (LOP) command from the CPU puts the data on the $\overline{\text{XO}}$ pins of the TMS 5501 by latching it in the output port. The data remains in the port until another LOP command is received. When the CPU requires data that is present on the External Input (XI) lines, it issues a command that gates the data onto the internal data bus of the TMS 5501 and consequently onto the CPU's data bus at the correct time during the CPU cycles.

Interval Timers

To start a countdown by any of the five interval timers, the program selects the particular timer by an address to the TMS 5501 and loads the required interval into the timer via the data bus. Loading the timer activates it and it counts down in increments of 64 microseconds. The 8-bit counters provide intervals that vary in duration from 64 to 16,320 microseconds. Much longer intervals can be generated by cascading the timers through software. When a timer reaches zero, it generates an interrupt that typically will be used to point to a subroutine that performs a servicing function such as polling a peripheral or scanning a keyboard. Loading an interval value of zero causes an immediate interrupt. A new value loaded while the interval timer is counting overrides the previous value and the interval timer starts counting down the new interval. When an interval timer reaches zero it remains inactive until a new interval is loaded.

Servicing Interrupts

The TMS 5501 provides a TMS 8080 system with several interrupt control functions by receiving external interrupt signals, generating interrupt signals, masking out undersired interrupts, establishing the priority of interrupts, and generating RST instructions for the TMS 8080. An external interrupt is received on pin 22, SENS. An additional external interrupt can be received on pin 32, XI7, if selected by a discrete command from the TMS 8080 (See Figure 4). The TMS 5501 generates an interrupt when any of the five interval timers count to zero. Interrupts are also generated when the receiver buffer is loaded and when the transmitter buffer is empty.

When an interrupt signal is received by the interrupt register from a particular source, a corresponding bit is set and gated to the mask register. A pattern will have previously been set in the mask register by a load-mask-register command from the TMS 8080. This pattern determines which interrupts will pass through to the priority logic. The priority logic allows an interrupt to generate an RST instruction to the TMS 8080 only if there is no higher priority interrupt that has not been accepted by the TMS 8080. The TMS 5501 prioritizes interrupts in the order shown below:

1st — Interval Timer #1
2nd — Interval Timer #2
3rd — External Sensor
4th — Interval Timer #3
5th — Receiver Buffer Loaded
6th — Transmitter Buffer Emptied
7th — Interval Timer #4
8th — Interval Timer #5 or an External Input (XI 7)

The highest priority interrupt passes through to the interrupt address logic, which generates the RST instruction to be read by the TMS 8080. See Table 3 for relationship of interrupt sources to RST instructions and Figures 6 and 8 for timing relationships.

The TMS 5501 provides two methods of servicing interrupts; an interrupt-driven system or a polled-interrupt system. In an interrupt-driven system, the INT signal of the TMS 5501 is tied to the INT input of the TMS 8080. The sequence of events will be: (1) The TMS 5501 receives (or generates) an interrupt signal and readies the appropriate RST instruction. (2) The TMS 5501 INT output, tied to the TMS 8080 INT input, goes high signaling the TMS 8080 that an interrupt has occured. (3) If the TMS 8080 is enabled to accept interrupts, it sets the INTA (interrupt acknowledge) status bit high at SYNC time of the next machine cycle. (4) If the TMS 5501 has previously received an interrupt-acknowledge-enable command from the CPU (see Bit 3, Paragraph 2.2.5), the RST instruction is transferred to the data bus.

In a polled-interrupt system, INT is not used and the sequence of events will be: (1) The TMS 5501 receives (or generates) an interrupt and readies the RST instruction. (2) The TMS 5501 interrupt-pending status bit (see Bit 5, Paragraph 2.2.4) is set high (the interrupt-pending status bit and the INT output go high simultaneously). (3) At the prescribed time, the TMS 8080 polls the TMS 5501 to see if an interrupt has occurred by issuing a read-TMS 5501-status command and reading the interrupt-pending bit. (4) If the bit is high, the TMS 8080 will then issue a read-interrupt-address command, which causes the TMS 5501 to transfer the RST instruction to the data bus as data for the instruction being executed by the TMS 8080.

1.3 APPLICATIONS

Communications Terminals

The functions of the TMS 5501 make it particularly useful in TMS 8080-based communications terminals and generally applicable in systems requiring periodic or random servicing of interrupts, generation of control signals to external devices, buffering of data, and transmission and reception of asynchronous serial data. As an example, a system configuration such as shown in Figure 2 can function as the controller for a terminal that governs employee entrance into a plant or security areas within a plant. Each terminal is identified by a central computer through ID switches. The central system supplies each terminal's RAM with up to 16 employee access categories applicable to that terminal. These categories are compared with an employee's badge character when he inserts his badge into the badge sensor. If a

match is not found, a reject light will be activated. If a match is found, the terminal will transmit the employee's badge number and access category to the central system, and a door unlock solenoid will be activated for 4 seconds. The central computer then may take the transmitted information and record it along with time and date of access.

The TMS 4700 is a 1024 x 8 ROM that contains the system program, and the TMS 4036 is a 64 x 8 RAM that serves as the stack for the TMS 8080 and storage for the access category information. TTL circuits control chip-enable information carried by the address bus. Signals from the CPU gate the address bits from the ROM, the RAM, or the TMS 5501 onto the data bus at the correct time in the CPU cycle. The clock generator consists of four TTL circuits along with a crystal, needed to maintain accurate serial data assembly and disassembly with the central computer.

The TMS 5501 handles the asynchronous serial communication between the TMS 8080 and the central system and gates data from the badge reader onto the data bus. It also gates control and status data from the TMS 8080 to the door lock and badge reader and controls the time that the door lock remains open. The TMS 5501 signals the TMS 8080 when the badge reader or the communication lines need service. The functions that the TMS 5501 is to perform are selected by an address from the TMS 8080 with the highest order address line tied to the TMS 5501 chip enable input and the four lowest order lines tied to the address inputs.

2. OPERATIONAL AND FUNCTIONAL DESCRIPTION

This detailed description of the TMS 5501 consists of:

INTERFACE SIGNALS – a definition of each of the circuit's external connections

COMMANDS – the address required to select each of the TMS 5501 commands and a description of the response to the command.

2.1 INTERFACE SIGNALS

The TMS 5501 communicates with the TMS 8080 via four address lines: a chip enable line, an eight-bit bidirectional data bus, an interrupt line, and a sync line. It communicates with system components other than the CPU via eight external inputs, eight external outputs, a serial receiver input, a serial transmitter output, and an external sensor input. Table 1 defines the TMS 5501 pin assignments and describes the function of each pin.

TABLE 1
TMS 5501 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	DESCRIPTION
		INPUTS
CE	18	Chip enable–When CE is low, the TMS 5501 address decoding is inhibited, which prevents execution of any of the TMS 5501 commands.
A3	17	Address bus–A3 through A0 are the lines that are addressed by the TMS 8080 to select a particular TMS 5501 function.
A2	16	
A1	15	
A0	14	
SYNC	19	Synchronizing signal–The SYNC signal is issued by the TMS 8080 and indicates the beginning of a machine cycle and availability of machine status. When the SYNC signal is active (high), the TMS 5501 will monitor the data bus bits D0 (interrupt acknowledge) and D1 ($\overline{\text{WO}}$, data output function).
$\overline{\text{RCV}}$	5	Receiver serial data input line–$\overline{\text{RCV}}$ must be held in the inactive (high) state when not receiving data. A transition from high to low will activate the receive circuitry.

TABLE 1 (continued)
TMS 5501 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	DESCRIPTION
		INPUTS
XI 0	39	External inputs—These eight external inputs are gated to the data bus when the read-external-inputs function is addressed. External input n is gated to data bus bit n without conversion.
XI 1	38	
XI 2	37	
XI 3	36	
XI 4	35	
XI 5	34	
XI 6	33	
XI 7	32	
SENS	22	External interrupt sensing – A transition from low to high at SENS sets a bit in the interrupt register, which, if enabled, generates an interrupt to the TMS 8080.
		OUTPUTS
$\overline{XO}$ 0	24	External outputs—These eight external outputs are driven by the complement of the output register; i.e., if output register bit n is loaded with a high (low) from data bus bit n by a load-output register command, the external output n will be a low (high). The external outputs change only when a load-output-register function is addressed.
$\overline{XO}$ 1	25	
$\overline{XO}$ 2	26	
$\overline{XO}$ 3	27	
$\overline{XO}$ 4	28	
$\overline{XO}$ 5	29	
$\overline{XO}$ 6	30	
$\overline{XO}$ 7	31	
XMT	40	Transmitter serial data output line—This line remains high when the TMS 5501 is not transmitting.
		DATA BUS INPUT/OUTPUT
D0	13	Data bus – Data transfers between the TMS 5501 and the TMS 8080 are made via the 8-bit bidirectional data bus. D0 is the LSB. D7 is the MSB.
D1	12	
D2	11	
D3	10	
D4	9	
D5	8	
D6	7	
D7	6	
INT	23	Interrupt—When active (high), the INT output indicates that at least one of the interrupt conditions has occurred and that its corresponding mask-register bit is set.
		POWER AND CLOCKS
V_{SS}	4	Ground reference
V_{BB}	1	Supply voltage (–5 V nominal)
V_{CC}	2	Supply voltage (5 V nominal)
V_{DD}	3	Supply voltage (12 V nominal)
$\phi 1$	20	Phase 1 clock
$\phi 2$	21	Phase 2 clock

2.2 TMS 5501 COMMANDS

The TMS 5501 operates as memory device for the TMS 8080. Functions are initiated via the TMS 8080 address bus and the TMS 5501 address inputs. Address decoding to determine the command function being issued is defined in Table 2.

TABLE 2
COMMAND ADDRESS DECODING
When Chip Enable Is High

A3	A2	A1	A0	COMMAND	FUNCTION	PARAGRAPH
L	L	L	L	Read receiver buffer	RBn → Dn	2.2.1
L	L	L	H	Read external inputs	XIn → Dn	2.2.2
L	L	H	L	Read interrupt address	RST → Dn	2.2.3
L	L	H	H	Read TMS 5501 status	(Status) → Dn	2.2.4
L	H	L	L	Issue discrete commands	See Figure 4	2.2.5
L	H	L	H	Load rate register	See Figure 4	2.2.6
L	H	H	L	Load transmitter buffer	Dn → TBn	2.2.7
L	H	H	H	Load output port	Dn → $\overline{XO}$n	2.2.8
H	L	L	L	Load mask register	Dn → MRn	2.2.9
H	L	L	H	Load interval timer 1	Dn → Timer 1	2.2.10
H	L	H	L	Load interval timer 2	Dn → Timer 2	2.2.10
H	L	H	H	Load interval timer 3	Dn → Timer 3	2.2.10
H	H	L	L	Load interval timer 4	Dn → Timer 4	2.2.10
H	H	L	H	Load interval timer 5	Dn → Timer 5	2.2.10
H	H	H	L	No function		
H	H	H	H	No function		

RBn Receiver buffer bit n
Dn Data bus I/O terminal n
XIn External input terminal n
RST 11 (IA_2) (IA_1) (IA_0) 1 1 1 (see Table 3)
TBn Transmit buffer bit n
$\overline{XO}$n Output register bit n
MRn Mask register bit n

TABLE 3
RST INSTRUCTIONS

DATA BUS BIT 0	1	2	3	4	5	6	7	INTERRUPT CAUSED BY
H	H	H	L	L	L	H	H	Interval Timer 1
H	H	H	H	L	L	H	H	Interval Timer 2
H	H	H	L	H	L	H	H	External Sensor
H	H	H	H	H	L	H	H	Interval Timer 3
H	H	H	L	L	H	H	H	Receiver Buffer
H	H	H	H	L	H	H	H	Transmitter Buffer
H	H	H	L	H	H	H	H	Interval Timer 4
H	H	H	H	H	H	H	H	Interval Timer 5 or X17

The following paragraphs define the functions of the TMS 5501 commands.

2.2.1 Read receiver buffer

Addressing the read-receiver-buffer function causes the receiver buffer contents to be transferred to the TMS 8080 and clears the receiver-buffer-loaded flag.

2.2.2 Read external input lines

Addressing the read-external-inputs function transfers the states of the eight external input lines to the TMS 8080.

2.2.3 Read interrupt address

Addressing the read interrupt address function transfers the current highest priority interrupt address onto the data bus as read data. After the read operation is completed, the corresponding bit in the interrupt register is reset.

If the read-interrupt-address function is addressed when there is no interrupt pending, a false interrupt address will be read. TMS 5501 status function should be addressed in order to determine whether or not an interrupt condition is pending.

2.2.4 Read TMS 5501 status

Addressing the read-TMS 5501-status function gates the various status conditions of the TMS 5501 onto the data bus. The status conditions, available as indicated in Figure 3, are described in the following paragraphs.

BIT:	7	6	5	4	3	2	1	0
	START BIT DETECT	FULL BIT DETECT	INTRPT PENDING	XMIT BUFFER EMPTY	RCV BUFFER LOADED	SERIAL RCVD	OVERRUN ERROR	FRAME ERROR

FIGURE 3—DATA BUS ASSIGNMENTS FOR TMS 5501 STATUS

Bit 0, framing error

A high in bit 0 indicates that a framing error was detected on the last character received (either one or both stop bits were in error). The framing error flag is updated at the end of each character. Bit 0 of the TMS 5501 status will remain high until the next valid character is received.

Bit 1, overrun error

A high in bit 1 indicates that a new character was loaded into the receiver buffer before a previous character was read out. The overrun error flag is cleared each time the read-I/O-status function is addressed or a reset command is issued.

Bit 2, serial received data

Bit 2 monitors the receiver serial data input line. This line is provided as a status input for use in detecting a break and for test purposes. Bit 2 is normally high when no data is being received.

Bit 3, receiver buffer loaded

A high in bit 3 indiciates that the receiver buffer is loaded with a new character. The receiver-buffer-loaded flag remains high until the read-receiver-buffer function is addressed (at which time the flag is cleared). The reset function also clears this flag.

Bit 4, transmitter buffer empty

A high in bit 4 indicates that the transmitter buffer register is empty and ready to accept a character. Note, however, that the serial transmitter register may be in the process of shifting out a character. The reset function sets the transmitter-buffer-empty flag high.

Bit 5, interrupt pending

A high in bit 5 indicates that one or more of the interrupt conditions has occured and the corresponding interrupt is enabled. This bit is the status of the interrupt signal INT.

Bit 6, full bit detected

A high in bit 6 indicates that the first data bit of a receive-data character has been detected. This bit remains high until the entire character has been received or until a reset is issued and is provided for test purposes.

Bit 7, start bit detected

A high in bit 7 indicates that the start bit of an incoming data character has been detected. This bit remains high until the entire character has been received or until a reset is issued and is provided for test purposes.

2.2.5 Issue discrete commands

Addressing the discrete command function causes the TMS 5501 to interpret the data bus information according to the following descriptions. See Figure 4 for the discrete command format. Bits 1 through 5 are latched until a different discrete command is received.

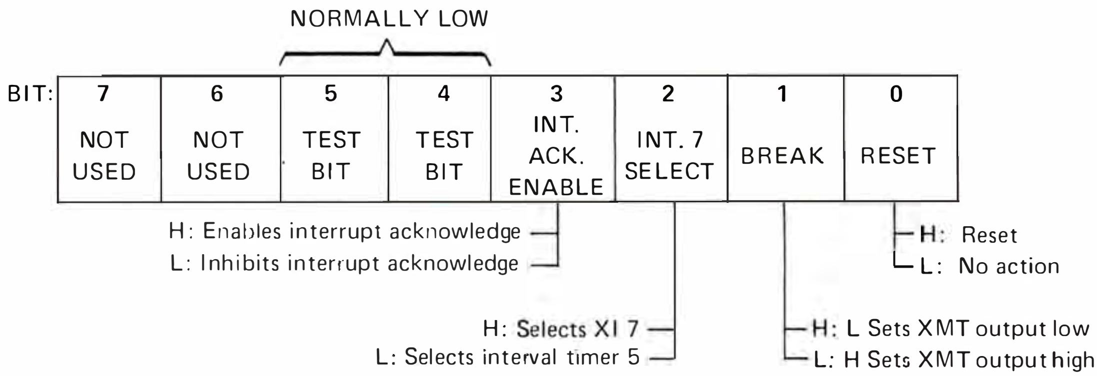


FIGURE 4—DISCRETE COMMAND FORMAT

Bit 0, reset

A high in bit 0 will cause the following:

1) The receiver buffer and register are cleared to the search mode including the receiver-buffer-loaded flag, the start-bit-detected flag, the full-bit-detected flag, and the overrun-error flag. The receiver buffer is not cleared and will contain the last character received.

2) The transmitter data output is set high (marking). The transmitter-buffer-empty flag is set high indicating that the transmitter buffer is ready to accept a character from the TMS 8080.

3) The interrupt register is cleared except for the bit corresponding to the transmitter buffer interrupt, which is set high.

4) The interval timers are inhibited.

A low in bit 0 causes no action. The reset function has no affect on the output port, the external inputs, interrupt acknowledge enable, the mask register, the rate register, the transmitter register, or the transmitter buffer.

Bit 1, break
A low in bit 1 causes the transmitter data output to be reset low (spacing).

If bit 0 and bit 1 are both high, the reset function will override.

Bit 2, interrupt 7 select
Interrupt 7 may be generated either by a low to high transition of external input 7 or by interval timer 5.

A high in bit 2 selects the interrupt 7 source to be the transition of external input 7. A low in bit 2 selects the interrupt 7 source to be interval timer 5.

Bit 3, interrupt acknowledge enable
The TMS 5501 decodes data bus (CPU status) bit 0 at SYNC of each machine cycle to determine if an interrupt acknowledge is being issued.

A high in bit 3 enables the TMS 5501 to accept the interrupt acknowledge decode. A low in bit 3 causes the TMS 5501 to ignore the interrupt acknowledge decode.

Bit 4 and bit 5 are used only during testing of the TMS 5501. For correct system operation both bits must be kept low.

Bit 6 and bit 7 are not used and can assume any value.

2.2.6 Load rate register

Addressing the load-rate-register function causes the TMS 5501 to load the rate register from the data bus and interpret the data bits (See Figure 5) as follows.

BIT:	7	6	5	4	3	2	1	0
	STOP BIT(s)	9600 baud	4800 baud	2400 baud	1200 baud	300 baud	150 baud	110 baud

Bit 7:
- H: One stop bit
- L: Two stop bits

FIGURE 5—DATA BUS ASSIGNMENTS FOR RATE COMMANDS

Bits 0 through 6, rate select

The rate select bits (bits 0 through 6) are mutually exclusive, i.e., only one bit may be high. A high in bits 0 through 6 will select the baud rate for both the transmitter and receiver circuitry as defined below and in Figure 5:

Bit 0	110 baud
Bit 1	150 baud
Bit 2	300 baud
Bit 3	1200 baud
Bit 4	2400 baud
Bit 5	4800 baud
Bit 6	9600 baud

If more than one bit is high, the highest rate indicated will result. If bits 0 through 6 are all low, both the receiver and the transmitter circuitry will be inhibited.

Bit 7, stop bits

Bit 7 determines whether one or two stop bits are to be used by both the transmitter and receiver circuitry. A high in bit 7 selects one stop bit. A low in bit 7 selects two stop bits.

2.2.7 Load transmitter buffer

Addressing the load-transmitter-buffer function transfers the state of the data bus into the transmitter buffer.

2.2.8 Load output port

Addressing the load-output-port function transfers the state of the data bus into the output port. The data is latched and remains on $\overline{XO\ 0}$ through $\overline{XO\ 7}$ as the complement of the data bus until new data is loaded.

2.2.9 Load mask register

Addressing the load-mask-register function loads the contents of the data bus into the mask register. A high in data bus bit n enables interrupt n. A low inhibits the corresponding interrupt.

2.2.10 Load timer n

Addressing the load-timer-n function loads the contents of the data bus into the appropriate interval timer. Time intervals of from 64 μs (data bus = LLLLLLLH) to 16,320 μs (data bus HHHHHHHH) are counted in 64-μs, steps. When the count of interval timer n reaches 0, the bit in the interrupt register that corresponds to timer n is set and an interrupt is generated. Loading all lows causes an interrupt immediately.

3. TMS 5501 ELECTRICAL AND MECHANICAL SPECIFICATIONS

3.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1) . . . . −0.3 V to 20 V
Supply voltage, V_{DD} (see Note 1 . . . . −0.3 V to 20 V
Supply voltage, V_{SS} (see Note 1) . . . . −0.3 V to 20 V
All input and output voltages (see Note 1) . . . . −0.3 V to 20 V
Continuous power dissipation . . . . 1.1 W
Operating free-air temperature range . . . . 0°C to 70°C
Storage temperature range . . . . −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the normally most negative supply voltage, V_{BB} (substrate). Throughout the remainder of this data sheet, voltage values are with respect to V_{SS} unless otherwise noted.

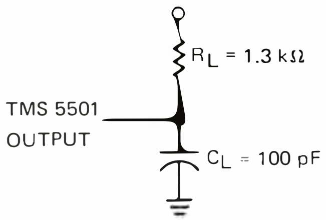
3.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−4.75	−5	−5.25	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	3.3		V_{CC}+1	V
High-level clock input voltage, $V_{IH(\phi)}$	V_{DD}−1		V_{DD}+1	V
Low-level input voltage, V_{IL} (all inputs except clocks) (see Note 2)	−1		0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$ (see Note 2)	−1		0.6	V
Operating free-air temperature, T_A	0		70	°C

NOTE 2: The algebraic convention where the most negative limit is designated as minimum is used in this specification for logic voltage levels only.

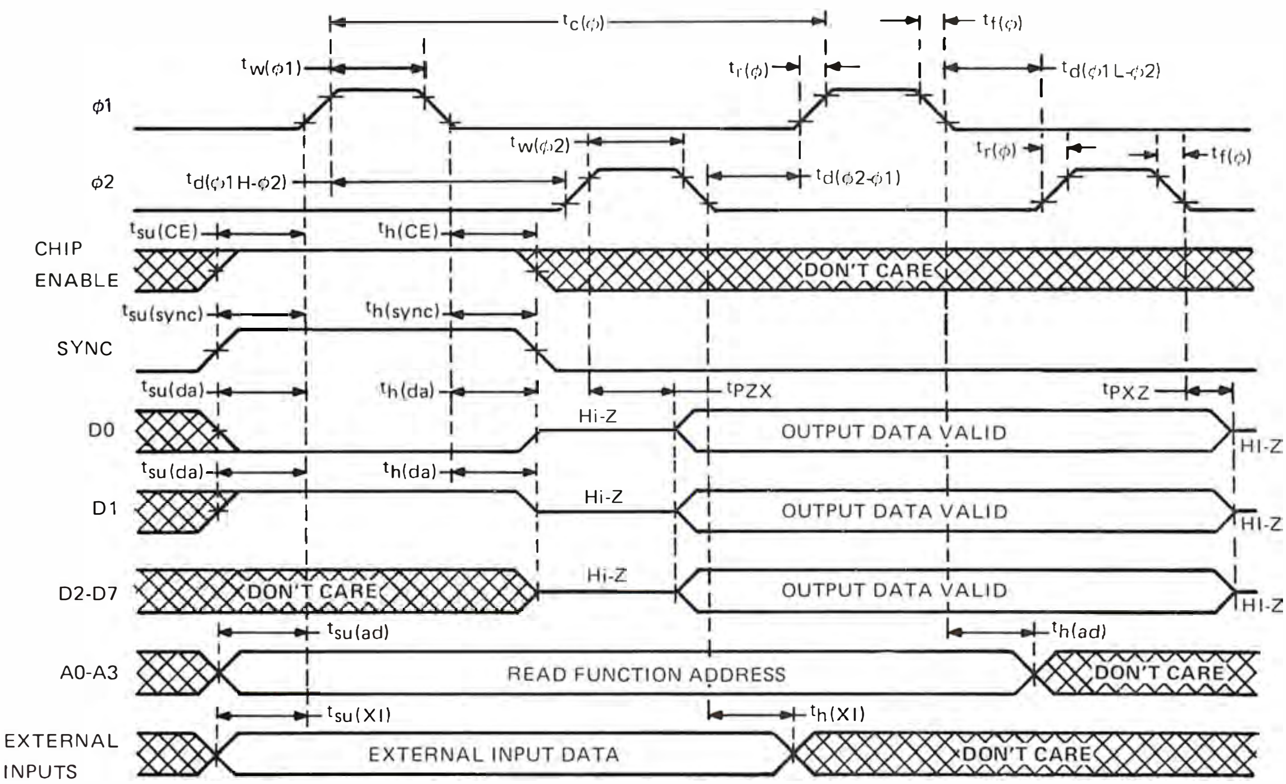
3.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 6 AND 7)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{PZX}	Data bus output enable time	C_L = 100 pF, R_L = 1.3 kΩ		200	ns
t_{PXZ}	Data bus output disable time to high-impedance state			180	ns
t_{PD}	External data output propagation delay time from $\phi 2$			200	ns



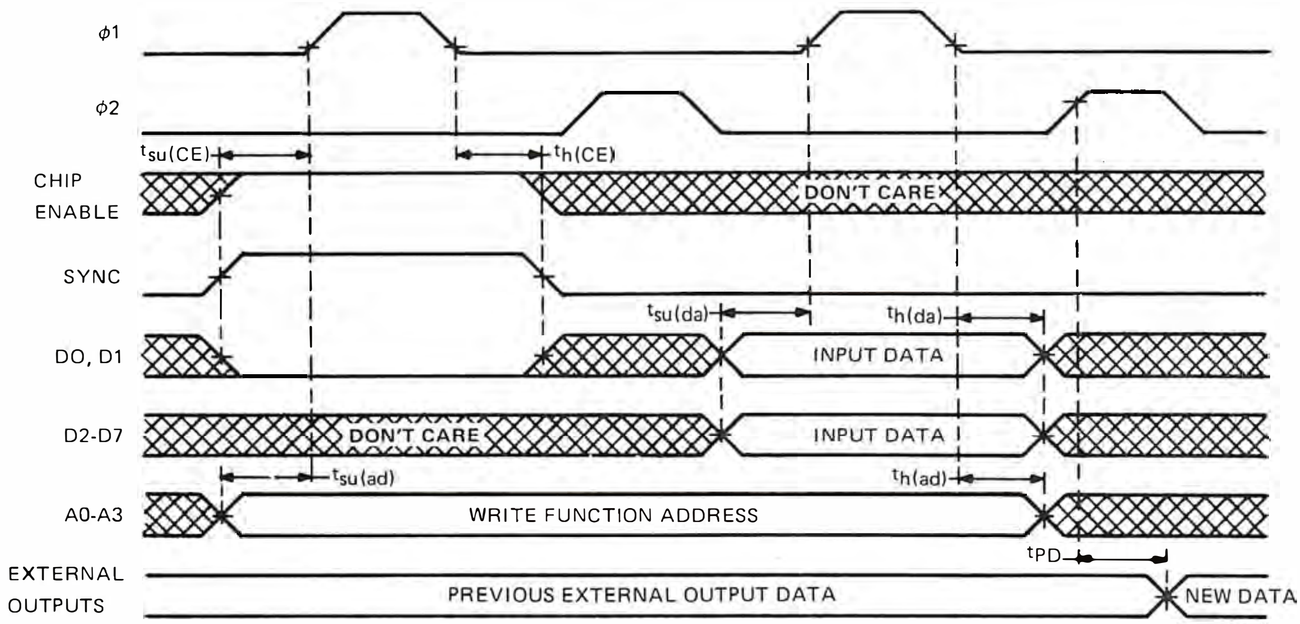
C_L includes probe and jig capacitance

LOAD CIRCUIT



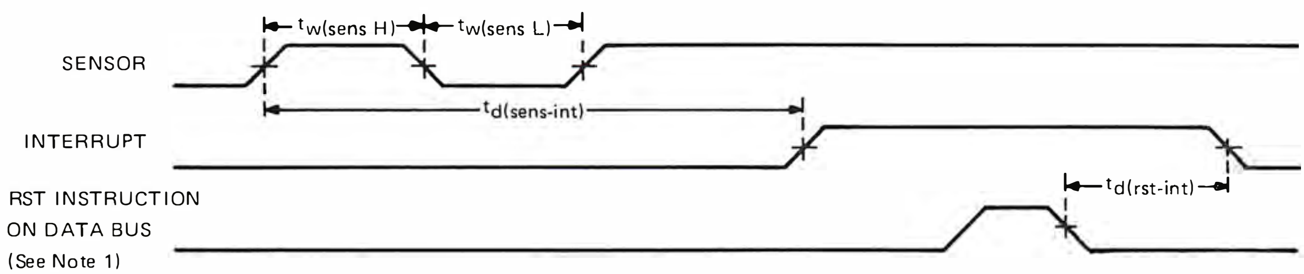
NOTE: For $\phi 1$ or $\phi 2$ inputs, high and low timing points are 90% and 10% of $V_{IH(\phi)}$. All other timing points are the 50% level.

FIGURE 6–READ CYCLE TIMING



NOTE: For $\phi 1$ and $\phi 2$ inputs, high and low timing points are 90% and 10% of $V_{IH(\phi)}$. All other timing points are the 50% level.

FIGURE 7—WRITE CYCLE TIMING



NOTES: 1. The RST instruction occurs during the output data valid time of tne read cycle.
2. All timing points are 50% of V_{IH}.

FIGURE 8—SENSOR/INTERRUPT TIMING

APPENDIX D

Appendix D

TMS 8080 Microprocessor

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TMS 8080 MICROPROCESSOR

1. ARCHITECTURE

1.1 INTRODUCTION

The TMS 8080 is an 8-bit parallel central processing unit (CPU) fabricated on a single chip using a high-speed N-channel silicon-gate process. (See Figure 1). A complete microcomputer system with a 2-μs instruction cycle can be formed by interfacing this circuit with any appropriate memory. Separate 8-bit data and 16-bit address buses simplify the interface and allow direct addressing of 65,536 bytes of memory. Up to 256 input and 256 output ports are also provided with direct addressing. Control signals are brought directly out of the processor and all signals, excluding clocks, are TTL compatible.

1.2 THE STACK

The TMS 8080 incorporates a stack architecture in which a portion of external memory is used as a pushdown stack for storing data from working registers and internal machine status. A 16-bit stack pointer (SP) is provided to facilitate stack location in the memory and to allow almost unlimited interrupt handling capability. The CALL and RST (restart) instructions use the SP to store the program counter (PC) into the stack. The RET (return) instruction uses the SP to acquire the previous PC value. Additional instructions allow data from registers and flags to be saved in the stack.

1.3 REGISTERS

The TMS 8080 has three categories of registers: general registers, program control registers, and internal registers. The general registers and program control registers are listed in Table 1. The internal registers are not accessible by the programmer. They include the instruction register, which holds the present instruction, and several temporary storage registers to hold internal data or latch input and output addresses and data.

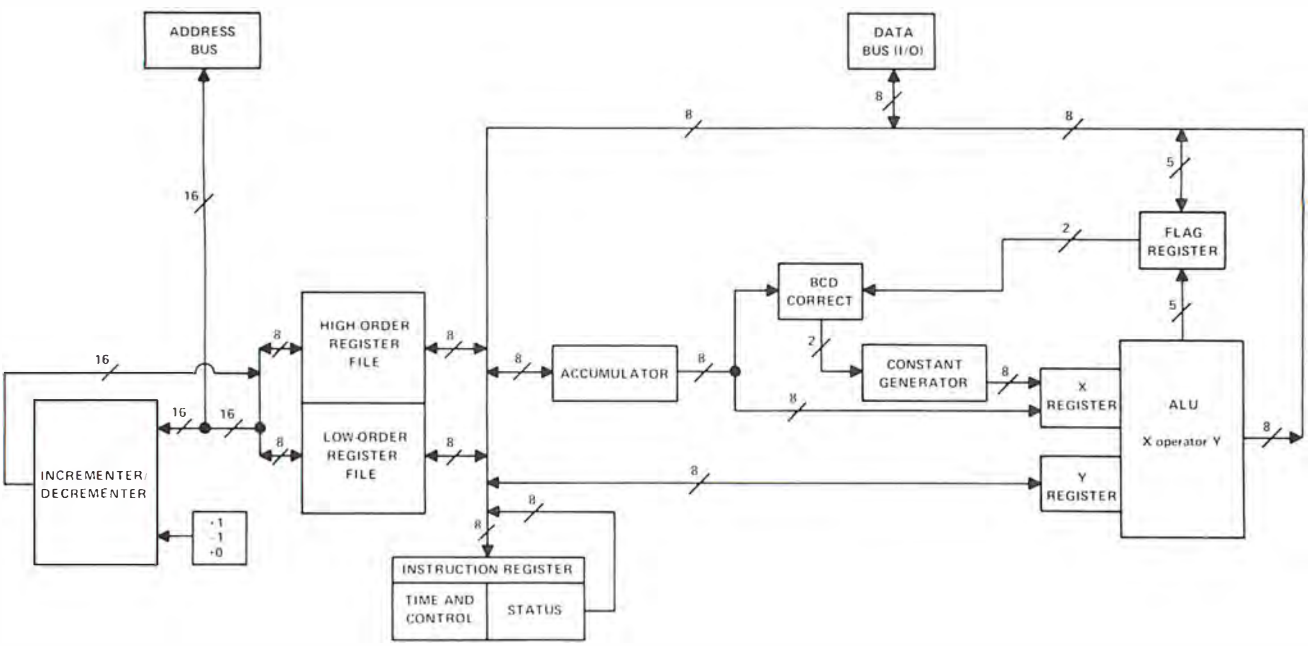


FIGURE 1—TMS 8080 FUNCTIONAL BLOCK DIAGRAM

1.4 THE ARITHMETIC UNIT

Arithmetic operations are performed in an 8-bit parallel arithmetic unit that has both binary and decimal capabilities. Four testable internal flag bits are provided to facilitate program control, and a fifth flag is used for decimal corrections. Table 2 defines these flags and their operation. Decimal corrections are performed with the DAA instruction. The DAA corrects the result of binary arithmetic operation on BCD data as shown in Table 3.

1.5 STATUS AND CONTROL

Two types of status are provided by the TMS8080. Certain status is indicated by dedicated control lines. Additional status is transmitted on the data bus during the beginning of each instruction cycle (machine cycle). Table 4 indicates the pin functions of the TMS8080. Table 5 defines the status information that is presented during the beginning of each machine cycle (SYNC time) on the data bus.

1.6 I/O OPERATIONS

Input/output operations (I/O) are performed using the IN and OUT instructions. The second byte of these instructions indicates the device address (256 device addresses). When an IN instruction is executed, the input device address appears in duplicate on A7 through A0 and A15 through A8, along with $\overline{\text{WO}}$ and INP status on the data bus. The addressed input device then puts its input data on the data bus for entry into the accumulator. When an OUT instruction is executed, the same operation occurs except that the data bus has OUT status and then has output data.

Direct memory access channels (DMA) can be OR-tied directly with the data and address buses through the use of the HOLD and HLDA (hold acknowledge) controls. When a HOLD request is accepted by the CPU, HLDA goes high, the address and data lines are forced to a high-impedance or "floating" condition, and the CPU stops until the HOLD request is removed.

Interfacing with different speed memories is easily accomplished by use of the WAIT and READY pins. During each machine cycle, the CPU polls the READY input and enters a wait condition until the READY line becomes true. When the WAIT output pin is high, it indicates that the CPU has entered the wait state.

Designing interrupt driven systems is simplified through the use of vectored interrupts. At the end of each instruction, the CPU polls the INT input to determine if an interrupt request is being made. This action does not occur if the CPU is in the HOLD state or if interrupts are disabled. The INTE output indicates if the interrupt logic is enabled (INTE is high). When a request is honored, the INTA status bit becomes high, and an RST instruction may be inserted to force the CPU to jump to one of eight possible locations. Enabling or disabling interrupts is controlled by special instructions (EI or DI). The interrupt input is automatically disabled when an interrupt request is accepted or when a RESET signal is received.

1.7 INSTRUCTION TIMING

The execution time of the instructions varies depending on the operation required and the number of memory references needed. A machine cycle is defined to be a memory referencing operation and is either 3, 4, or 5 state times long. A state time (designated S) is a full cycle of clocks $\phi 1$ and $\phi 2$. (NOTE: The exception to this rule is the DAD instruction, which consists of 1 memory reference in 10 state times). The first machine cycle (designated M1) is either 4 or 5 state times long and is the "instruction fetch" cycle with the program counter appearing on the address bus. The CPU then continues with as many M cycles as necessary to complete the execution of the instruction (up to a maximum of 5). Thus the instruction execution time varies from 4 state times (several including ADDr) to 18 (XTHL). The WAIT or HOLD conditions may affect the execution time since they can be used to control the machine (for example to "single step") and the HALT instruction forces the CPU to stop until an interrupt is received. As the instruction execution is completed (or in the HALT state) the INT pin is polled for an interrupt. In the event of an interrupt, the PC will not be incremented during the next M1 and an RST instruction can be inserted.

TABLE 1
TMS 8080 REGISTERS

NAME	DESIGNATOR	LENGTH	PURPOSE
Accumulator	A	8	Used for arithmetic, logical, and I/O operations
B Register	B	8	General or most significant 8 bits of double register BC
C Register	C	8	General or least significant 8 bits of double register BC
D Register	D	8	General or most significant 8 bits of double register DE
E Register	E	8	General or least significant 8 bits of double register DE
H Register	H	8	General or most significant 8 bits of double register HL
L Register	L	8	General or least significant 8 bits of double register HL
Program Counter	PC	16	Contains address of next byte to be fetched
Stack Pointer	SP	16	Contains address of the last byte of data saved in the memory stack
Flag Register	F	5	Five flags (C, Z, S, P, C1)

NOTE: Registers B and C may be used together as a single 16-bit register, likewise, D and E, and H and L.

TABLE 2
FLAG DESCRIPTIONS

SYMBOL	TESTABLE	DESCRIPTION
C	YES	C is the carry/borrow out of the MSB (most significant bit) of the ALU (Arithment Logic Unit). A TRUE condition (C = 1) indicates overflow for addition or underflow for subtraction.
Z	YES	A TRUE condition (Z = 1) indicates that the output of the ALU is equal to zero.
S	YES	A TRUE condition (S = 1) indicates that the MSB of the ALU output is equal to a one (1).
P	YES	A TRUE condition (P = 1) indicates that the output of the ALU has even parity (the number of bits equal to one is even).
C1	NO	C1 is the carry out of the fourth bit of the ALU (TRUE condition). C1 is used only for BCD correction with the DAA instruction.

TABLE 3
FUNCTION OF THE DAA INSTRUCTION

Assume the accumulator (A) contains two BCD digits, X and Y

	7 … 4	3 … 0
ACC	X	Y

ACCUMULATOR BEFORE DAA				ACCUMULATOR AFTER DAA			
C	$A_7 \ldots A_4$	C1	$A_3 \ldots A_0$	C	$A_7 \ldots A_4$	C1	$A_3 \ldots A_0$
0	$X < 10$	0	$Y < 10$	0	X	0	Y
0	$X < 10$	1	$Y < 10$	0	X	0	$Y + 6$
0	$X < 9$	0	$Y \geq 10$	0	$X + 1$	1	$Y + 6$
1	$X < 10$	0	$Y < 10$	1	$X + 6$	0	Y
1	$X < 10$	1	$Y < 10$	1	$X + 6$	0	$Y + 6$
1	$X < 10$	0	$Y \geq 10$	1	$X + 7$	1	$Y + 6$
0	$X \geq 10$	0	$Y < 10$	1	$X + 6$	0	Y
0	$X \geq 10$	1	$Y < 10$	1	$X + 6$	0	$Y + 6$
0	$X \geq 9$	0	$Y \geq 10$	1	$X + 7$	1	$Y + 6$

NOTE: The corrections shown in Table 3 are sufficient for addition. For subtraction, the programmer must account for the borrow condition that can occur and give erroneous results. The most straight forward method is to set A = 99_{16} and carry = 1. Then add the minuend to A after subtracting the subtrahend from A.

TABLE 4
TMS 8080 PIN DEFINITIONS

SIGNATURE	PIN	I/O	DESCRIPTION
A15 (MSB)	36	OUT	A15 through A0 comprise the address bus. True memory or I/O device addresses appear on this 3-state bus during the first state time of each instruction cycle.
A14	39	OUT	
A13	38	OUT	
A12	37	OUT	
A11	40	OUT	
A10	1	OUT	
A9	35	OUT	
A8	34	OUT	
A7	33	OUT	
A6	32	OUT	
A5	31	OUT	
A4	30	OUT	
A3	29	OUT	
A2	27	OUT	
A1	26	OUT	
A0 (LSB)	25	OUT	
D7 (MSB)	6	IN/OUT	D7 through D0 comprise the bidirectional 3-state data bus. Memory, status, or I/O data is transferred on this bus.
D6	5	IN/OUT	
D5	4	IN/OUT	
D4	3	IN/OUT	
D3	7	IN/OUT	
D2	8	IN/OUT	
D1	9	IN/OUT	
D0 (LSB)	10	IN/OUT	
V_{SS}	2		Ground reference
V_{BB}	11		Supply voltage (−5 V nominal)
V_{CC}	20		Supply voltage (5 V nominal)
V_{DD}	28		Supply voltage (12 V nominal)
$\phi 1$	22	IN	Phase 1 clock.
$\phi 2$	15	IN	Phase 2 clock. See page 19 for $\phi 1$ and $\phi 2$ timing.
RESET	12	IN	Reset. When active (high) for a minimum of 3 clock cycles, the RESET input causes the TMS 8080 to be reset. PC is cleared, interrupts are disabled, and after RESET, instruction execution starts at memory location 0. To prevent a lockup condition, a HALT instruction must not be used in location 0.
HOLD	13	IN	Hold signal. When active (high) HOLD causes the TMS 8080 to enter a hold state and float (put the 3-state address and data bus in a high-impedance state). The chip acknowledges entering the hold state with the HLDA signal and will not accept interrupts until it leaves the hold state.
INT	14	IN	Interrupt request. When active (high) INT indicates to the TMS8080 that an interrupt is being requested. The TMS8080 polls INT during a HALT or at the end of an instruction. The request will be accepted except when INTE is low or the CPU is in the HOLD condition.
INTE	16	OUT	Interrupts enabled. INTE indicates that an interrupt will be accepted by the TMS 8080 unless it is in the hold state. INTE is set to a high logic level by the EI (Enable Interrupt) instruction and reset to a low logic level by the DI (Disable Interrupt) instruction. INTE is also reset when an interrupt is accepted and by a high on RESET.
DBIN	17	OUT	Data bus in. DBIN indicates whether the data bus is in an input or an output mode. (high = input, low = output).

TABLE 4 (CONTINUED)

SIGNATURE	PIN	I/O	DESCRIPTION
$\overline{\text{WR}}$	18	OUT	Write. When active (low) WR indicates a write operation on the data bus to memory or to an I/O port.
SYNC	19	OUT	Synchronizing control line. When active (high) SYNC indicates the beginning of each machine cycle of the TMS8080. Status information is also present on the data bus during SYNC for external latches.
HLDA	21	OUT	Hold acknowledge. When active (high) HLDA indicates that the TMS8080 is in a hold state.
READY	23	IN	Ready control line. An active (high) level indicates to the TMS 8080 that an external device has completed the transfer of data to or from the data bus. READY is used in conjunction with WAIT for different memory speeds.
WAIT	24	OUT	Wait status. When active (high) WAIT indicates that the TMS8080 has entered a wait state pending a READY signal from memory.

TABLE 5

TMS 8080 STATUS

SIGNATURE	DATA BUS BIT	DESCRIPTION
INTA	D0	Interrupt acknowledge.
$\overline{\text{WO}}$	D1	Indicates that current machine cycle will be a read (input) (high = read) or a write (output) (low = write) operation.
STACK	D2	Indicates that address is stack address from the SP.
HLTA	D3	HALT instruction acknowledge.
OUT	D4	Indicates that the address bus has an output device address and the data bus has output data.
M1	D5	Indicates instruction acquisition for first byte.
INP	D6	Indicates address bus has address of input device.
MEMR	D7	Indicates that data bus will be used for memory read data.

2. TMS 8080 INSTRUCTION SET

2.1 INSTRUCTION FORMATS

TMS 8080 instructions are either one, two, or three bytes long and are stored as binary integers in successive memory locations in the format shown below.

One-Byte Instructions

D7 D6 D5 D4 D3 D2 D1 D0	OP CODE

Two-Byte Instructions

D7 D6 D5 D4 D3 D2 D1 D0	OP CODE
D7 D8 D5 D4 D3 D2 D1 D0	OPERAND

Three-Byte Instructions

D7 D6 D5 D4 D3 D2 D1 D0	OP CODE
D7 D6 D5 D4 D3 D2 D1 D0	LOW ADDRESS OR OPERAND 1
D7 D6 D5 D4 D3 D2 D1 D0	HIGH ADDRESS OR OPERAND 2

2.2 INSTRUCTION SET DESCRIPTION

Operations resulting from the execution of TMS 8080 instructions are described in this section. The flags that are affected by each instruction are given after the description.

2.2.1 INSTRUCTION SYMBOLS

SYMBOL	DESCRIPTION		
<b_2>	Second byte of instruction		
<b_3>	Third byte of instruction		
r_a		Register #	Register Name
		000	B
		001	C
		010	D
		011	E
		100	H
		101	L
		111	A
r_b		Register #	Register Name
		00	BC
		01	DE
		10	HL
		11	SP
r_c		Register #	Register Name
		0	BC
		1	DE
r_d		Register #	Register Name
		00	BC
		01	DE
		10	HL
r_{dL}	Least significant 8 bits of r_d		
r_{dH}	Most significant 8 bits of r_d		
f	Flags	True condition	
	Zero (Z)	Result is zero	
	Carry (C)	Carry/borrow out of MSB is one	
	Parity (P)	Parity of result is even	
	Sign (S)	MSB of result is one	
	Carry 1(C1)	Carry out of fourth bit is one	
M	Memory address defined by registers H and L		
( )	Contents of specified address or register		
[ ]	Contents at address contained in specified register		
←	Is transferred to		
↔	Exchange		
Am	Bit m of A register (accumulator)		
{ }	Flags affected		
b_2	Single byte immediate operand		
b_3b_2	Double byte immediate operand		
$(nnn)_8$	(nnn) is an octal (base 8) number		

2.2.2 ACCUMULATOR GROUP INSTRUCTIONS

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
ACI	b_2	2	2/7	$(A) \leftarrow (A) + <b_2> + (carry)$, add the second byte of the instruction and the contents of the carry flag to register A and place in A. {C,Z,S,P,C1}
ADC	M	1	2/7	$(A) \leftarrow (A) + (M) + (carry)$. {C,Z,S,P,C1}
ADC	r_a	1	1/4	$(A) \leftarrow (A) + (r_a) + (carry)$. {C,Z,S,P,C1}
ADD	M	1	2/7	$(A) \leftarrow (A) + (M)$, add the contents of M to register A and place in A. {C,Z,S,P,C1}
ADD	r_a	1	1/4	$(A) \leftarrow (A) + (r_a)$. {C,Z,S,P,C1}
ADI	b_2	2	2/7	$(A) \leftarrow (A) + <b_2>$. {C,Z,S,P,C1}
ANA	M	1	2/7	$(A) \leftarrow (A)$ AND (M), take the logical AND of M and register A and place in A. The carry flag will be reset low. {C,Z,S,P,C1}
ANA	r_a	1	1/4	$(A) \leftarrow (A)$ AND (r_a). {C,Z,S,P,C1}
ANI	b_2	2	2/7	$(A) \leftarrow (A)$ AND $<b_2>$. {C,Z,S,P,C1}
CMA		1	1/4	$(A) \leftarrow (\bar{A})$, complement A.
CMC		1	1/4	$(carry) \leftarrow (\overline{carry})$, complement the carry flag. {C}
CMP	M	1	2/7	$(A) - (M)$, compare the contents of M to register A and set the flags accordingly. {C,Z,S,P,C1} $(A) = (M)$ $Z = 1$ $(A) \neq (M)$ $Z = 0$ $(A) < (M)$ $C = 1$ $(A) > (M)$ $C = 0$
CMP	r_a	1	1/4	$(A) - (r_a)$. {C,Z,S,P,C1}
CPI	b_2	2	2/7	$(A) - <b_2>$. {C,Z,S,P,C1}
DAA		1	1/4	$(A) \leftarrow$ BCD correction of (A). The 8 bit A contents is corrected to form two 4 bit BCD digits after a binary arithmetic operation. A fifth flag C1 indicates the overflow from A_3. The carry flag C indicates the overflow from A_7 (See Table 3). {C,Z,S,P,C1}
DAD	r_b	1	1/10	$(HL) \leftarrow (HL) + (r_b)$, add the contents of double register r_b to double register HL and place in HL. {C}
LDA	b_3b_2	3	4/13	$(A) \leftarrow [<b_3> <b_2>]$
LDAX	r_c	1	2/7	$(A) \leftarrow [(r_c)]$
ORA	M	1	2/7	$(A) \leftarrow (A)$ OR (M), take the logical OR of the contents of M and register A and place in A. The carry flag will be reset. {C,Z,S,P,C1}
ORA	r_a	1	1/4	$(A) \leftarrow (A)$ OR (r_a). {C,Z,S,P,C1}
ORI	b_2	2	2/7	$(A) \leftarrow (A)$ OR $<b_2>$. {C,Z,S,P,C1}
RAL		1	1/4	$A_{m+1} \leftarrow A_m$, $A_0 \leftarrow (carry)$, $(carry) \leftarrow (A_7)$. Shift the contents of register A to the left one bit through the carry flag. {C}
RAR		1	1/4	$A_m \leftarrow A_{m+1}$, $A_7 \leftarrow (carry)$, $(carry) \leftarrow A_0$. {C}
RLC		1	1/4	$A_{m+1} \leftarrow A_m$, $A_0 \leftarrow A_7$ $(carry) \leftarrow (A_7)$. Shift the contents of register A to the left one bit. Shift A_7 into A and into the carry flag. {C}
RRC		1	1/4	$A_m \leftarrow A_{m+1}$, $A_7 \leftarrow A_0$, $(carry) \leftarrow (A_0)$. {C}

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
SBB	M	1	2/7	$(A) \leftarrow (A)-(M)-(carry)$, subtract the contents of M and the contents of the carry flag from register A and place in A. Two's complement subtraction is used and a true borrow causes the carry flag to be set (underflow condition). {C,Z,S,P,C1}
SBB	r_a	1	1/4	$(A) \leftarrow (A)-(r_a)-(carry)$. {C,Z,S,P,C1}
SBI	b_2	2	2/7	$(A) \leftarrow (A)-\langle b_2 \rangle-(carry)$. {C,Z,S,P,C1}
STA	b_3b_2	3	4/13	$[\langle b_3 \rangle\ \langle b_2 \rangle] \leftarrow (A)$, store contents of A in memory address given in bytes 2 and 3.
STAX	r_c	1	2/7	$[(r_c)] \leftarrow (A)$, store contents of A in memory address given in BC or DE.
STC		1	1/4	$(carry) \leftarrow 1$, set carry flag to a 1 (true condition).
SUB	M	1	2/7	$(A) \leftarrow (A)-(M)$, subtract the contents of M from register A and place in A. Two's complement subtraction is used and a true borrow causes the carry flag to be set (underflow condition). {C,Z,S,P,C1}
SUB	r_a	1	1/4	$(A) \leftarrow (A)-(r_a)$. {C,Z,S,P,C1}
SUI	b_2	2	2/7	$(A) \leftarrow (A)-\langle b_2 \rangle$. {C,Z,S,P,C1}
XRA	M	1	2/7	$(A) \leftarrow (A)$ XOR (M), take the exclusive OR of the contents of M and register A and place in A. The carry flag will be reset. {C,Z,S,P,C1}
XRA	r_a	1	1/4	$(A) \leftarrow (A)$ XOR (r_a). {C,Z,S,P,C1}
XRI	b_2	2	2/7	$(A) \leftarrow (A)$ XOR $\langle b_2 \rangle$. {C,Z,S,P,C1}

2.2.3 INPUT/OUTPUT INSTRUCTIONS

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
IN	b_2	2	3/10	$(A) \leftarrow$ (input data from data bus), byte 2 is sent on bits A7-A0 and A15-A8 as the input device address. INP status is given on the data bus.
OUT	b_2	2	3/10	(Output data) $\leftarrow (A)$, byte 2 is sent on bits A7-A0 and A15-A8 as the output device address. OUT status is given on the data bus.

2.2.4 MACHINE INSTRUCTIONS

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
HLT		1	2/7	Halt, all machine operations stop. All registers are maintained. Only an interrupt can return the TMS 8080 to the run mode. Note that a HLT should not be placed in location zero, otherwise after the reset pin is active, the TMS 8080 will enter a nonrecoverable state (until power is removed), i.e., in halt with interrupts disabled. This condition also occurs if a HLT is executed while interrupts are disabled. HLTA status is given on the data bus.
NOP		1	1/4	$(PC) \leftarrow (PC)+1$, no operation.

2.2.5 PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
CALL	b_3b_2	3	5/17	[(SP)−1] [(SP)−2]←(PC), (SP)←(SP)−2, (PC)←<b_3> <b_2>, transfer PC to the stack address given by SP, decrement SP twice, and jump unconditionally to address given in bytes 2 and 3.
Conditional call instructions for true flags:				
(f)			5/17 (Pass)	If (f) = 1, [(SP)−1] [(SP)−2]←(PC), (SP)←(SP)−2, (PS)←<b_3> <b_2>, otherwise (PC)←(PC)+3. If the flag specified, f, is 1, then execute a call. Otherwise, execute the next instruction.
CC (carry)	b_3b_2	3	3/11 (Fail)	
CPE (parity)	b_3b_2	3		
CM (sign)	b_3b_2	3		
CZ (zero)	b_3b_2	3		
Conditional call instructions for false flags:				
(f)			5/17 (Pass)	If (f) = 0, [(SP)−1] [(SP)−2]←(PC), (SP)←(SP)−2, (PC)←<b_3> <b_2>, otherwise (PC)←(PC)+3.
CNC (carry)	b_3b_2	3	3/11 (Fail)	
CPO (parity)	b_3b_2	3		
CP (sign)	b_3b_2	3		
CNZ (zero)	b_3b_2	3		
DI		1	1/4	Disable interrupts. INTE is driven false to indicate that no interrupts will be accepted.
EI		1	1/4	Enable interrupts. INTE is driven true to indicate that an interrupt will be accepted. Execution of this instruction is delayed to allow the next instruction to be executed before the INT input is polled.
JMP	b_3b_2	3	3/10	(PC)←<b_3> <b_2>, jump unconditionally to address given in bytes 2 and 3.
Conditional jump instructions for true flags:				
(f)			3/10	If (f) = 1, (PC)←<b_3><b_2>, otherwise (PC)←(PC)+3. If the flag specified, f, is 1, execute a JMP. Otherwise, execute the next instruction.
JC (carry)	b_3b_2	3		
JPE (parity)	b_3b_2	3		
JM (sign)	b_3b_2	3		
JZ (zero)	b_3b_2	3		
Conditional jump instructions for false flags:				
(f)			3/10	If (f) = 0, (PC)←<b_3> <b_2>, othewise (PC)←(PC)+3.
JNC (carry)	b_3b_2	3		
JPO (parity)	b_3b_2	3		
JM (sign)	b_3b_2	3		
JNZ (zero)	b_3b_2	3		
PCHL		1	1/5	(PC)←(HL)
POP	PSW	1	3/10	(F)←[(SP)], (A)←[(SP)+1], (SP)←(SP)+2, restore *the last* stack values addressed by SP into A and F. Increment SP twice.
POP	r_d	1	3/10	(r_{dL})←[(SP)], (r_{dH})←[(SP)+1], (SP)←(SP)+2.
PUSH	PSW	1	3/11	[(SP)−1]←(A), [(SP)−2]←(F), (SP)←(SP)−2, save the contents of A and F into the stack addressed by SP. Decrement SP twice.
PUSH	r_d	1	3/11	[(SP)−1]←(r_{dL}), [(SP)−2]←(r_{dH}), (SP)←(SP)−2.
RET		1	3/10	(PC)←[(SP)] [(SP)+1], (SP)←(SP)+2, return to program at memory address given by last values in the stack. The SP is incremented by two.

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
Conditional return instructions for true flags:				
(f)			3/11 (Pass)	If (f) = 1, (PC)←[(SP)] [(SP+1)], (SP)←(SP)+2. If the flag specified, f, is 1, execute a RET. Otherwise, execute the next instruction.
RC (carry)	C	1	1/5 (Fail)	
RPE (parity)	P	1		
RM (sign)	S	1		
RZ (zero)	Z	1		
Conditional return instructions for false flags:				
(f)			3/11 (Pass)	If (f) = 0, (PC)←[(SP)] [(SP)+1], (SP)←(SP)+2.
RNC (carry)	C	1	1/5 (Fail)	
RPO (parity)	P	1		
RP (sign)	S	1		
RNZ (zero)	Z	1		
RST		1	3/11	[(SP)−1] [(SP)−2] ←(PC) (SP)←(SP)−2, (PC)←$0000R0_8$ where R is a 3 bit field in RST (RST=$3R7_8$). Transfer PC to the stack address given by SP, decrement SP twice, and jump to the address specified by R.
SPHL		1	1/5	(SP)←(HL).

2.2.6 REGISTER GROUP INSTRUCTIONS

MNEMONIC	OPERANDS	BYTES	M CYCLES/ STATES	DESCRIPTION
DCR	M	1	3/10	(M)←(M)−1, decrement the contents of memory location specified by H and L. {Z,S,P,C1}
DCR	r_a	1	1/5	$(r_a)\leftarrow(r_a)-1$, decrement the contents of register r_a. {Z,S,P,C1}
DCX	r_b	1	1/5	$(r_b)\leftarrow(r_b)-1$, decrement double registers BC, DE, HL, or SP.
INR	M	1	3/10	(M)←(M)+1, increment the contents of memory location specified by H and L. {Z,S,P,C1 }
INR	r_a	1	1/5	$(r_a)\leftarrow(r_a)+1$, increment the contents of register r_a. {Z,S,P,C1}
INX	r_b	1	1/5	$(r_b)\leftarrow(r_b)+1$, increment double registers BC, DE, HL, or SP.
LHLD	b_3b_2	3	5/16	(L)←[$<b_3>$ $<b_2>$] ; (H)← [$<b_3>$ $<b_2>+1$], load registers H and L with contents of the two memory locations specified by bytes 3 and 2.
LXI	$r_bb_3b_2$	3	3/10	$(r_{bH})\leftarrow<b_3>$; $(r_{bL})\leftarrow<b_2>$, load double registers BC, DE, HL, or SP immediate with bytes 3, 2, respectively.
MVI	M,b_2	2	3/10	(M)←$<b_2>$, store immediate byte 2 in the address specified by HL
MVI	r_ab_2	2	2/7	$(r_a)\leftarrow<b_2>$, load register r_a immediate with byte 2 of the instruction.
MOV	Mr_a	1	2/7	(M)←(r_a), store register r_a in the memory location addressed by H and L.
MOV	r_aM	1	2/7	(r_a)←(M), load register r_a with contents of memory addressed by HL.
MOV	$r_{a1}r_{a2}$	1	1/5	$(r_{a1})\leftarrow(r_{a2})$, load register r_{a1} with contents of r_{a2}. r_{a2} contents remain unchanged.
SHLD	b_3b_2	3	5/16	[$<b_3>$ $<b_2>$]←(L); [$<b_3>$ $<b_2>+1$]←(H), store the contents of H and L into two successive memory locations specified by bytes 3 and 2.
XCHG		1	1/4	(H)↔(D); (L)↔(E), exchange double registers HL and DE
XTHL		1	5/18	(L)↔[(SP)], (H)↔[(SP)+1], (SP)=(SP), exchange the top of the stack with register HL.

2.3 INSTRUCTION SET OPCODES ALPHABETICALLY LISTED

MNEMONIC	BYTES	DESCRIPTION	REGISTER AFFECTED	POSITIVE-LOGIC HEX OPCODE D7–D4	POSITIVE-LOGIC HEX OPCODE D3–D0	CLOCK CYCLES*
ACI	2	Add immediate to A with carry†		C	E	7
ADC M	1	Add memory to A with carry†		8	E	7
ADC r	1	Add register to A with carry†	B	8	8	4
			C	8	9	
			D	8	A	
			E	8	B	
			H	8	C	
			L	8	D	
			A	8	F	
ADD M	1	Add memory to A†		8	6	7
ADD r	1	Add register to A†	B	8	0	4
			C	8	1	
			D	8	2	
			E	8	3	
			H	8	4	
			L	8	5	
			A	8	7	
ADI	2	Add immediate to A†		C	6	7
ANA M	1	AND memory with A†		A	6	7
ANAr	1	AND register with A†	B	A	0	4
			C	A	1	
			D	A	2	
			E	A	3	
			H	A	4	
			L	A	5	
			A	A	7	
ANI	2	AND immediate with A†		E	6	7
CALL	3	Call unconditional		C	D	17
CC	3	Call on carry		D	C	11/17
CM	3	Call on minus		F	C	11/17
CMA	1	Complement A		2	F	4
CMC	1	Complement carry‡		3	F	4
CMP M	1	Compare memory with A†		B	E	7
CMP r	1	Compare register with A				
			B	B	8	4
			C	B	9	
			D	B	A	
			E	B	B	
			H	B	C	
			L	B	D	
			A	B	F	
CNC	3	Call on no carry		D	4	11/17
CNZ	3	Call on no zero		C	4	11/17
CP	3	Call on positive		F	4	11/17
CPE	3	Call on parity even		E	C	11/17
CPI	2	Compare immediate with A†		F	E	7
CPO	3	Call on parity odd		E	4	11/17
CZ	3	Call on zero		C	C	11/17
DAA	1	Decimal adjust A†		2	7	4

* Two possible cycle times (11/17) indicate instruction cycles dependent on condition flags.
† All flags (C, Z, S, P, C1) affected.
‡ Only carry flag affected.

MNEMONIC	BYTES	DESCRIPTION	REGISTER AFFECTED	POSITIVE-LOGIC HEX OPCODE D7–D4	D3–D0	CLOCK CYCLES
DAD B	1	Add B&C to H&L‡		0	9	10
DAD D	1	Add D&E to H&L‡		1	9	10
DAD H	1	Add H&L to H&L‡		2	9	10
DAD SP	1	Add stack pointer to H&L‡		3	9	10
DCR M	1	Decrement Memory§		3	5	10
DCR r	1	Decrement Register§	B	0	5	5
			C	0	D	
			D	1	5	
			E	1	D	
			H	2	5	
			L	2	D	
			A	3	D	
DCX B	1	Decrement B&C		0	B	5
DCX D	1	Decrement D&E		1	B	5
DCX H	1	Decrement H&L		2	B	5
DCX SP	1	Decrement stack pointer		3	B	5
DI	1	Disable interrupts		F	3	4
EI	1	Enable interrupts		F	B	4
HLT	1	Halt		7	6	7
IN	2	Input		D	B	10
INR M	1	Increment memory§		3	4	10
INR r	1	Increment register§	B	0	4	5
			C	0	C	
			D	1	4	
			E	1	C	
			H	2	4	
			L	2	C	
			A	3	C	
INX B	1	Increment B&C register		0	3	5
INX D	1	Increment D&E register		1	3	5
INX H	1	Increment H&L register		2	3	5
INX SP	1	Increment stack pointer		3	3	5
JC	3	Jump on carry		D	A	10
JM	3	Jump on minus		F	A	10
JMP	3	Jump unconditional		C	3	10
JNC	3	Jump on no carry		D	2	10
JNZ	3	Jump on no zero		C	2	10
JP	3	Jump on positive		F	2	10
JPE	3	Jump on parity even		E	A	10
JPO	3	Jump on parity odd		E	2	10
JZ	3	Jump on zero		C	A	10
LDA	1	Load A direct		3	A	13
LDAX B	1	Load A indirect		0	A	7
LDAX D	1	Load A indirect		1	A	7
LHLD	3	Load H&L direct		2	A	16
LXI B	3	Load immediate register pair B&C		0	1	10
LXI D	3	Load immediate register pair D&E		1	1	10
LXI H	3	Load immediate register		2	1	10
LXI SP	3	Load immediate stack pointer		3	1	10

‡Only carry flag affected.
§All flags except carry affected.

MNEMONIC	BYTES	DESCRIPTION	REGISTER AFFECTED	POSITIVE-LOGIC HEX OPCODE D7–D4	D3–D0	CLOCK CYCLES
MOV M,r	1	Move register to memory	B	7	0	7
			C	7	1	
			D	7	2	
			E	7	3	
			H	7	4	
			L	7	5	
			A	7	7	
MOV r,M	1	Move memory to register	B	4	6	7
			C	4	E	
			D	5	6	
			E	5	E	
			H	6	6	
			L	6	E	
			A	7	E	
MOV r_1, r_2	1	Move register to register	B,B	4	0	5
			B,C	4	1	
			B,D	4	2	
			B,E	4	3	
			B,H	4	4	
			B,L	4	5	
			B,A	4	7	
			C,B	4	8	
			C,C	4	9	
			C,D	4	A	
			C,E	4	B	
			C,H	4	C	
			C,L	4	D	
			C,A	4	F	
			D,B	5	0	
			D,C	5	1	
			D,D	5	2	
			D,E	5	3	
			D,H	5	4	
			H,L	5	5	
			D,A	5	7	
			E,B	5	8	
			E,C	5	9	
			E,D	5	A	
			E,E	5	B	
			E,H	5	C	
			E,L	5	D	
			E,A	5	F	
			H,B	6	0	
			H,C	6	1	
			H,D	6	2	
			H,E	6	3	
			H,H	6	4	
			H,L	6	5	
			H,A	6	7	
			L,B	6	8	

MNEMONIC	BYTES	DESCRIPTION	REGISTER AFFECTED	POSITIVE-LOGIC HEX OPCODE D7–D4	D3–D0	CLOCK CYCLES*
MOV r_1, r_2	1	Move register to register (continued)	L,C	6	9	
			L,D	6	A	
			L,E	6	B	
			L,H	6	C	
			L,L	6	D	
			L,A	6	F	
			A,B	7	8	
			A,C	7	9	
			A,D	7	A	
			A,E	7	B	
			A,H	7	C	
			A,L	7	D	
			A,A	7	F	
MVI M	2	Move immediate memory		3	6	10
MVI r	2	Move immediate register	B	0	6	7
			C	0	E	
			D	1	6	
			E	1	E	
			H	2	6	
			L	2	E	
			A	3	E	
NOP	1	No operation	4	0	0	4
ORA M	1	OR memory with A†		B	6	7
ORA r	1	OR register with A†	B	B	0	4
			C	B	1	
			D	B	2	
			E	B	3	
			H	B	4	
			L	B	5	
			A	B	7	
ORI	2	OR immediate with A†		F	6	7
OUT	2	Output		D	3	10
PCHL	1	H&L to program counter		E	9	5
POP B	1	Pop register pair B&C off stack		C	1	10
POP D	1	Pop register pair D&E off stack		D	1	10
POP H	1	Pop register pair H&L off stack		E	1	10
POP PSW	1	Pop A and flags off stack†		F	1	10
PUSH B	1	Push register pair B&C		C	5	11
PUSH D	1	Push register pair D&C		D	5	11
PUSH H	2	Push register pair H&L on stack		E	5	11
PUSH PSW	1	Push A and Flags on stack		F	5	11
RAL	1	Rotate A left through carry ‡		1	7	4
RAR	1	Rotate A right through carry ‡		1	F	4
RC	1	Return on carry		D	8	5/11
RET	1	Return		C	9	10
RLC	1	Rotate A left ‡		0	7	4
RM	1	Return on minus		F	8	5/11
RNC	1	Return on no carry		D	0	5/11
RNZ	1	Return on no zero		C	0	5/11
RP	1	Return on positive		F	0	5/11

* Two possible cycles times (11/17) indicate instruction cycles dependent on condition flags.
† All flags (C, Z, S, P, C1) affected.
‡ Only carry flag affected.

MNEMONIC	BYTES	DESCRIPTION	REGISTER AFFECTED	POSITIVE-LOGIC HEX OPCODE D7–D4	POSITIVE-LOGIC HEX OPCODE D3–D0	CLOCK CYCLES*
RPE	1	Return on parity even		E	8	5/11
RPO	1	Return on parity odd		E	0	5/11
RRC	1	Rotate A right‡		0	F	4
RST	1	Restart				11
			$PC \leftarrow 0000_{16}$	C	7	
			$PC \leftarrow 0008_{16}$	C	F	
			$PC \leftarrow 0010_{16}$	D	7	
			$PC \leftarrow 0018_{16}$	D	F	
			$PC \leftarrow 0020_{16}$	E	7	
			$PC \leftarrow 0028_{16}$	E	F	
			$PC \leftarrow 0030_{16}$	F	7	
			$PC \leftarrow 0038_{16}$	F	F	
RZ	1	Return on Zero		C	8	5/11
SBB M	1	Subtract memory from A with borrow†		9	E	7
SBB r	1	Subtract register from A with borrow†	B	9	8	4
			C	9	9	
			D	9	A	
			E	9	B	
			H	9	C	
			L	9	D	
			A	9	F	
SBI	2	Subtract immediate from A with borrow†		D	E	7
SHLD	3	Store H&L direct		2	2	16
SPHL	1	H&L to stack pointer		F	9	5
STA	3	Store A direct		3	2	13
STAX B	1	Store A indirect		0	2	7
STAX D	1	Store A indirect		1	2	7
STC	1	Set carry‡		3	7	4
SUB M	1	Subtract memory from A†		9	6	7
SUB r	1	Subtract register from A†	B	9	0	4
			C	9	1	
			D	9	2	
			E	9	3	
			H	9	4	
			L	9	5	
			A	9	7	
SUI	2	Subtract immediate from A†		D	6	7
XCHG	1	Exchange D&E, H&L registers		E	B	4
XRA M	1	Exclusive OR memory with A†		A	E	7
XRA r	1	Exclusive OR register with A†	B	A	8	4
			C	A	9	
			D	A	A	
			E	A	B	
			H	A	C	
			L	A	D	
			A	A	F	
XRI	2	Exclusive OR immediate with A†		E	E	7
XTHL	1	Exchange top of stack H&L		E	3	18

*Two possible cycles times (11/17) indicate instruction cycles dependent on condition flags.
†All flags (C, Z, S, P, C1) affected.
‡Only carry flag affected.

3. TMS 8080 ELECTRICAL AND MECHANICAL SPECIFICATIONS

3.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Supply voltage, V_{CC} (see Note 1) . . . . −0.3 V to 20 V
Supply voltage, V_{DD} (see Note 1 . . . . −0.3 V to 20 V
Supply voltage, V_{SS} (see Note 1) . . . . −0.3 V to 20 V
All input and output voltages (see Note 1) . . . . −0.3 V to 20 V
Continuous power dissipation . . . . 1.5 W
Operating free-air temperature range . . . . 0°C to 70°C
Storage temperature range . . . . −65°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: Under absolute maximum ratings voltage values are with respect to the normally most negative supply voltage, V_{BB} (substrate). Throughout the remainder of this data sheet, voltage values are with respect to V_{SS} unless otherwise noted.

3.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−4.75	−5	−5.25	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks) (see Note 2)	3.3		V_{CC}+1	V
High-level clock input voltage, $V_{IH(\phi)}$	V_{DD}−1		V_{DD}+1	V
Low-level input voltage, V_{IL} (all inputs except clocks) (see Note 3)	−1		0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$ (see Note 3)	−1		0.6	V
Operating free-air temperature, T_A	0		70	C

3.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current (any input except clocks and data bus)	V_I = 0 V to V_{CC}			±10	μA
$I_{I(\phi)}$	Clock input current	$V_{I(\phi)}$ = 0 V to V_{DD}			±10	μA
$I_{I(DB)}$	Input current, data bus	$V_{I(DB)}$ = 0 V to V_{CC}			−100	μA
$I_{I(hold)}$	Address or data bus input current during hold	$V_{I(ad)}$ or $V_{I(DB)}$ = V_{CC}			10	μA
		$V_{I(ad)}$ or $V_{I(DB)}$ = 0 V			−100	
V_{OH}	High-level output voltage	I_{OH} = 100 μA	3.7			V
V_{OL}	Low-level output voltage	$I_{OL(DB)}$ = 1.7 mA, I_{OL} = 0.75 mA (any output except DB)			0.45	V
$I_{BB(av)}$	Average supply current from V_{BB}	Operating at $t_{c(\phi)}$ = 480 ns, T_A = 25°C		−0.01	−1	mA
$I_{CC(av)}$	Average supply current from V_{CC}			60	75	
$I_{DD(av)}$	Average supply current from V_{DD}			40	67	
C_i	Capacitance, any input except clock	V_{CC} = V_{DD} = V_{SS} = 0 V, V_{BB} = −4.75 to −5.25 V, f = 1 MHz, All other pins at 0 V		10	20	pF
$C_{i(\phi)}$	Clock input capacitance			5	10	
C_o	Output capacitance			10	20	

†All typical values are at T_A = 25°C and nominal voltages.

NOTES: 2. Active pull-up resistors of nominally 2 kΩ will be switched onto the data bus when DBIN is high and the data input voltage is more positive than V_{IH} min.
3. The algebraic convention where the most negative limit is designated as minimum is used in this specification for logic voltage levels only.

3.4 TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURE 2)

		MIN	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time (see Note 5)	480	2000	ns
$t_{r(\phi)}$	Clock rise time	5	50	ns
$t_{f(\phi)}$	Clock fall time	5	50	ns
$t_{w(\phi 1)}$	Pulse width, clock 1 high	60		ns
$t_{w(\phi 2)}$	Pulse width, clock 2 high	220		ns
$t_{d(\phi 1L\text{-}\phi 2)}$	Delay time, clock 1 low to clock 2	0		ns
$t_{d(\phi 2\text{-}\phi 1)}$	Delay time, clock 2 to clock 1	70		ns
$t_{d(\phi 1H\text{-}\phi 2)}$	Delay time, clock 1 high to clock 2 (time between leading edges)	130		ns
$t_{su(da\text{-}\phi 1)}$	Data setup time with respect to clock 1	50		ns
$t_{su(da\text{-}\phi 2)}$	Data setup time with respect to clock 2	150		ns
$t_{su(hold)}$	Hold input setup time	140		ns
$t_{su(int)}$	Interrupt input setup time	180		ns
$t_{su(rdy)}$	Ready input setup time	120		ns
$t_{h(da)}$	Data hold time (see Note 6)	$t_{PD(DBI)}$		ns
$t_{h(hold)}$	Hold input hold time	0		ns
$t_{h(int)}$	Interrupt input hold time	0		ns
$t_{h(rdy)}$	Ready input hold time	0		ns

NOTES: 5. $t_{c(\phi)} = t_{d(\phi 1L\text{-}\phi 2)} + t_{r(\phi 2)} + t_{w(\phi 2)} + t_{f(\phi 2)} + t_{d(\phi 2\text{-}\phi 1)} + t_{r(\phi 1)}$. 480 ns $\leqslant t_{c(\phi)} \leqslant$ 2000 ns.

6. The data input should be enabled using the DBIN status signal. No bus conflict can then occur and the data hold time requirement is thus assured.

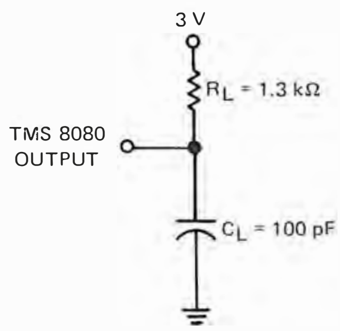
3.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURE 2)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
$t_{PD(ad)}$	Propagation delay time, clock 2 to address outputs	C_L = 100 pF, R_L = 1.3 kΩ		200	ns
$t_{PD(da)}$	Propagation delay time, clock 2 to data bus			220	ns
$t_{PD(cont)}$	Propagation delay time, clocks to control outputs			120	ns
$t_{PD(DBI)}$	Propagation delay time, clock 2 to DBIN output		25	140	ns
$t_{PD(int)}$	Propagation delay time, clock 2 to INTE output			200	ns
t_{DI}	Time for data bus to enter input mode			$t_{PD(DBI)}$	ns
t_{PXZ}	Disable time to high-impedance state during hold (address outputs and data bus)			120	ns

The time that the address outputs and output data will remain stable after $\overline{WR}$ goes high, t_{WA} and $t_{WD} \geqslant t_{d(\phi 1H\text{-}\phi 2)}$.
The time between address outputs becoming stable and $\overline{WR}$ going low, $t_{AW} \leqslant 2\ t_{c(\phi)} - t_{d(\phi 1H\text{-}\phi 2)} - t_{r(\phi)} - 120$ ns.
The time between output data becoming stable and $\overline{WR}$ going low, $t_{DW} \geqslant t_{c(\phi)} - t_{d(\phi 1H\text{-}\phi 2)} - t_{f(\phi)} - 150$ ns.
The following are relevant when interfacing to devices requiring V_{IH} min of 3.3 V:

a) Maximum output rise time (t_{TLH}) from 0.8 V to 3.3 V is 140 ns with C_L as specified for the propagation delay times above.

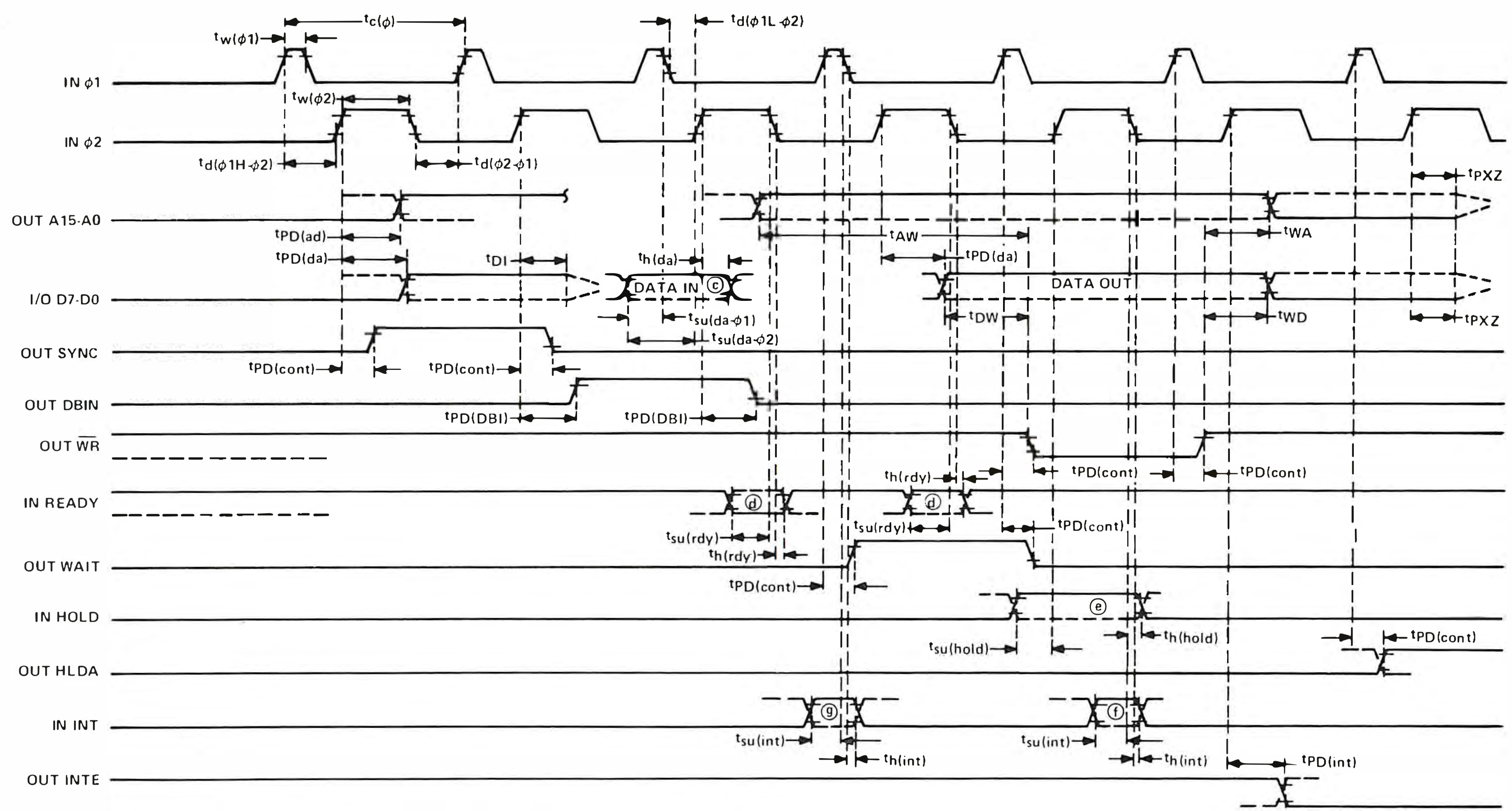
b) Maximum propagation delay times when measured to $V_{ref(H)}$ = 3 V (instead of 2 V) will be 60 ns more than as specified above with C_L as specified.



C_L includes probe and jig capacitance.

LOAD CIRCUIT

voltage waveforms (see notes a and b)

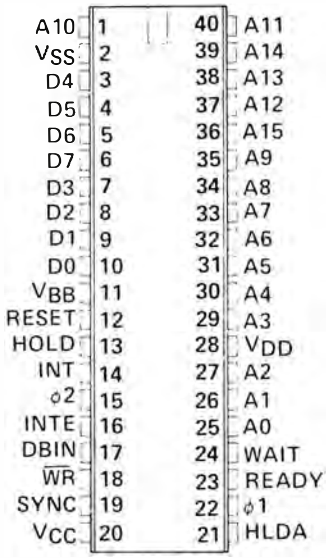


NOTES:
a. This timing diagram shows timing relationships only, it does not represent any specific machine cycle.
b. Time measurements are made at the following reference voltages: Clock, $V_{ref(H)}$ = 9.5 V, $V_{ref(L)}$ = 1 V. Other inputs, $V_{ref(H)}$ = 2 V, $V_{ref(L)}$ = 0.8 V.
c. Data in must be stable for this period when DBIN is high during S3. Requirements for both $t_{su(da\text{-}\phi 1)}$ and $t_{su(da\text{-}\phi 2)}$ must be satisfied.
d. The ready signal must be stable for this period during S2 or SW. This requires external synchronization.
e. The hold signal must be stable for this period during S2 or SW when entering the hold mode and during S3, S4, S5 and SWH when in the hold mode. This requires external synchronization.
f. The interrupt signal must be stable during this period on the last clock cycle of any instruction to be recognized on the following instruction. External synchronization is not required.
g. During halt mode only, timing is with respect to the clock 1 falling edge.

FIGURE 2

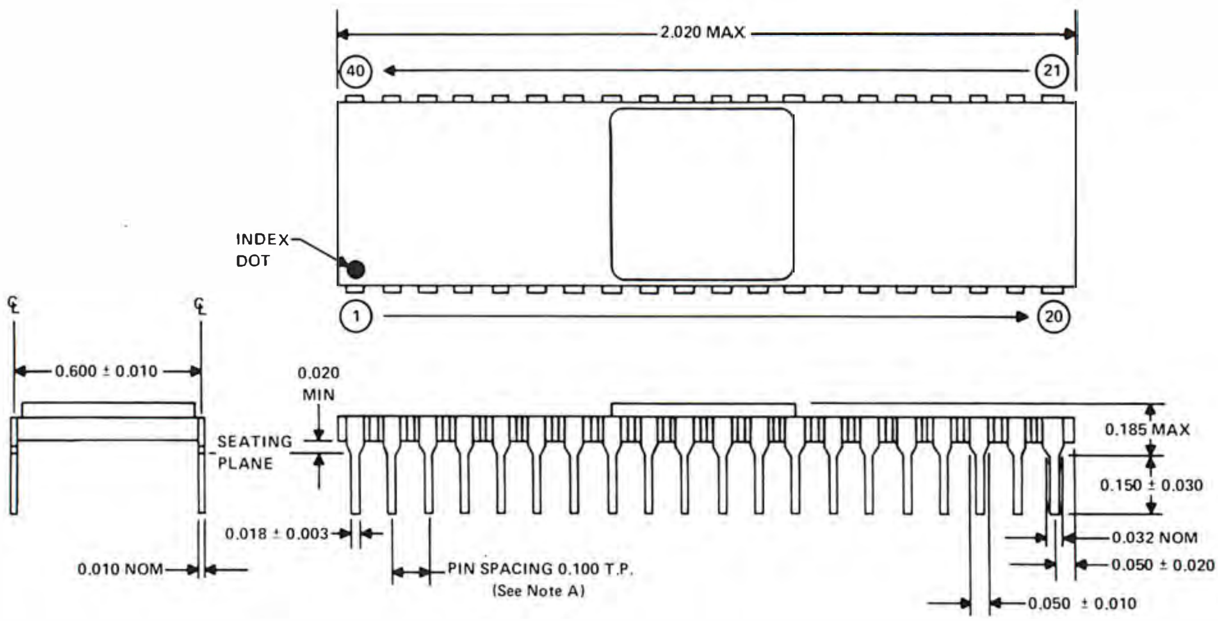
3.6 TERMINAL ASSIGNMENTS

TMS 8080



3.7 MECHANICAL DATA

40-PIN CERAMIC PACKAGE



NOTES: A. The true-position pin spacing is 0.100 between centerlines. Each pin centerline is located within 0.010 of its true longitudinal position relative to pins 1 and 40.

B. All dimensions are in inches unless otherwise noted.

APPENDIX E

APPENDIX E

How to Align the Intecolor® 8001.

CONTENTS

1.0 SAFETY PRECAUTIONS

WARNING: The following precautions should be observed:

1. Do not install, remove, or handle the picture tube in any manner unless shatter-proof goggles are worn. People not so equipped should be kept away while picture tubes are handled. Keep picture tube away from the body while handling.

2. Part of the High Voltage is connected to the AC line directly. This circuitry, found on the Analog Module (100047), is isolated from the remainder of the circuitry by optical isolator, U3, and driver transformer, T101. Should service of the High Voltage be required it is recommended that an isolation transformer be inserted in the power line between the Intecolor® 8001 and the AC supply before any service is performed. When the Chassis must be operated directly from the AC supply, the power plug should always be inserted in the correct polarity to connect the High Voltage common (emitter of Q5) to the ground side of the AC line. Check with a VOM or oscilloscope to see if a potential exists between this point and a known earth ground. A zero reading should be obtained. If any voltage reading is obtained, reverse the power plug and recheck for zero meter reading.

3. When service is required, observe the original lead dress. Extra precaution should be given to assure correct lead dress in the high voltage circuitry and video area. Where a short circuit has occurred, replace those components that indicate evidence of overheating. Always use the manufacturer's recommended replacement component.

1.0.1 HIGH VOLTAGE

NOTE: THE NOMINAL HIGH VOLTAGE FOR THE INTECOLOR® 8001 17" or 19" TERMINAL IS 25 KV. THE HIGH VOLTAGE MUST NOT, UNDER ANY CIRCUMSTANCES, EXCEED 27.5KV.

Each time a terminal's High Voltage requires servicing, measurements should be made at normal viewing settings of the Brightness Control. This will afford assurance that;

1. The High Voltage is within limits specified.
2. The High Voltage regulation circuit is functioning properly.
3. X-Radiation is at a minimum.

If the High Voltage measures abnormally high or the High Voltage Regulation Circuit is not functioning properly, the Terminal should be restored to normal operation through service or adjustments. (See 2.0.5 for High Voltage Adjustment procedure.)

IT IS IMPORTANT TO USE AN ACCURATE AND RELIABLE HIGH VOLTAGE METER.

1.0.2 X-RADIATION PRECAUTIONS

The primary source of X Radiation in this Terminal is the picture tube.

The tube utilized for the above mentioned function in the terminal is specifically constructed to limit X-Radiation emissions.

For continued X-Radiation protection, the replacement tube must be the same type as the original, including suffix letter, or an ISC approved type.

2.0 INSTALLATION AND SERVICE ADJUSTMENTS

2.0.1 SERVICING PRECAUTIONS

Purity, Color, Temperature, and Convergence adjustments for the Intecolor® 8001 are essentially the same as for conventional shadow mask color tubes. Certain precautions should be taken, however, in servicing the Intecolor® 8001 terminal.

Some precautions to observe while servicing the solid state chassis are listed below:

1. Always connect the ground lead of a test instrument to the chassis before connecting the positive lead; conversely, always remove the ground lead of a test instrument last.

2. Do not check for high voltage by drawing an arc. Use a high voltage meter or a high voltage probe with a VOM.

3. Do not bridge electrolytic capacitors since resultant surges may damage solid state devices.

4. Some transistors are equipped with heat sinks. Do not operate the transistor with the heat sink removed.

5. All soldering irons used where transistors and integrated chips are concerned should be 35 watt (6 volts) irons and grounded in such a way that no voltage will be applied to the solid state device during the soldering operation. This precaution is to prevent possible damage to the device due to excessive heat or voltage applied under no bias conditions.

6. When servicing the video circuitry it is recommended that an oscilloscope of at least 100 MHZ bandwidth, such as the Tektronix 454A, be used.

2.0.2 AC LINE TAP SELECTOR

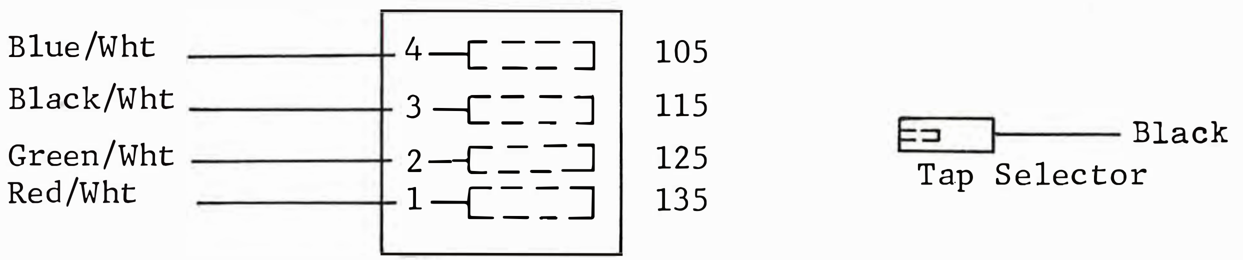
The AC Line Tap Selector is located inside the chassis on the right hand side as viewed from the rear (See Figure 2.0.2.1). In areas having a 115VAC line supply, this tap should be left in the 115 VAC position. Other taps are shown depending on the line voltage.

2.0.3 VERTICAL DEFLECTION

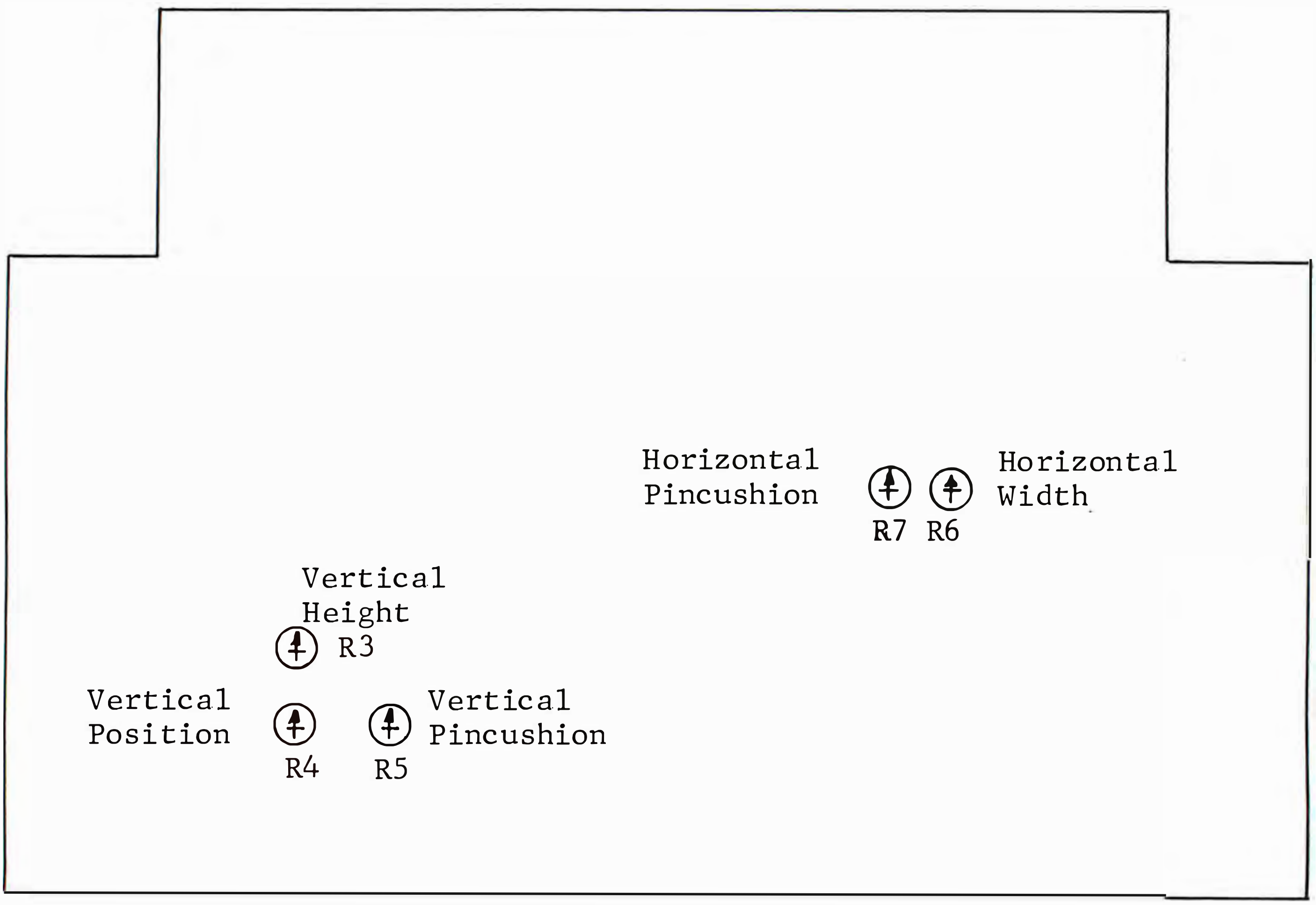
At 115 volts line voltage adjust the VERTICAL HEIGHT CONTROL, R3, (See Figure 2.0.3.1) and the VERTICAL POSITION CONTROL, R4, so that the picutre is centered and there is a 12" wide by 10" high display. A suitable display is found by filling up the screen with a single character or erasing the screen with a background color.

2.0.4 HORIZONTAL DEFLECTION

Adjust the HORIZONTAL WIDTH CONTROL, R6, (Analog Module, 100047) (See Figure 2.0.3.1) so that the picture has a 12" wide by 10" high display. HORIZONTAL CENTERING is accomplished by adjusting R3 on the rear edge of the Display Generator Card, 100117. Adjusting the Pot R3 causes one character movements to the right or left of the screen.



AC LINE TRANSFORMER TAP SELECTION
FIGURE 2.0.2.1



ANALOG MODULE (100047) PRINTED CIRCUIT BOARD BOTTOM VIEW
FIGURE 2.0.3.1

2.0.5 HIGH VOLTAGE ADJUSTMENT

Preset High Voltage Adjustment Control R8 (Analog Module 100047) to 1/2 clockwise, and Brightness Control R17, to maximum counterclockwise (minimum brightness).

Remove the High Voltage Anode Cap from the tube and connect a Pomona #2900A or equivalent to the High Voltage Cap. CAUTION: BE SURE HV PROBE GROUND IS GROUNDED. INSURE THAT ANODE CAP IS ISOLATED FROM ALL PERSONS AND EQUIPMENT. Adjust High Voltage Control, R8 for 25 KV.

2.0.6 FOCUS ADJUSTMENT

Create a full page of white dots on the CRT screen by utilizing the following procedure:

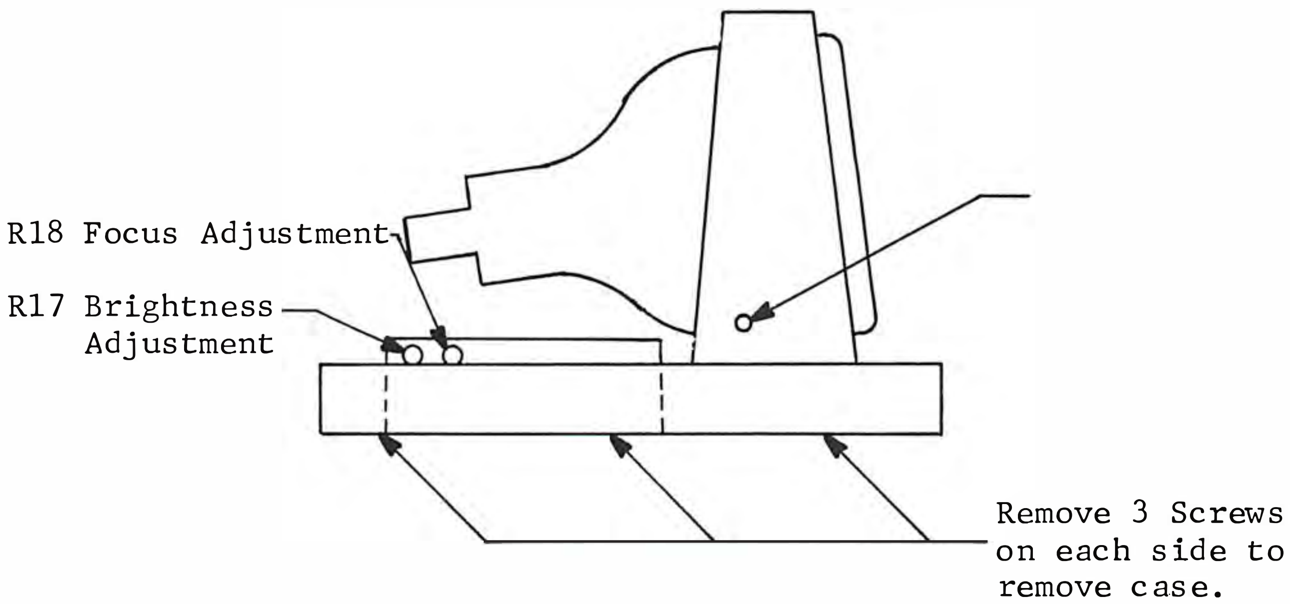
1. Select Foreground Color - WHITE
2. Select Background Color - BLACK
3. Press keyboard "." (period) and allow to repeat until screen is full of white dots.

Adjust the FOCUS pot (found on the right side (viewing from rear) of the Analog Card mounting bracket. Remove the external case with 6 screws) for optimum focus over the entire screen. (See Figure 2.0.6.1)

2.0.7 PURITY ADJUSTMENT

The Intecolor® 8001 should always be facing either north or south during purity adjustment. This assures that any effect of the earth's normal magnetic field upon beam landing will be negligible when the terminal is placed in its normal viewing location.

The instrument should be at room temperature (60°F or above) for at least 30 minutes before set-up adjustments are made. Allow a minimum of ten minutes operation at high beam current (brightness full without bloom) before attempting purity or convergence adjustments.



FOCUS AND BRIGHTNESS ADJUSTMENT LOCATION

FIGURE 2.0.6.1

Should any parts of the chassis become magnetized, it will be necessary to manually degauss the affected areas. Move a manual (GC 9317 or equivalent) degaussing coil slowly around those areas and the face of the CR Tube and slowly withdraw to a distance of six feet before disconnecting the coil from the AC power source.

Before performing the purity adjustments, the center of the raster must be converged and the dynamic convergence set roughly as explained in Section 2.0.12. Check that the focus control is properly set (See Section 2.0.6). The focus adjustment should be made with the brightness control set at maximum beam current without bloom.

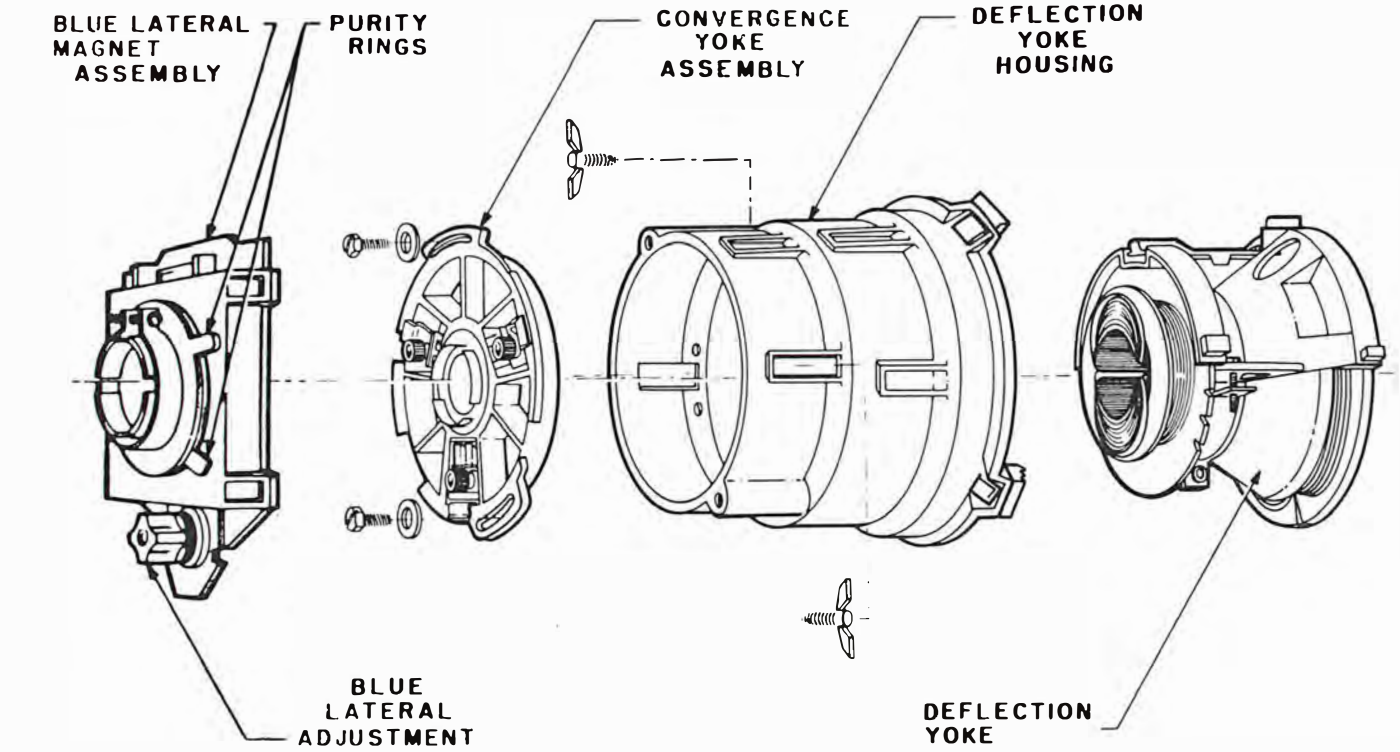
1. Purity adjustments are most accurate while observing one screen only, preferably red. Erase the screen with the background color "RED".

2. Loosen the yoke wing nuts and move the yoke to the rear as far as possible. (See Figure 2.0.7.1)

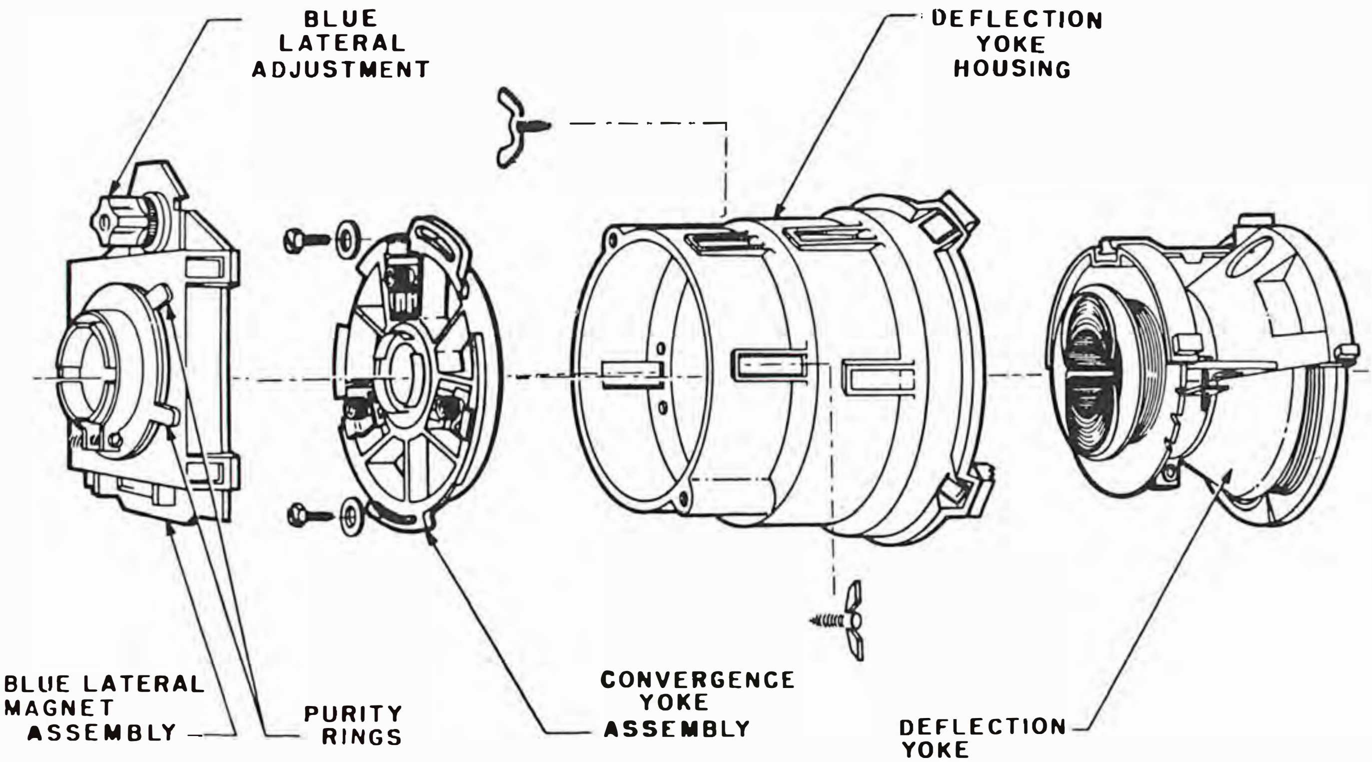
3. Rotate the purity magnets and adjustment tabs so that a clean red area is produced at the center of the screen. Push the yoke forward until a uniform red raster is obtained. Tighten the yoke wing nuts.

4. Erase the screen with the background color "WHITE". Check for a uniform white screen (see COLOR TEMPERATURE ADJUSTMENTS, Section 2.0.8, for procedure). If uniformity has not been obtained, reconverge the center of the screen and repeat the purity adjustments.

5. It should be noted that purity adjustments also affect the focus and DC Horizontal and Vertical screen positions and these parameters may have to be readjusted as outlined under Sections 2.0.3, 2.0.4, and 2.0.6.



17" SCREEN



19" SCREEN

YOKE; BLUE LATERAL, AND PURITY

LOCATIONS AND ADJUSTMENTS

FIGURE 2.0.7.1

2.0.8 COLOR TEMPERATURE ADJUSTMENTS

1. Place a screen full of WHITE characters or ERASE the screen in WHITE. Turn the screen grid drive controls R14 (RED), R15 (GREEN), R16 (BLUE) (Analog Module 100047) to minimum drive (Fully CCW) then turn the BRIGHTNESS Control, R17 to maximum brightness (Fully CW).

2. Turn the RED control, R14, clockwise until the red vertical retrace raster line at the top of the screen is just visible. Turn the GREEN Control, R15, clock- (R-16) wise until the green vertical retrace raster line at the top of the screen is just visible. Repeat the same for the BLUE Control, R16. (R-15)

3. Adjust the BRIGHTNESS Control, R17, until there is no visible vertical retrace raster line and the brightness is at a comfortable viewing level with a minimum of color saturation.

4. Adjust each screen grid drive control, RED (R14), GREEN (R15), and BLUE (R16), until a white screen is obtained, or a 9300°K color temperature (WHITE).

2.0.9 TOP, BOTTOM, AND SIDE PINCUSHION ADJUSTMENT

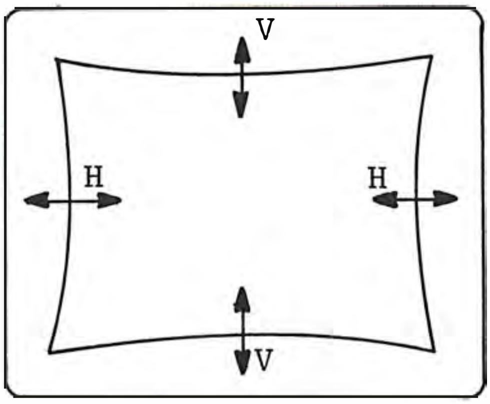
Place a suitable test pattern on the screen such as all "+" (plus) symbols or all "." (periods). (See Section 2.0.6 for pattern set-up). Any color or WHITE may be used.

The top and bottom (Vertical) pin cushion adjustment is made, if necessary, by adjusting R5 on the Analog Module (100047) for straight horizontal lines at the top and bottom of the raster as shown in Figure 2.0.3.1 and Figure 2.0.9.1.

The side (Horizontal) pin cushion adjustment is made by adjusting R7 on the Analog Module (100047) for straight vertical lines on the left and right side of the raster.

V= Vertical
Pincushion= R7

H= Horizontal
Pincushion= R5

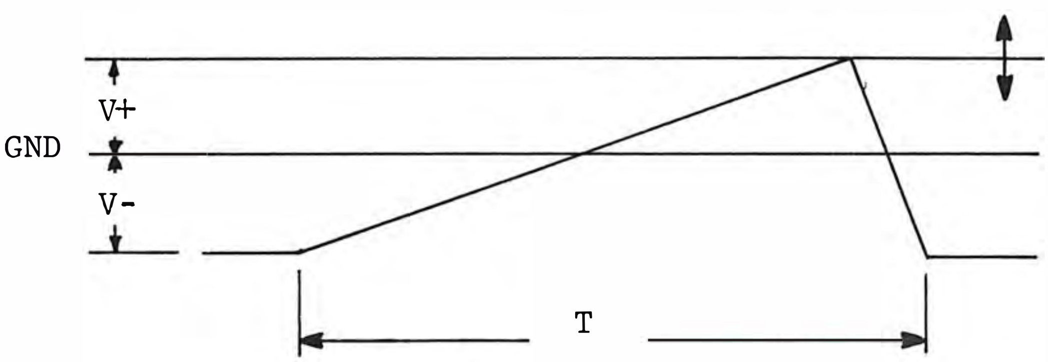


PINCUSHION ADJUSTMENT

FIGURE 2.0.9.1

U11/6= Horizontal=R1

U10/6= Vertical=R2



HORIZONTAL AND VERTICAL RAMP ADJUSTMENTS
ANALOG MODULE (100047)

FIGURE 2.0.10.1.1

2.0.10 CONVERGENCE ADJUSTMENT PRELIMINARIES

The CONVERGENCE ADJUSTMENT PRELIMINARIES are necessary only if convergence cannot be obtained as outlined under FINAL CONVERGENCE ADJUSTMENTS (Section 2.0.12), or if these areas have required service or parts replacements, or the adjustment pots have been tampered with. An oscilloscope, such as the Tektronix 454, or equivalent will be necessary for these adjustments.

2.0.10.1 PRELIMINARY HORIZONTAL RAMP ADJUSTMENT

The Horizontal Ramp U11/6 amplitude is adjusted by R1 on the Analog Module (100047). The ramp is adjusted so that the positive peak is equal in height to the negative peak (symmetrical about ground or V+ = V-). (See Figure 2.0.10.1.1).

2.0.10.2 PRELIMINARY VERTICAL RAMP ADJUSTMENT

The VERTICAL RAMP U10/6 amplitude is adjusted by R2 on the Analog Module (100047) in the same manner as the HORIZONTAL RAMP ADJUSTMENT (See Figure 2.0.10.1.1).

2.0.10.3 PRELIMINARY HORIZONTAL PARABOLA ADJUSTMENT (U7/3) RIGHT & LEFT CENTER, TUBE AREAS 4 & 5 (See Figure 2.0.12.2).

Adjust R9 on the Analog Module (100047) until the bottom of the Parabola is at Ground level. See Figure 2.0.10.3.1.

2.0.10.4 PRELIMINARY VERTICAL PARABOLA ADJUSTMENT (U8/3) TOP & BOTTOM CENTER, TUBE AREAS 2 & 3 (See Figure 2.0.12.2).

Adjust R10 on the Analog Module (100047) until the bottom of the Parabola is at ground level. See Figure 2.0.10.3.1.

2.0.10.5 HORIZONTAL AND VERTICAL RAMP ADJUSTMENTS.

Monitor the HORIZONTAL PARABOLA at U7/3 on the Analog Module (100047). Superimpose a small amount of the video signals (with a screen full of WHITE characters) by adding a small amount of the "B" trace (connect a scope probe to the collector of Q26 or Q27 or Q28) on the oscilloscope (CHOP, INVERT B, ADD) to the "A" trace (connected to U7/3). The above may also be accomplished by simply connecting the "A" channel Scope ground to a ground in the vicinity of Q26, Q27, or Q28. The video will be apparent on the parabola, as shown in Figure 2.0.10.5.1.

Adjust R1 until the superimposed video is as shown in Figure 2.0.10.5.1.

Monitor the VERTICAL PARABOLA at U8/3 and adjust R2 of the Analog Module (100047) until the end points of the parabola are equal in height.

The above procedure is shown in Figure 2.0.10.5.2.

2.0.10.6 VACANT

2.0.10.7 CORNER PARABOLA ADJUSTMENTS
TUBE AREAS 6, 7, 8, and 9 (See Figure 2.0.12.2)

The CORNER PARABOLA ADJUSTMENTS are made by R11, R12 and R13 on the Analog Module 100047 and monitoring the waveform as shown at U14/3 as in Figure 2.0.10.7.1. OFFSET is adjusted to zero by R13 by adjusting the waveform baseline to ground as shown in Figure 2.0.10.7.1, Waveform A.

BASELINE SLANT is adjusted by R12 on Analog Module (100047) as shown in B of Figure 2.0.10.7.1. Adjust for V_{SC} equal to "0" volts.

VERTICAL SYMMETRY is adjusted as shown in C of Figure 2.0.10.7.1 using R11 on Analog Module (100047). Alignment is made by adjusting R11 until $+V_{HC} = -V_{HC}$.

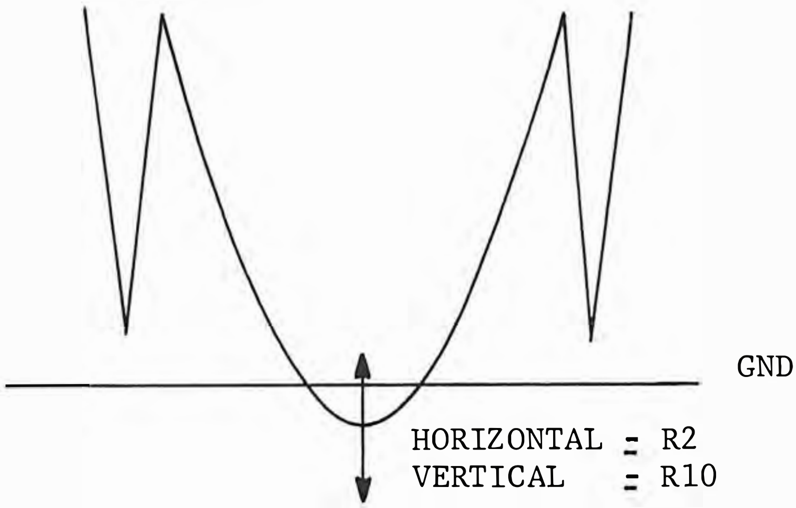
2.0.10.8 HORIZONTAL, VERTICAL and CORNER PARABOLA TOUCH-UP

Touch up of the HORIZONTAL, VERTICAL, and CORNER PARABOLAS can best be accomplished by monitoring the waveforms on the J1 on the Convergence Module (100014).

A. Adjust the HORIZONTAL PARABOLA offset, V_{HP} with R9 on the Analog Module (100047) by monitoring the waveform at J1/1 on the Convergence Module (100014) as shown in Figure 2.0.10.8.1, A.

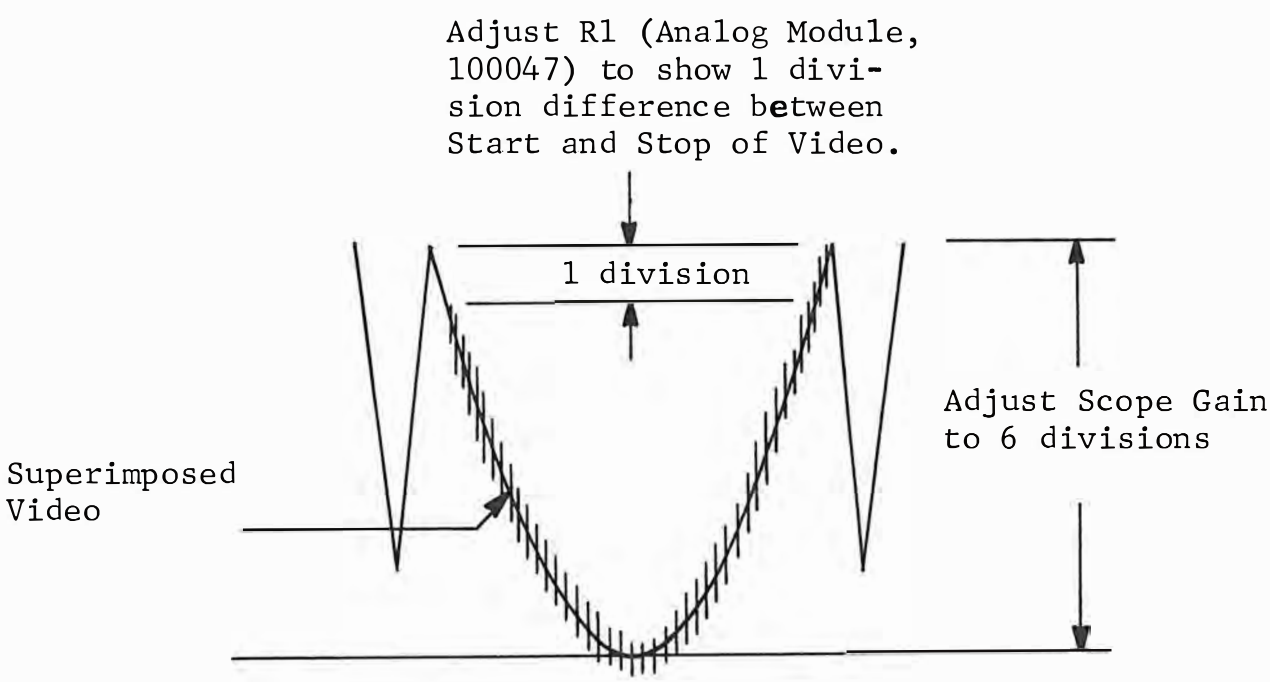
B. Adjust the VERTICAL PARABOLA offset, V_{VP} with R10 on the Analog Module (100047) by monitoring the waveform at J1/5 on the Convergence Module (100014) as shown in Figure 2.0.10.8.1, B

C. Adjust the CORNER PARABOLA offset, V_{CP} with R13 on the Analog Module (100047) by monitoring the waveform at J1/7 on the Convergence Module (100014) as shown in Figure 2.0.10.8.1, C.

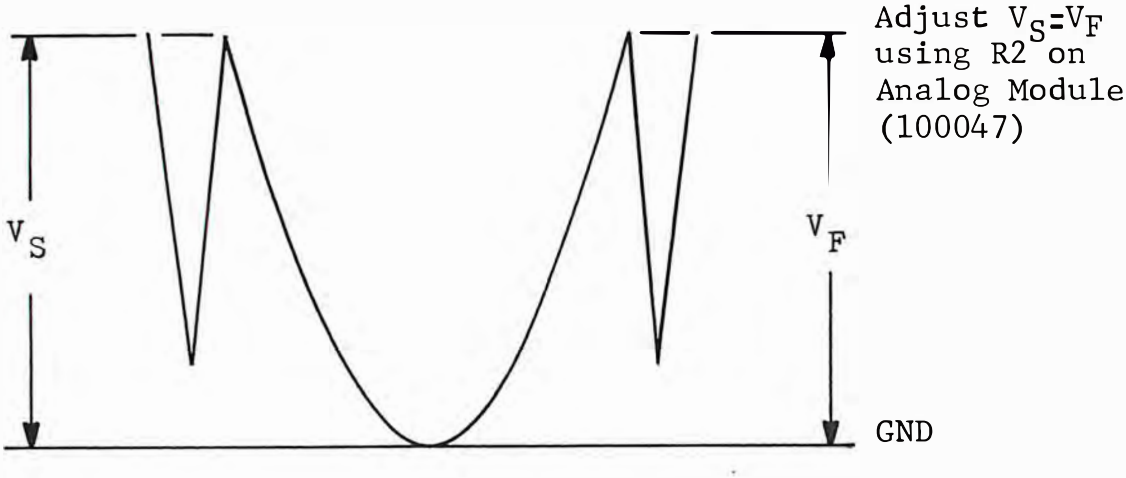


HORIZONTAL AND VERTICAL PARABOLA ADJUSTMENTS

FIGURE 2.0.10.3.1



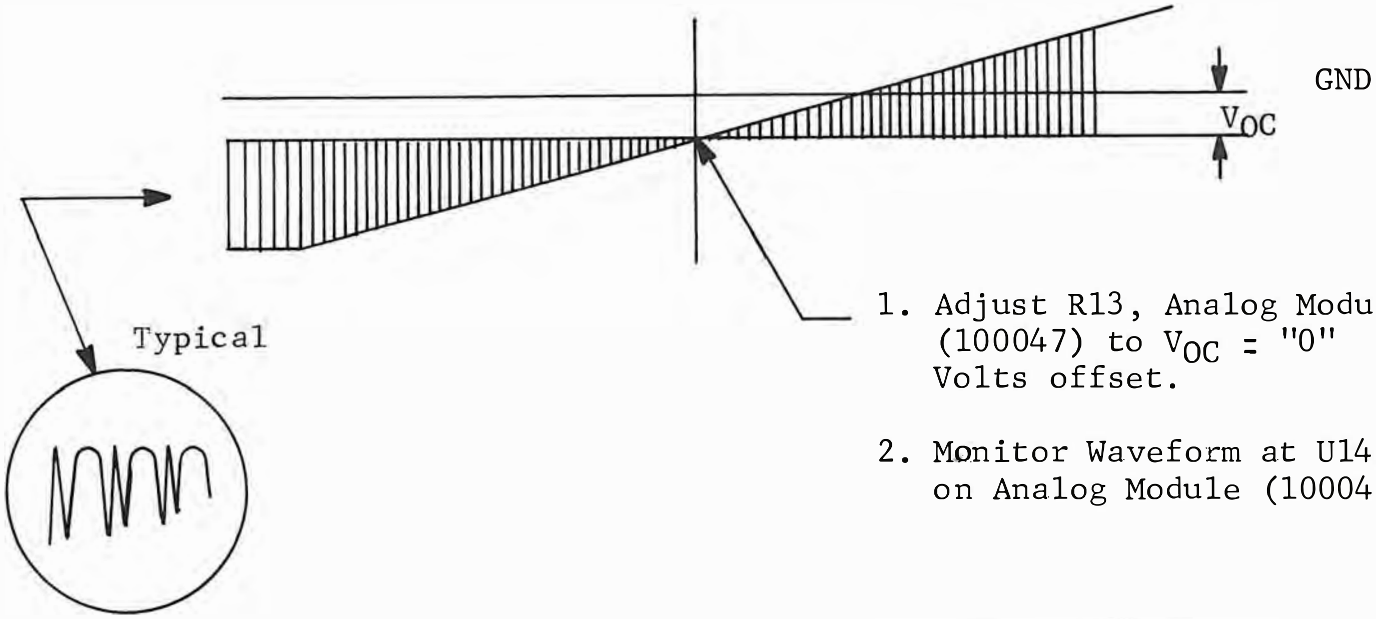
HORIZONTAL PARABOLA VIDEO ADJUSTMENT
FIGURE 2.0.10.5.1



VERTICAL PARABOLA HEIGHT ADJUSTMENTS

FIGURE 2.0.10.5.2

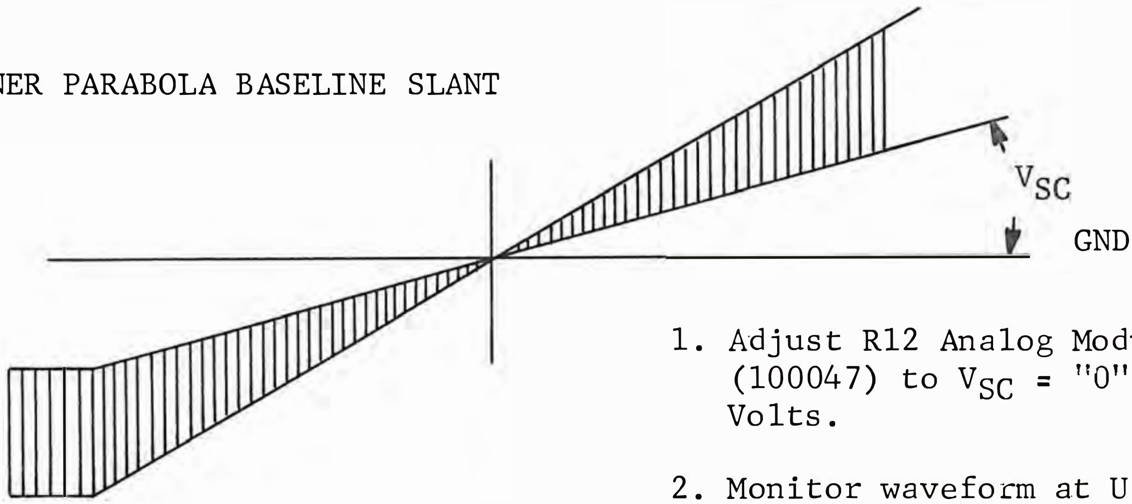
A. CORNER PARABOLA OFFSET



1. Adjust R13, Analog Module (100047) to V_{OC} = "0" Volts offset.

2. Monitor Waveform at U14/3 on Analog Module (100047)

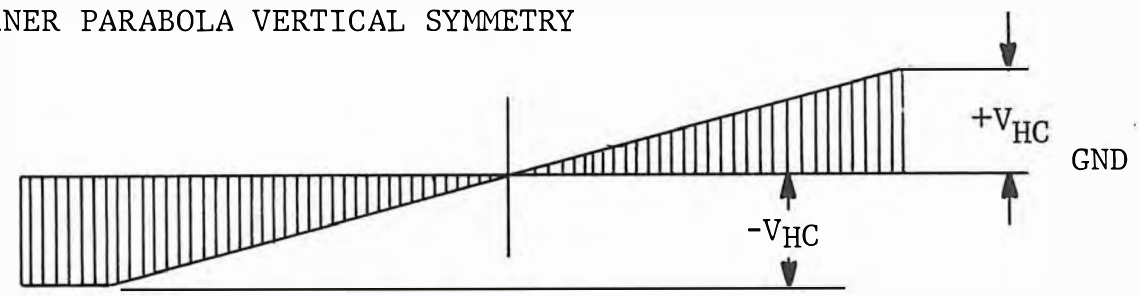
B. CORNER PARABOLA BASELINE SLANT



1. Adjust R12 Analog Module (100047) to V_{SC} = "0" Volts.

2. Monitor waveform at U14/3 on Analog Module (100047)

C. CORNER PARABOLA VERTICAL SYMMETRY

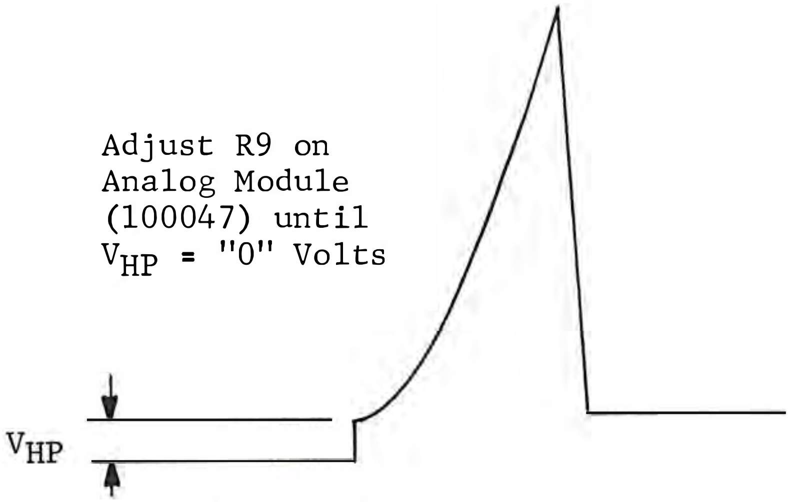


1. Adjust R11 Analog Module (100047) to $+V_{HC} = -V_{HC}$

2. Monitor waveform at U14/3 on Analog Module (100047)

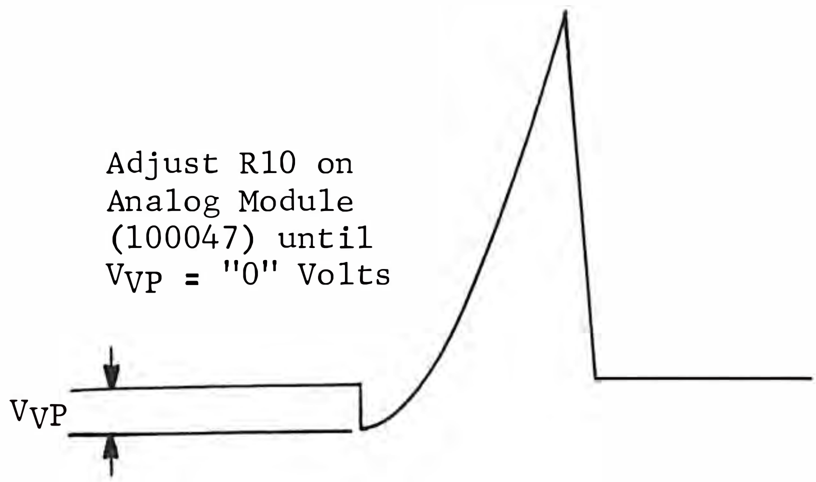
CORNER PARABOLA ADJUSTMENTS

FIGURE 2.0.10.7.1



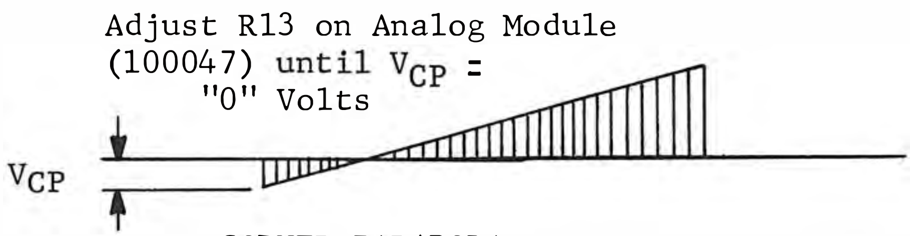
HORIZONTAL PARABOLA
J1/1 ON CONVERGENCE MODULE (100014)

A.



VERTICAL PARABOLA
J1/5 ON CONVERGENCE MODULE (100014)

B.



CORNER PARABOLA
J1/7 ON CONVERGENCE MODULE (100014)
C.

HORIZONTAL, VERTICAL, AND CORNER PARABOLA TOUCH-UP

FIGURE 2.0.10.8.1

2.0.11 CONVERGENCE STATIC ADJUSTMENTS

Place a dot pattern on the screen in the following manner from the Keyboard.

Define FOREGROUND COLOR AS "WHITE"
BACKGROUND COLOR AS "BLACK"

Depress "." (period) Key and allow to repeat until the screen is full of white dots.

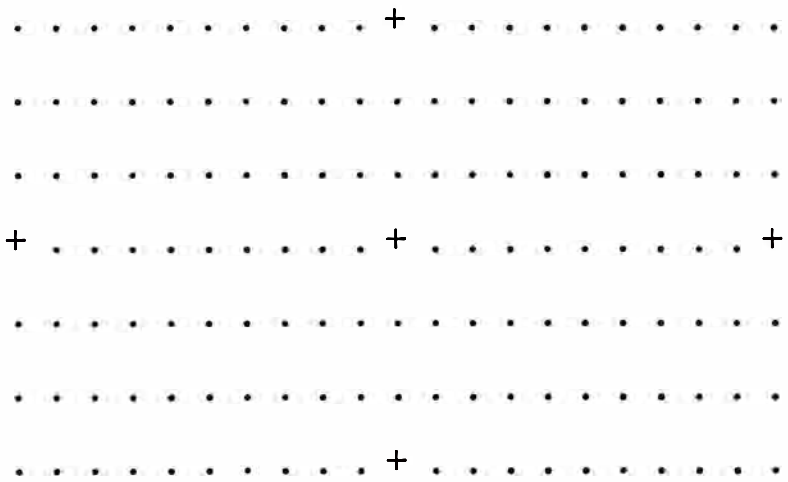
The above will fill up the screen with dots. Now place "+" symbols utilizing the keyboard as shown in Figure 2.0.11.1

Turn all the pots on the Convergence Module (100014) to the straight up position as shown in Figure 2.0.11.3.

Now adjust the static magnets and the Blue Lateral Magnet to align the "+" symbols R,G,B, colors in Screen Sector 1, as shown in Figure 2.0.11.2, so as to appear as "WHITE". This will occur when the RED, GREEN, AND BLUE colors are accurately superimposed on top of each other. With the exception of BLUE lateral which is explained below.

For the above to be accurate the tube must have been externally degaused, the Purity adjusted, the FOCUS R18 adjusted for sharp, and the BRIGHTNESS, R17, Analog Module (100047), set for a low level with the color temperature being set to 9600°K as explained in previous sections. DO NOT ATTEMPT FURTHER CONVERGENCE UNLESS THE ABOVE HAS BEEN PREVIOUSLY PERFORMED. (See Sections

The beams move at approximately the same angle as the convergence magnets are offset from the vertical plane. Blue, since it is mounted in the vertical plane moves



CONVERGENCE TEST PATTERN

FIGURE 2.0.11.1

the beam up and down vertically; red and green move the respective beams on a line at about a 60° angle from the vertical. The blue lateral magnet moves all three beams in the horizontal plane, the blue beam in one direction and the red and green beams in the opposite direction in a 5 to 1 ratio. The blue beam has the greatest lateral shift.

The thumb screw adjustment of red, green, and blue center convergence magnets can be rotated in either direction continuously. Flux change is accomplished by rotating the pole position of the magnets, not by moving the magnets farther from or closer to the respective guns.

Adjust the Static Blue so that the Blue in the center of the screen is superimposed on the RED and GREEN.

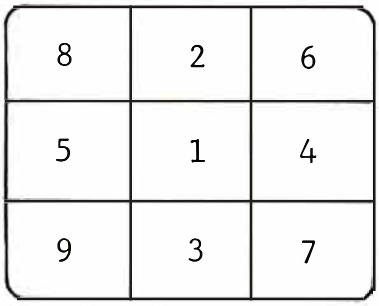
2.0.12 FINAL CONVERGENCE

Touch up the center convergence with the pots R13 (GREEN), R14 (RED) and R15 (BLUE) on the Convergence Module (100014) as shown in Figure 2.0.11.2 and Figure 2.0.11.3.

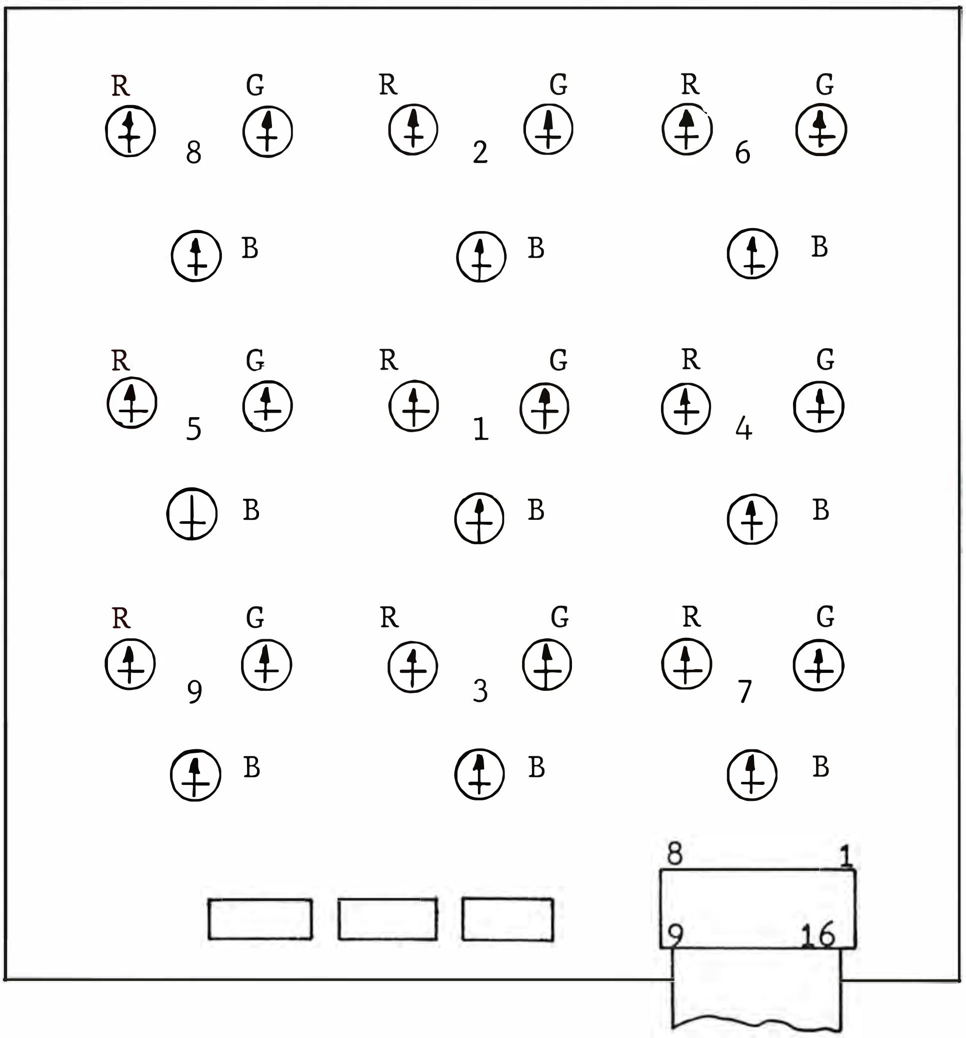
Once center convergence has been adjusted proceed to the next convergence Screen Sector, 2, as shown in Figure 2.0.11.2. Proceed with the alignment in the order of the sector numbers as shown in Figure 2.0.11.2. After each Sector is aligned, check and touch up the center convergence. Note that the adjustment pots on the Convergence Module (100014) are arranged in the same location as each Screen Sector as viewed on the tube face (and the component side of the board) and the trio of pot groups in each sector are arranged as GREEN, RED, AND BLUE corresponding to the location of the GREEN, RED, and BLUE electron beams as viewed from the tube face.

When completed with the above, touch up each Screen Sector as needed in the SAME ORDER as outlined above. Do not violate the order of the Screen Sector numbers in the adjustment procedure.

Never attempt a convergence procedure without first setting the Convergence Module (100014) pots to the center position as shown in Figure 2.0.12.3 and following the Screen Sector numbers. It is seldom necessary for the static magnets to be adjusted unless shipment vibration causes convergence coil or static magnet movements or unless convergence coil or yoke replacements become necessary.



CR TUBE CONVERGENCE SECTORS (SCREEN VIEW)
FIGURE 2.0.11.2



CONVERGENCE BOARD ASSEMBLY SHOWING
CONTROLS ASSOCIATED WITH TUBE SECTORS
(TOP VIEW)
NOTE: Green and Red Pots are interchanged on all 17" Tubes.
FIGURE 2.0.12.3

The CPU Operating System

TABLE OF CONTENTS

--TERMS AND ABBREVIATIONS--

TERMS:

TERM	DESCRIPTION
Address	A 16 bit number assigned to a memory location corresponding to its sequential position.
Bit	The smallest unit of information which can be represented. (A bit may be in one of two states, 0 or 1).
Byte	A group of 8 contiguous bits occupying a single memory location.
Console	Refers to the 8001 CRT Display as the output device, and the 8001 keyboard as the input device. Allows operator interface with the CPU operating system.
Instruction	The smallest single operation that the computer can be directed to execute.
Object Program	A program which can be loaded directly into the computer's memory and which requires no alteration before execution. An object program is usually on paper tape, and is produced by assembling (or compiling) a source program. Instructions are re-presented by binary machine code in an object program.
Program	A sequence of instructions which, taken as a group, allow the computer to accomplish a desired task.
Source Program	A progam which is readable by a programmer but which must be transformed into object program format before it can be loaded into the computer and executed. Instructions in an assembly language source program are represented by their assembly language mnemonic.
System Program	A program written to help in the process of creating user programs.

TERMS -- (Continued):

TERM	DESCRIPTION
User Program	A program written by the user to make the computer perform any desired task.
Word	A group of 16 contiguous bits occupying two successive memory locations. (2 bytes).

ABBREVIATIONS:

ABBREVIATION	DESCRIPTION
Cr	Carriage return
CPU	Central Processing Unit
Lf	Line feed
PROM	Programmable Read Only Memory
Sp	Space Bar
nnn B	nnn represents a number in binary format.
nnn D	nnn represents a number in decimal format.
nnn O	nnn represents a number in octal format.
nnn Q	nnn represents a number in octal format.
nnn H	nnn represents a number in hexadecimal format.
	Shaded portions of CPU/operator dialog represent Console output.

CPU O.S. COMMANDS AND MESSAGES

2.0 CPU OPERATING SYSTEM (O.S.)

STARTING ADDRESS - 100 When in 8708 ERASABLE PROM

4 - STARTS AT EØØØ, PAUSES AT FFFF
ØØ-Ø1
Ø1-Ø2
Ø2-Ø3

All arguments are in hexadecimal form.

A RAM TEST 2,3,4?

B GO BACK TO CRT O.S.

D DISPLAY IN HEXADECIMAL FORMAT

D low address, high address

Memory from low address to high address is displayed in hexadecimal form.

E END

E address

Endfile mark is created; 60 null characters are written on punch device

F FILL MEMORY

F low address, high address, data

Memory from low address to high address is filled with data.

G GO TO

G Address, bkptl, bkpt2

Program control is transferred to address. Breakpoints are set at bkptl and bkpt2. When break points are executed, all of the CPU registers are automatically displayed.

H HEXADECIMAL ARITHMETIC

H number, number sp

The sum and difference of the two numbers is printed in hexadecimal.

L LOAD HEXADECIMAL TAPE

L Bias address

A hexadecimal format tape is read into memory at tape address plus bias address.

M MOVE

M low address, high address, destination address

A block of memory from low address to high address is moved to location destination address.

N PUNCH NULL

N

Sixty null characters are punched.

R BAUD RATE FOR SECOND RS-232 CHANNEL

R rate number

The rate number must be between 1 and 7. See the "How to Use the 8001" Manual.

S SUBSTITUTE

S address Sp

Memory at address is displayed, and can be modified by typing in new data. Termination with space opens next sequential address, termination with carriage return ends command.

X EXAMINE REGISTERS OR MEMORY

X reg ident

Register is displayed, and can be modified as in the S command.

W WRITE HEXADECIMAL

W low address, high address

Memory from low address to high address is punched in hexadecimal format.

MESSAGES

. CPU O.S. ready to accept commands

? Error. Reenter command

3.0 INTECOLOR® 8001 CONFIGURATION

3.1 I/O SYSTEM

The Intecolor® 8001 can support a number of input/output devices, from the CRT display and the RS232C I/O to devices supplied by the user. In general, it may be convenient to have two devices which can perform the same function, but to use them for different purposes at various times. For example, if a program is being assembled, you might want the program listing to be written on one device, while any system messages not relevant to the assembly would be written on a separate device.

The I.O system described below permits this type of change. Devices may be assigned functions via the System Monitor S command (see Section 4.2.11) or via the user's program. That is, it is possible to write programs which read from several different input devices and write to several different output devices of the program's choosing, without requiring any human intervention.

3.1.1 LOGICAL AND PHYSICAL DEVICES

Regardless of how many I/O devices a particular Intecolor® 8001 has, there are only four operations which can be performed to any of them. For example, a WRITE operation can be performed either to the RS232C channel 1 to a host computer or a high speed tape system. All system programs and user-written programs, therefore, access four LOGICAL DEVICES (i.e., a WRITE device) which are then translated to a PHYSICAL DEVICE (i.e., a high speed tape) by the I/O system.

The four logical devices available to programs are:

CONSOLE	An interactive, character-oriented device used for both input and output.
READER	A character-oriented, input-only device which transfers data on command and signals the program when where is no more data (an end-of-file condition).
WRITE	A character-oriented, output-only device which accepts a character from the program and records it on some external medium.
LIST	A character-oriented, output-only device which accepts a character from the program and records it on some external medium in human readable form.

9F9Ø = -24688

Each of these four logical devices may be associated with one of four physical devices at any instant, giving a total of 16 physical devices. The mapping from logical to physical devices is specified by an I/O status byte which resides in memory and is accessible to system and user programs via substitute command. The address of the I/O status byte is 9F9Ø hex. A pointer to the I/O status byte is also contained in memory locations ØØ36 and ØØ37 (low byte of pointer, high byte of pointer). The possible mappings appear as follows:

I/O Status Byte:
Initially

A_7 A_6	A_5 A_4	A_3 A_2	A_1 A_0
10	00	00	10

A_7A_6 = LIST FIELD A_1A_0= CONSOLE FIELD

A_5A_4 = PUNCH FIELD A_3A_2= READER FIELD

LOGICAL DEVICES	I/O DEV FIELD	PHYSICAL DEVICES
CONSOLE	00	RS232 Channel 1
	01	RS232 Channel 2
	10	CR Tube = Console Output Keyboard= Console Input
	11	(user console device)
READER	00	RS232 Channel 1
	01	RS232 Channel 2
	10	Keyboard
	11	(user reader device 1)

LOGICAL DEVICES	I/O DEV FIELD	PHYSICAL DEVICES
WRITE	00	RS232 Channel 1
	01	RS232 Channel 2
	10	CR Tube
	11	(user punch device 1)
LIST	00	RS232 Channel 1
	01	RS232 Channel 2
	10	CR Tube
	11	(user list device 1)

At cold start or system reset, the I/O status byte is set equal to 82H, causing the CR Tube and keyboard to be selected for console I/O and LIST, while the RS232 Channel 1 is selected for both READ and WRITE.

3.1.2 I/O SUBROUTINES

The way in which a program performs an I/O operation to any of the four logical devices is by calling the appropriate subroutine supplied by the I/O system. The available subroutines and their locations in memory are given in the following table:

ROUTINE	FUNCTION	MEMORY LOCATION
CI	Console input	103H
CO	Console Output	109H
RI	Reader input	106H
PO	Punch output	10CH
LO	List output	10FH
SO	Console String Output	12AH

The rest of this section gives a description and examples of how to call these subroutines.

CI - CONSOLE INPUT

This routine returns a character received from the selected console device to the caller in the A register. The A register and the condition bits are affected by this operation.

Example:

Assembly Language

```
...
CALL    CI
STA     DATA
...
```

CO - CONSOLE OUTPUT

CO transmits a character, passed from the calling program in the A register, to the device selected for console output. The A register and the condition bits are affected.

Example:

Assembly Language

```
...
MVI     A,'.'
CALL    CO                  ;PRINT '.' ON CONSOLE
...
```

RI - READER INPUT

RI returns a character read from the reader device in the A register. If no character was read from the device (i.e., end of file), the CARRY condition bit is set equal to 1, and the A register is zeroed. If data is ready, the CARRY bit is zeroed. If no character is received from the physical device then striking any key causes an end of file to be simulated and control is returned to the calling program.

Example:

Assembly Language

```
...
CALL    RI
JC      EOF                 ; END OF FILE SENSED
STA     DATA
...
```

PO - WRITE OUTPUT

PO transmits a character from the calling program to the device selected as the punch device. PO is identical in format to CO.

LO - LIST OUTPUT

LO performs the same function to the selected list device as CO and PO do to their selected devices.

SO - CONSOLE STRING OUTPUT

SO transmits a character string to the device selected for console output. A pointer to the beginning of the string is passed from the calling program in the HL register pair. The string should be terminated by a byte having the value 239 (decimal). SO also provides repeat loops of the form: ..., 237, N, D1, D2, ..., DM, 238, ... where N is the repeat count for the string of bytes D1 through DM.

Example:

Assembly Language

```
             ...
             LXI      H, STR
             CALL     SO
             ...
    STR:  DB   'AB', 237, 3, 'CD', 238, 'EFG', 239
```

This example will print 'ABCDCDCDEFG' on the console device.

FLOPPY TAPE I/O SUBROUTINES

Three I/O subroutines are provided for the Intecolor Floppy Tape. These routines are:

ROUTINE	FUNCTION	MEMORY LOCATION
TWR	Write to Floppy Tape	~~Ø13ØH~~ 686ØH
TRD	Read from Floppy Tape	~~Ø133H~~ 68Ø5H
TVF	Compare memory with Floppy Tape	~~Ø136H~~ 68ØØH

The Floppy Tape is a block-transfer device. One record is written per track. The inputs from the calling program to each of the three I/O routines are:

```
HL register pair - pointer to memory buffer

DE register pair - byte count

A register       - Tape drive/track code:
                   BIT3     -   DRIVE:  Ø or 1
                   BITS2-Ø -   Track:  Ø through 7
```

After calling any one of the routines, the A register will contain a status code and will have been tested (ORA A):

```
A=Ø    :    No Errors
A=2    :    Keyboard Abort (Pressing any key on the
              keyboard during the data transfer will
              abort the operation)
A=4    :    Buffer too large for write.
A=6    :    Buffer too small for read.
A=8    :    Read Failure:  A complete, correctly
              formatted record could not be read
              from the tape.
A=10   :    Checksum error.
A=12   :    Verify failure.  A mismatch was detected between
              data in memory and data read from the tape
              during a memory compare operation (TVF).
```

Also, after calling any of the routines, the HL register pair will point one byte past the last byte manipulated in the memory buffer.

3.1.3 USER-SUPPLIED DEVICES

This section describes the necessary steps in hooking up a user-supplied I/O device to the I/O system.

The I/O subroutines described in Section 3.3.2 assume that programs (called drivers) exist which perform the actual transfer of data between I/O devices and the CPU. For instance, when the console input routine is called, it checks to see which physical device is assigned to the console, and then branches to the driver appropriate to the device Therefore, when the user supplies his own device, he must:

1) Write a program to perform the data transfer, making sure that the program saves and restores any CPU registers it uses that are not specifically changed by the I/O subroutine.

2) Store a JMP to this driver's address in the appropriate location as defined in the following table:

MEMORY LOCATION	USE
9F91H	USER DEFINED CONSOLE INPUT
9F94H	USER DEFINED CONSOLE OUTPUT
9F97H	USER DEFINED READER (1)
9F9AH	USER DEFINED WRITE (1)
9F9DH	USER DEFINED LIST (1)

Thus, if the user supplied a custom built listing device, he would write a driver to transfer data to it in an appropriate manner, then store the JMP to the driver's address at location 9F9DH. By assigning LIST=3, his device would receive any listing output generated.

4.0 CPU OPERATING SYSTEM

The Intecolor 8001 CPU O.S. enables the operator to easily manipulate the contents of memory, read and produce MAG tapes, and execute programs.

The CPU O.S., and all Intecolor® 8001 system software in general, use the last 80 memory locations after the refresh area for storage of temporary data. Therefore, if the operator runs a program beginning in these locations, and then uses the CPU O.S. Text Editor, or Assembler, he must re-load these 80 bytes of his program before running it again. Alternatively, programs could be written beginning at any higher location. Then system programs and user programs could be executed in any order, without requiring the re-load operation.

For a 25 line system these locations are 8FBOH to 8FFFH. The 48 line system uses locations 9FBOH to 9FFFH.

The CPU O.S. is the operator's interface to the 8080 CPU, and controls loading and execution of user programs, and to some extent the debugging of user programs. Figure 4-1 illustrates memory utilization during various stages of system software use. While the CPU O.S. is running, it uses an area at the top of memory for data storage and scratch work.

4.1 CPU OPERATING SYSTEM IMPLEMENTATION AND EXECUTION

4.1.1 CPU O.S. IMPLEMENTATION

The Intecolor® 8001 CPU O.S. program is implemented on two E PROM modules, which are pre-installed into each Intecolor 8001 with Option 34. This allows the CPU to be used with great ease, as it is not necessary to wait for lengthy paper-tape loading operations. All that is required to go on-line with CPU O.S. is to turn the Intecolor 8001 on, hit the ESCAPE key, and then the CPU O.S. key, and begin execution.

4.1.2. STARTING SYSTEM MONITOR

To begin operating the CPU O.S., press two keys in sequence, 'ESCAPE', (CPU O.S.) and the Intecolor 8001 will automatically jump to the starting address of the CPU O.S.

4.2 CPU O.S. OPERATION AND COMMANDS

The commands consist of a single letter typed into the Intecolor® 8001 keyboard followed by a number of arguments, possibly none. The arguments are separated, if there are more than one, by spaces or commas. A command is terminated and executed by typing a carriage return or space, depending upon the command.

A (RAM TEST 2,3,4?) [4 is visual]

4.2.1 B COMMAND (BACK TO CRT O.S.)

4.2.2 D COMMAND (DISPLAY DATA)

The format of the D command is:

D low address, high address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Description: Upon execution of this command, memory data from (low address) to (high address) is displayed upon the list device (normally the CR tube). Data are displayed in hexadecimal form. Up to sixteen bytes per line are printed, preceded by the hexadecimal address of the first byte of that line. A carriage return is forced after a byte having a low order digit of F in its memory address is printed.

Example: Enter at the keyboard the command:

```
.D10F, 123(Cr)
```

and the CR Tube will display:

```
010F    AA
0110    BB CC DD EE FF 11 22 33 44 55 66 77 88 99 AB CD
0120    EF 12 34 56
```

where memory locations 010F through 0123 are assumed to contain

```
AA BB CC DD EE FF 11 22 33 44 55 66 77 88 99 AB CD EF
12 34 56
```

the D command should be used only to examine memory contents. To punch the memory contents onto a tape, the W command should be used. These commands produce a tape in the proper formats, while the D command causes a simple sequence of characters to be output.

Error conditions:

1. If low address or high address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command

```
.D30010,AB0013(Cr)
```

is equivalent to the command

```
.D0010,0013(Cr)
```

2. If low address is greater than high address, only the one byte at low address will be displayed.

Example: The command:

.D10,6

is equivalent to the command

.D10,10

3. Non-existent memory is equivalent to a string of bytes all containing FF H.

Example: If memory address 2000 H- 2010 H are invalid, then the command:

.D2000,2010

will cause the teletype to print:

```
2000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2010 FF
```

4. If low address or high address contains an invalid character, or if high address is omitted, the CR Tube will immediately display '?(Cr) (lf). and await the next command.

Example: If the user attempts to enter the number OG as an address, the following will be displayed:

.DOG?

4.2.3 F COMMAND (FILL MEMORY WITH CONSTANT)

The format of the F command is:

F low address, high address, data

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Data is an 8 bit data value.

Description: Execution of this command causes memory locations (low address) through (high address) to be filled with the constant (data).

Example: The command:

.F7,14,AA(Cr)

will set bytes 0007 through 0014 equal to AA H.

```
0007    AA AA AA AA AA AA AA AA AA
0010    AA AA AA AA AA
```

Error Conditions:

1. If low address of high address is greater than 16 bits (or data is greater than 8 bits), only the last 4 (or 2) hex digits will be used.

Example: The command:

.F7AB0007,0014,FFACAA(Cr)

is equivalent to the command:

.F0007,0014,AA(Cr)

2. If low address is greater than high address, data will replace only the byte at low address.

Example: If locations 7, 8, and 9 contain AA H, BB H, and CC H, execution of the command:

.F7,1,33(Cr)

will cause memory to appear as follows:

```
0007   33   BB   CC
```

3. If a non-existent memory address is specified, this command has no effect.

4. If low address, high address, or data contain an invalid character, the CR Tube will immediately display '? (Cr) (lf).' and await the next command.

Example: If the user tries to enter BQ as data, the following will be displayed:

.F0012,14,BQ?

4.2.4 G COMMAND (GO TO)

The format of the G command is:

G address, bkptl, bkpt2

Address, bkptl, and bkpt2 are valid 16 bit hexadecimal memory addresses.

Description: The G command causes program control to be transferred to location address. If either bkptl or bkpt2 is specified, a breakpoint will be set in the program at the corresponding address(es). The specified address must correspond to the first byte of a program instruction. If either breakpoint is encountered during program execution, the CPU O.S. will save and display all program status (CPU registers and condition bits), clear all existing breakpoints, and take control. The user may then examine and/or modify registers or memory, or use any other monitor commands. This feature allows the user to debug portions of a program.

If address is not specified, the program status is restored and the saved value of the program counter is used as the new starting address.

Example: The command:

G24A

will cause program execution to begin at location 24AH, with no breakpoints being set.

The command:

G,12C

will cause a breakpoint to be set at 12CH, and program execution to resume at the address indicated by the saved value of the program counter.

The command:

G

will cause program execution to resume at the address indicated by the saved value of the program counter, with all status restored and no breakpoints set.

Error Conditions:

1. If address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command:

.G3C0010(Cr)

is equivalent to the command

.G0010(Cr)

2. If address is a non-existent memory address, the system will attempt to transfer control and then return to the CRT O.S. with no response. The CPU O.S. must then be manually restarted.

4.2.5. H COMMAND (HEXADECIMAL ARITHMETIC)

The format of the H command is:

.H number, number Sp

Number is a 16 bit hexadecimal number.

Description: The H command is designed to aid the user in performing hexadecimal arithmetic while using the CPU O.S. It causes the sum and difference it arguments to be displayed in two-s complement hexadecimal form. This command is terminated by a space, rather than by a carriage return.

Example:

.H1E,5C 007A FFC2

Error Conditions:

1. If either number is greater than 16 bits, only the last 4 hex digits are used.

Example: The command:

.H00ABC,23Sp

is equivalent to the command:

.H0ABC,23Sp

2. If number contains an invalid character, the CR Tube will immediately display '?(Cr) (lf).' and await the next command.

Example: If the user attempts to enter 01P, the following will be displayed:

.H01P?

4.2.6 I COMMAND (RESET CRT TO STATE S_O)

The format of the I command is:

> I causes the same action as the CPU reset key being typed.

4.2.7 L COMMAND (LOAD HEXADECIMAL FILE)

The format for the L command is:

L bias address

Bias Address is a 16 bit two's complement hexadecimal number.

Description: This command loads tape written in hexadecimal format (using the W command) into memory. The address at which the tape is loaded is determined by adding the address on the tape to the bias address using two's complement arithmetic. The bias may be negative, but in this case must be in two's complement form. If the tape was produced using an E command with a non-zero entry point address (see section 4.2.11), control will be transferred to that location in memory. Otherwise, the CPU O.S. will remain in control and request another command.

Example: If a tape was used which began at location 0100 H, the following command:

.LFFB0(Cr)

will cause the tape to be read and loaded into location 50 H. (1000+FFB0=50).

NOTE: If an error occurs while reading the tape (such as a checksum error), the CPU O.S. will immediately stop reading the tape, display '?(Cr) (Lf).' and await the next command. The operation may be retried by backing up the tape to any point before the last colon and issuing another L command, since each data word specifies the address at which it is to be loaded. The CPU O.S. will read up to the first colon it encounters, and then begin loading data.

Note that this means that, if you wish to change data in locations in memory, it is not necessary to regenerate an entirely new tape with the change; instead you may read in the original tape, then read in a patch tape which reloads only the erroneous locations.

Error Conditions:

1. If the bias address is greater than 16 bits, only the last 4 hex digits are used as the bias address.

Example: The command:

.L00FFB0 (Cr)

is equivalent to the command:

.LFFB0 (Cr)

2. If an invalid character is present in the bias address, the CR Tube will immediately display '*(Cr) (lf).' and await the next command.

Example: If the user attempts to enter G00 as a bias address, the following will be displayed:

.RG?

4.2.8 M COMMAND (MOVE MEMORY)

The format of the M command is:

.M low address, high address, destination address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Destination address is a valid 16 bit memory address.

Description: The M command causes the block of memory from low address through high address to be moved to the locations in memory beginning at destination address.

Example: If memory appears as follows:

LOCATIONS		DATA
0300-0304	contain	01020304
0200-0204	contain	A1A2A3A4

Then the command:

M200,204,300

will cause the following:

LOCATIONS		DATA
0300-0304	contain	A1A2A3A4
0200-0204	contain	A1A2A3A4

Note: The movement is performed byte by byte: the byte at low address is moved to destination address, then low address +1 is moved to destination address+1, etc. Therefore, the MOVE command may be used to fill memory with a byte or sequence of bytes.

Example: If location 0300 H contains FF H, the command

.M300,310,301(Cr)

will cause locations 300 through 310 to contain FF H. The FF at 300 is moved to 301, then the byte at 301 (which is now FF), is moved to 302, and so on.

Error Conditions:

1. If any address is greater than 16 bits, only the last 4 hex digits are used as the address.

Example: The command:

.M00302,303,00405(Cr)

is equivalent to the command:

M302,303,405(Cr)

2. If low address is greater than high address, only one byte will be moved from low address to destination address.

Example: The command:

.M300,2F0,100(Cr)

is equivalent to the command:

.M300,300,100(Cr)

3. If low address through high address specifies a non-existent range of memory, bytes of FF H will be moved to the memory locations specified by destination address.

Example: If locations 2000 H through 2005 are non-existent, the commend:

.M2000,2005,100(Cr)

will cause locations 0100 H through 0105 H to contain FF H.

4. If an invalid character is entered in an address, the CR Tube will display '?(Cr) (lf).' and await the next command.

Example: If the user attempts to enter OBAG as the destination address, the following will be displayed:

M100,10F,OBAG*

4.2.9. R COMMAND (BAUD RATE SELECT)

The format of the R command is

R rate value

The rate value must be between 1 and 7. See chart below.

NUMBER	1	2	3	4	5	6	7
NORMAL BAUD RATE	110	150	300	1200	2400	4800	9600
HIGH SPEED BAUD RATE	880	1200	2400	9600	19,200	38,400	76,800

4.2.10 S COMMAND (SUBSTITUTE MEMORY)

The S command is used to display and/or modify the contents of individual memory locations. It is used as follows:

1. Type an S, followed by the hexadecimal address of the first memory location you wish to display. Type space.

2. The data from the selected address is displayed, followed by a dash (-).

3. To modify memory, type in the new data followed by a space or a carriage return. If you do not wish to modify the contents of that location, do not type any data in, but only type a space or carriage return.

4. If a space was typed in step 3, the next memory location will be displayed as in step 2. If a carriage return was typed, operation will be returned to the CPU O.S.

Example: The contents of the first four bytes of memory is 00 A1 CE FF. You wish to change it to 00 A3 CE 11.

.S0000Sp00Sp A1 - A3Sp CE - Sp FF - 11Cr

User entries are unshaded. Display back is shaded.

Error Conditions:

1. If address is greater than 16 bits, or the data to be substituted is greater than 8 bits, only the last 4 or 2 hex digits respectively are used.

Example: The following sequence is equilvalent to the previous example:

.S0AB0000Sp 00 - Sp A1 - BA3Sp CE - Sp FF - 011Cr

2. If an invalid character is encountered, the CR Tube will immediately display '?(Cr) (lf).' and await the next command.

4.2.11 X COMMAND (EXAMINE AND MODIFY REGISTERS)

The format of the X command is:

X reg ident

X (C.R.) PRINTS ALL REGISTERS, ANNOTATED!

Reg ident is a single character specifying a CPU register as follows:

A = A register
B = B register
C = C register
D = D register
E = E register
F = Flag byte, displayed in the form as it is stored by the instruction PUSH PSW
H = H register
L = L register
M = H and L registers combined (16 bits)
P = Program counter (16 bits)
S = Stack pointer (16 bits)

Note: The format of the flag byte F is:

```
       A
S Z 0 C 0 P 1 C
```

Sign bit	State of carry bit
Zero bit	Always 1
Always 0	State of parity bit
Auxiliary carry bit	Always 0

Description: The X command is used to display and/or modify CPU registers. It operates similar to the S command, as follows:

1. Type an X, followed by the register identifier.

2. The data from the selected register is displayed, followed by a dash (-). Four hexadecimal digits are displayed for M, P, and S; two hex digits for the other register identifiers.

3. To modify the register, type in the new data followed by a space or a carriage return. If you do not wish to modify the register, type only the space or carriage return.

4. If a space was typed in step 3, the next register in alphabetical order is displayed. If carriage return was typed, the X command is terminated. If a space is typed after register S has been displayed, the command is terminated, this being the last register identifier in the list.

Example: The A, B, C, and D registers contain AAH, BBH, CCH, and DDH, respectively. You wish to change the B and C registers to 00H and FFh, respectively.

XASp AA- Sp BB- 00Sp CC- FFSp DD-Cr

Note: Values set by the X-command will become the actual contents of the registers after execution of the next GO command.

The values displayed by the X-command are the contents of the registers prior to the execution of the last breakpoint set by the GO command. These displayed values, however, will reflect any changes of register "contents" made by the execution of X-commands since this last breakpoint.

Error Conditions:

1. If the data to be substituted is greater than 16 bits for registers M, P, S, or 8 bits for the other register identifiers, only the last 4 or 2 hex digits respectively are used.

2. If an invalid register identifier or character is encountered, the CR Tube will immediately display '?(Cr) (Lf).' and await the next command.

4.2.12 E COMMAND (END FILE)

The format of the E command is:

E address

Address is a valid 16 bit memory address.

Description: The E command causes an end-of-file mark and sixty null characters to be written at the end of a hexadecimal output file. The end of file mark is hexadecimal record of length 00. (See Appendix D). If address is 0 or absent, the L command which loads the file will return control to the CPU O.S. If address is non-zero, the L command will transfer control to that memory address immediately after loading the file.

4.2.13 W COMMAND (WRITE MEMORY)

No mention of how to specify the drive or the track number.

The format of the W command is:

W low address; high address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Description: The W command is used to output memory locations low address through high address to the system punch device in hexadecimal format. A series of W commands may be issued in order to punch various non-contiguous memory locations onto a continuous strip of tape.

Any series of W commands should be terminated with an E command in order to punch a termination character, so that when the tape is read it will be handled properly.

Example: If memory locations 1 through 3 contain 53F8EC, the command: .W0001,0003(Cr)

produces:

:0300010053F8ECC5

(See Appendix D for an explanation of tape format.)

Error Conditions:

1. If low address or high address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command:

WAB0010,100(Cr)

is equivalent to the command:

W0010,100(Cr)

2. If low address is greater than high address, only the one byte at low address will be written:

Example: The command:

.W10,0(Cr)

is equivalent to the command:

.W10,10(Cr)

3. Non-existent memory is equivalent to a string of bytes all containing FF H.

4. An invalid character in either address will cause the CR Tube to display '?(Cr) (lf).' and await the next command.

Example: If the user attempts to enter 3Z as low address, the following will be displayed:

.W3Z?

4.2.14 N COMMAND (NULL PUNCH)

The N command consists only of the letter N followed by a carriage return and causes 60 null characters to be written on the punch device.

APPENDIX A

-- INSTRUCTION SUMMARY --

This appendix provides a summary of 8080 assembly language instructions. Abbreviations used are as follows:

A	The accumulator (register A)
A_n	Bit n of the accumulator contents, where n may have any value from 0 to 7 and 0 is the least significant (rightmost) bit.
ADDR	Any memory address
Aux. carry	The auxiliary carry bit
Carry	The carry bit
CODE	An operation code
DATA	8 bits (one byte) of data
DATA16	16 bits (2 bytes) of data
DST	Destination register or memory byte
EXP	A constant or mathematical expression
INTE	The 8080 interrupt enable flip-flop
LABEL	Any instruction label
M	A memory byte
Parity	The parity bit
PC	Program Counter
PCH	The most significant 8 bits of the program counter
PCL	The least significant 8 bits of the program counter
REGM	Any register or memory byte
RP	A register pair. Legal register pair symbols are: B for registers B and C D for registers D and E H for registers H and L SP for the 16 bit stack pointer PSW for condition bits and register A

RP1	The first register of register pair RP
RP2	The second register of register pair RP
sign	The sign bit
SP	The 16-bit stack pointer register
SRC	Source register or memory byte
zero	The zero bit
XY	The value obtained by concatenating the values X and Y
[ ]	An optional field enclosed by brackets
( )	Contents of register or memory byte enclosed by parentheses
$\longleftarrow$	Replace value on lefthand side of arrow with value on right-hand side of arrow

CARRY BIT INSTRUCTIONS

Format: [LABEL:] CODE

CODE	DESCRIPTION	
STC	(carry) $\longleftarrow$ 1	Set carry
CMC	(carry) $\longleftarrow$ $\overline{\text{(carry)}}$	Complement carry

Condition bits affected: Carry

SINGLE REGISTER INSTRUCTIONS

Format:

[LABEL:]	INR	REGM
	-or-	
[LABEL:]	DCR	REGM
	-or-	
[LABEL:]	CMA	
	-or-	
[LABEL:]	DAA	

Code	Description	
INR	(REGM) $\leftarrow$ (REGM) + 1	Increment register REGM
DCR	(REGM) $\leftarrow$ (REGM) - 1	Decrement register REGM
CMA	(A) $\leftarrow$ $(\overline{A})$	Complement accumulator
DAA	If $(A_0-A_3) > 9$ or (aux. carry = 1, (A) $\leftarrow$ (A)+6 Then if $(A_4-A_7) > 9$ or (carry)= 1 (A) = (A) + 6 $*2^4$	Convert accumulator contents to form two decimal digits

Condition bits affected:	INR,DCR	:	Zero, sign, parity
	CMA	:	None
	DAA	:	Zero, sign, parity, carry, aux. carry

NOP INSTRUCTION

Format:

[LABEL:] NOP

Code	Description
NOP	-------------- No operation

Condition bits affected: None

DATA TRANSFER INSTRUCTIONS

Format:

[LABEL:]	MOV	DST,SRC
	-or-	
[LABEL:]	CODE	RP

NOTE: SRC and DST not both = M

NOTE: RP = B or D

Code	Description	
MOV	(DST) ⟵ (SRC)	Load register DST from register SRC
STAX	((RP)) ⟵ (A)	Store accumulator at memory location referenced by the specified register pair
LDAX	(A) ⟵ ((RP))	Load accumulator from memory location refereced by the specified register pair

Condition bits affected: None

REGISTER OR MEMORY TO ACCUMULATOR INSTRUCTIONS

Format:

[LABEL:]	CODE	REGM

Code	Description	
ADD	(A) ⟵ (A)+(REGM)	Add REGM to accumulator
ADC	(A) ⟵ (A)+(REGM)+(carry)	Add REGM to accumulator with carry
SUB	(A) ⟵ (A)-(REGM)	Subtract REGM from accumulator
SBB	(A) ⟵ (A)-(REGM)-(carry)	Subtract REGM from accumulator with borrow
ANA	(A) ⟵ (A) AND (REGM)	AND accumulator with REGM
XRA	(A) ⟵ (A) XOR (REGM)	EXCLUSIVE-OR accumulator with REGM

Code	Description	
ORA	(A) ← (A) OR (REGM)	OR accumulator with REGM
CMP	Condition bits set by (A)-(REGM)	Compare REGM with accumulator

Condition bits affected:

ADD, ADC, SUB, SBB: Carry, sign, zero, parity, aux. carry
ANA, XRA, ORA: Sign, zero, parity. Carry is zeroed.
CMP: Carry, sign, zero, parity, aux. carry. Zero set if (A)=(REGM)
Carry reset if (A) < (REGM)
Carry set if (A) ≥ (REGM)

ROTATE ACCUMULATOR INSTRUCTIONS

Format:

[LABEL:] CODE

Code	Description	
RLC	(carry) ← A_7, A_{n+1} ← A_n, A_0 ← A_7	Set carry = A_7, rotate accumulator left
RRC	(carry) ← A_0, A_n ← A_{n+1}, A_7 ← A_0	Set carry = A_0, rotate accumulator right
RAL	A_{n+1} ← A_n, (carry) ← A_7, A_0 ← (carry)	Rotate accumulator left through the carry
RAR	A_n ← A_{n+1}, (carry) ← A_0, A_7 ← (carry)	Rotate accumulator right through carry

Condition bits affected: Carry

REGISTER PAIR INSTRUCTIONS

Format:

[LABEL:] CODE1 RP
-or-
[LABEL:] CODE2

Note: For PUSH and POP, RP=B,D,H or PSW
For DAD, INX, and DCX, RP=B,D,H, or SP

Code1	Description	
PUSH	((SP)-1) ← (RP1), ((SP)-2) ← (RP2), (SP) ← (SP)-2	Save RP on the stack RP=A saves accumulator and condition bits.
POP	(RP1) ← ((SP)+1), (RP2) ← ((SP)), (SP) ← (SP)+2	Restore RP from the stack RP=A restores accumulator and condition bits.
DAD	(HL) ← (HL) + (RP)	Add RP to the 16-bit number in H and L.
INX	(RP) ← (RP)+1	Increment RP by 1
DCX	(RP) ← (RP)-1	Decrement RP by 1

Code2	Description	
XCHG	(H) ↔ (D), (L) ↔ (E)	Exchange the 16 bit number in H and L with that in D and E.
XTHL	(L) ↔ ((SP)), (H) ↔ ((SP)+1)	Exchange the last values saved in the stack with H and L.
SPHL	(SP) ← (H):(L)	Load stack pointer from H and L.

Condition bits affected:

PUSH, INX, DCX, XCHG, XTHL, SPHL: None
POP : If RP=PSW, all condition bits are restored from the stack, otherwise none are affected.
DAD : Carry

IMMEDIATE INSTRUCTIONS

Format:

[LABEL:]	LX1	RP, DATA16
	-or-	
[LABEL:]	MV1	REGM, DATA
	-or-	
[LABEL:]	CODE	REGM

Note: RP=B,D,H, or SP

CODE	DESCRIPTION	
LXI	(RP) ← DATA 16	Move 16 bit immediate Data into RP
MVI	(REGM) ← DATA	Move immediate DATA into REGM
ADI	(A) ← (A) + DATA	Add immediate data to accumulator
ACI	(A) ← (A) + DATA + (carry)	Add immediate data to accumulator with carry
SUI	(A) ← (A) - DATA	Subtract immediate data from accumulator
SBI	(A) ← (A) - DATA - (carry)	Subtract immediate data from accumulator with borrow
ANI	(A) ← (A) AND DATA	AND accumulator with immediate data
XRI	(A) ← (A) XOR DATA	EXCLUSIVE-OR accumulator with immediate data
ORI	(A) ← (A) OR DATA	OR accumulator with immediate data
CPI	Condition bits set by (A)-DATA	Compare immediate data with accumulator

Condition bits affected:

LXI, MVI: None
ADI, ACI, SUI, SBI: Carry, sign, zero, parity, aux. carry
ANI, XRI, ORI: Zero, sign, parity. Carry is zeroed.
CPI: Carry, sign, zero, parity, aux. carry. Zero is set if (A) = DATA
Carry reset if (A) $<$ DATA
Carry set if (A) $\geqslant$ DATA

DIRECT ADDRESSING INSTRUCTIONS

Format:

[LABEL:] CODE ADDR

CODE	DESCRIPTION	
STA	(ADDR) ← (A)	Store accumulator at location ADDR
LDA	(A) ← (ADDR)	Load accumulator from location ADDR
SHLD	(ADDR) ← (L), (ADDR+1) ← (H)	Store L and H at ADDR and ADDR+1
LHLD	(L) ← (ADDR), (H) ← (ADDR+1)	Load L and H from ADDR and ADDR+1

Condition bits affected: None

JUMP INSTRUCTIONS

Format:

[LABEL:] PCHL

-or-

[LABEL:] CODE ADDR

CODE	DESCRIPTION	
PCHL	(PC) ← (HL)	Jump to location specified by register H and L
JMP	(PC) ← ADDR	Jump to location ADDR
JC	If (carry) = 1, (PC) ← ADDR If (carry) = 0, (PC) ← (PC)+3	Jump to ADDR if carry set
JNC	If (carry) = 0, (PC) ← ADDR If (carry) = 1, (PC) ← (PC)+3	Jump to ADDR if carry reset
JZ	If (zero) = 1, (PC) ← ADDR If (zero) = 0, (PC) ← (PC)+3	Jump to ADDR of zero set
JNZ	If (zero) = 0, (PC) ← ADDR If (zero) = 1, (PC) ← (PC)+3	Jump to ADDR if zero reset
JP	If (sign) = 0, (PC) ← ADDR If (sign) = 1, (PC) ← (PC)+3	Jump to ADDR if plus
JM	If (sign) = 1, (PC) ← ADDR If (sign) = 0, (PC) ← (PC)+3	Jump to ADDR if minus
JPE	If (parity)= 1, (PC) ← ADDR If (parity)= 0, (PC) ← (PC)+3	Jump to ADDR if parity even
JPO	If (parity)= 0, (PC) ← ADDR If (parity)= 1, (PC) ← (PC)+3	Jump to ADDR is parity odd

Condition bits affected: None

CALL INSTRUCTIONS

Format:

[LABEL:]   CODE   ADDR

CODE	DESCRIPTION
CALL	((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR call subroutine and push return address onto stack
CC	If (carry) = 1, ((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (carry) = 0, (PC) ← (PC)+3 Call subroutine if carry set
CNC	If (carry) = 0, ((SP)-1 ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (carry) = 1, (PC) ← (PC)+3 Call subroutine if carry reset
CZ	If (zero) = 1, ((SP)-1)- (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (zero) = 0, (PC) ← (PC)+3 Call subroutine if zero set
CNZ	If (zero) = 0, ((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (zero) = 1, (PC) ← (PC)+3 Call subroutine if zero reset
CP	If (sign) = 0, ((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (sign) = 1, (PC) ← (PC)+3 Call subroutine if sign plus
CM	If (sign) = 1, ((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (sign) = 0, (PC) ← (PC)+3 Call subroutine if sign minus
CPE	If (parity)= 1, ((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (parity)= 0, (PC) ← (PC)+3 Call subroutine if parity even
CPO	If (parity)= 0, ((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2, (PC) ← ADDR If (parity)= 1, (PC) ← (PC)+3 Call subroutine if parity odd

Condition bits affected: None

RETURN INSTRUCTIONS

Format:

[LABEL:] CODE

CODE	DESCRIPTION
RET	(PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 Return from subroutine
RC	If (carry) = 1, (PCH) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (carry) = 0, (PC) ← (PC)+3 Return if carry set
RNC	If (carry) = 0, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (carry) = 1, (PC) ← (PC)+3 Return if carry reset
RZ	If (zero) = 1, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (zero) = 0, (PC) ← (PC)+3 Return if zero set
RNZ	If (zero) =0, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (zero) =1, (PC) ← (PC)+3 Return if zero set
RM	If (sign) = 1, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (sign) = 0, (PC) ← (PC) +3 Return if minus
RP	If (sign) = 0, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (sign) = 1, (PC) ← (PC) +3 Return if plus
RPE	If (parity)=1, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (parity)=0, (PC) ← (PC) __ (PC)+3 Return if parity even
RPO	If (parity)=0, (PCL) ← ((SP)), (PCH) ← ((SP)+1), (SP) ← (SP)+2 If (parity)=1, (PC) ← (PC)+3 Return if parity odd

Condition bits affected: None

RST INSTRUCTION

Format:

[LABEL:] RST EXP

Note: 0 EXP 7

CODE	DESCRIPTION
RST	((SP)-1) ← (PCH), ((SP)-2) ← (PCL), (SP) ← (SP)+2 (PC) ← 0000000000EXP000B Call subroutine at address specified by EXP

Condition bits affected: None

INTERRUPT FLIP FLOP INSTRUCTIONS

Format:

[LABEL:] CODE

CODE	DESCRIPTION	
EI	(INTE) ← 1	Enable the interrupt system
DI	(INTE) ← 0	Disable the interrupt system

Condition bits affected: None

INPUT/OUTPUT INSTRUCTIONS

Format:

[LABEL:] CODE EXP

CODE	DESCRIPTION	
IN	(A) ← input device	Read a byte from device EXP into the accumulator
OUT	output device ← (A)	Send the accumulator contents to device EXP

Condition bits affected: None

HLT INSTRUCTION

Format:

[LABEL:] HLT

CODE	DESCRIPTION	
HLT	----------------	Instruction execution halts until an interrupt occurs.

Condition bits affected: None

PSEUDO - INSTRUCTIONS

ORG PSEUDO - INSTRUCTION

Format:

ORG EXP

Code	Description	
ORG	LOCATION COUNTER ← EXP	Set Assembler location counter to EXP

EQU PSEUDO - INSTRUCTION

Format:

NAME EQU EXP

Code	Description	
EQU	NAME ← EXP	Assign the value EXP to the symbol NAME

END PSEUDO - INSTRUCTION

Format:

END

Code	Description
END	End the assembly.

APPENDIX B

--INSTRUCTION EXECUTION TIMES AND BIT PATTERNS--

This appendix summarizes the bit patterns and number of time states associated with every 8080 CPU instruction.

When using this summary, note the following symbology:

1) DDD represents a destination register. SSS represents a source register. Both DDD and SSS are interpreted as follows:

DDD or SSS	Interpretation
000	Register B
001	Register C
010	Register D
011	Register E
100	Register H
101	Register L
110	A memory register
111	The accumulator

2) Instruction execution time equals number of time periods multiplied by the duration of a time period.

   A time period may vary from 480 nanosecs to 2 microsec.

   When two numbers of time periods are shown (eg. 5/11), it means that the smaller number of time periods will be required if a condition is not met, and the larger number of time periods will be required if the condition is met.

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Time Periods
CALL	1	1	0	0	1	1	0	1	17
CC	1	1	0	1	1	1	0	0	11/17
CNC	1	1	0	1	0	1	0	0	11/17
CZ	1	1	0	0	1	1	0	0	11/17
CNZ	1	1	0	0	0	1	0	0	11/17
CP	1	1	1	1	0	1	0	0	11/17
CM	1	1	1	1	1	1	0	0	11/17
CPE	1	1	1	0	1	1	0	0	11/17
CPO	1	1	1	0	0	1	0	0	11/17
RET	1	1	0	0	1	0	0	1	10
RC	1	1	0	1	1	0	0	0	5/11
RNC	1	1	0	1	0	0	0	0	5/11
RZ	1	1	0	0	1	0	0	0	5/11
RNZ	1	1	0	0	0	0	0	0	5/11
RP	1	1	1	1	0	0	0	0	5/11
RM	1	1	1	1	1	0	0	0	5/11
RPE	1	1	1	0	1	0	0	0	5/11
RPO	1	1	1	0	0	0	0	0	5/11
RST	1	1	A	A	A	1	1	1	11
IN	1	1	0	1	1	0	1	1	10
OUT	1	1	0	1	0	0	1	1	10
LXI B	0	0	0	0	0	0	0	1	10
LXI D	0	0	0	1	0	0	0	1	10
LXI H	0	0	1	0	0	0	0	1	10
LXI SP	0	0	1	1	0	0	0	1	10
PUSH B	1	1	0	0	0	1	0	1	11
PUSH D	1	1	0	1	0	1	0	1	11
PUSH H	1	1	1	0	0	1	0	1	11
PUSH A	1	1	1	1	0	1	0	1	11
POP B	1	1	0	0	0	0	0	1	10
POP D	1	1	0	1	0	0	0	1	10
POP H	1	1	1	0	0	0	0	1	10
POP A	1	1	1	1	0	0	0	1	10
STA	0	0	1	1	0	0	1	0	13
LDA	0	0	1	1	1	0	1	0	13
XCHG	1	1	1	0	1	0	1	1	4
XTHL	1	1	1	0	0	0	1	1	18
SPHL	1	1	1	1	1	0	0	1	5
PCHL	1	1	1	0	1	0	0	1	5
DAD B	0	0	0	0	1	0	0	1	10
DAD D	0	0	0	1	1	0	0	1	10
DAD H	0	0	1	0	1	0	0	1	10
DAD SP	0	0	1	1	1	0	0	1	10
STAX B	0	0	0	0	0	0	1	0	7
STAX D	0	0	0	1	0	0	1	0	7
LDAX B	0	0	0	0	1	0	1	0	7
LDAS D	0	0	0	1	1	0	1	0	7
INX B	0	0	0	0	0	0	1	1	5
INX D	0	0	0	1	0	0	1	1	5
INX H	0	0	1	0	0	0	1	1	5
INX SP	0	0	1	1	0	0	1	1	5

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Time Periods
MOV r_1,r_2,	0	1	D	D	D	S	S	S	5
MOV M,r	0	1	1	1	0	S	S	S	7
MOV r,M	0	1	D	D	D	1	1	0	7
HLT	0	1	1	1	0	1	1	0	7
MVI r	0	0	D	D	D	1	1	0	7
MVI M	0	0	1	1	0	1	1	0	10
INR	0	0	D	D	D	1	0	0	5
DCR	0	0	D	D	D	1	0	1	5
INR A	0	0	1	1	1	1	0	0	5
DCR A	0	0	1	1	1	1	0	1	5
INR M	0	0	1	1	0	1	0	0	10
DCR M	0	0	1	1	0	1	0	1	10
ADD r	1	0	0	0	0	S	S	S	4
ADC r	1	0	0	0	1	S	S	S	4
SUB r	1	0	0	1	0	S	S	S	4
SBB r	1	0	0	1	1	S	S	S	4
NDA r	1	0	1	0	0	S	S	S	4
XRA r	1	0	1	0	1	S	S	S	4
ORA r	1	0	1	1	0	S	S	S	4
CMP r	1	0	1	1	1	S	S	S	4
ADD M	1	0	0	0	0	1	1	0	7
ADC M	1	0	0	0	1	1	1	0	7
SUB M	1	0	0	1	0	1	1	0	7
SBB M	1	0	0	1	1	1	1	0	7
NDA M	1	0	1	0	0	1	1	0	7
XRA M	1	0	1	0	1	1	1	0	7
ORA M	1	0	1	1	0	1	1	0	7
CMP M	1	0	1	1	1	1	1	0	7
ADI	1	1	0	0	0	1	1	0	7
ACI	1	1	0	0	1	1	1	0	7
SUI	1	1	0	1	0	1	1	0	7
SBI	1	1	0	1	1	1	1	0	7
NDI	1	1	1	0	0	1	1	0	7
XRI	1	1	1	0	1	1	1	0	7
ORI	1	1	1	1	0	1	1	0	7
CPI	1	1	1	1	1	1	1	0	7
RLC	0	0	0	0	0	1	1	1	4
RRC	0	0	0	0	1	1	1	1	4
RAL	0	0	0	1	0	1	1	1	4
RAR	0	0	0	1	1	1	1	1	4
JMP	1	1	0	0	0	0	1	1	10
JC	1	1	0	1	1	0	1	0	10
JNC	1	1	0	1	0	0	1	0	10
JZ	1	1	0	0	1	0	1	0	10
JNZ	1	1	0	0	0	0	1	0	10
JP	1	1	1	1	0	0	1	0	10
JM	1	1	1	1	1	0	1	0	10
JPE	1	1	1	0	1	0	1	0	10
JPO	1	1	1	0	0	0	1	0	10

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Time Periods
DCX B	0	0	0	0	1	0	1	1	5
DXC D	0	0	0	1	1	0	1	1	5
DCX H	0	0	1	0	1	0	1	1	5
DCX SP	0	0	1	1	1	0	1	1	5
CMA	0	0	1	0	1	1	1	1	4
STC	0	0	1	1	0	1	1	1	4
CMC	0	0	1	1	1	1	1	1	4
DAA	0	0	1	0	0	1	1	1	4
SHLD	0	0	1	0	0	0	1	0	17
LHLD	0	0	1	0	1	0	1	0	17
EI	1	1	1	1	1	0	1	1	4
DI	1	1	1	1	0	0	1	1	4
NOP	0	0	0	0	0	0	0	0	4

APPENDIX C

HEXADECIMAL PROGRAM TAPE FORMAT

The hexadecimal tape format used by the Intecolor® 8001 system is a modified memory image, blocked into discrete records. Each record contains record length, record type, memory address, and checksum information in addition to data. A frame by frame description is as follows:

Frame 0	Record Mark, Signals the start of a record. The ASCII character colon (":" HEX 3A) is used as the record mark.
Frames 1,2 (0-9,A-F)	Record Length. Two ASCII characters representing a hexadecimal number in the range 0 to 'FF'H (0 to 255). This is the count of actual data bytes in the record type or checksum. A record length of 0 indicates end of file.
Frames 3 to 6	Load Address. Four ASCII characters that represent the initial memory location where the data following will be loaded. The first data byte is stored in the location pointed to by the load address, succeeding data bytes are loaded into ascending addresses.
Frames 7,8	Record Type. Two ASCII characters. Currently all records are type 0, this field is reserved for future expansion.
Frames 9 to 9+2* (Record Length) - 1	Data. Each 8 bit memory word is represented by two frames containing the ASCII characters (0 to 9, A to F) to represent a hexadecimal value 0 to 'FF'H (0 to 255).

Frames 9+2* (Record Length) to 9+2* (Record Length) +1	Checksum. The checksum is the negative of the sum of all 8 bit bytes in the record since the record mark (":") evaluated modulus 256. That is, if you add together all the 8 bit bytes, ignoring all carries out of an 8-bit sum, then add the checksum, the result is zero.

Example: If memory locations 1 through 3 contain 53F8EC, the format of the hex file produced when these locations are punched is:

:0300010053F8ECC5

Note: This format is also known as the Intel format.

16 COMBINATIONS BETWEEN BLUE & RED (5 EACH)
MULTIPLY BY 4 DIFFERENT GREEN STATES (12 EACH)

64 COMPUCOLOR COLORS? BLUE LEFT, RED RIGHT, GREEN TOP

B (BE)	G (BF)	R (BF)		B (BE)	G (BE)	R (BE)	
00	00	00	BLACK	10	00	00	BLUE
00	00	01	DIM RED	10	00	01	VIOLET
00	00	10	RED	10	00	10	MAGENTA
00	00	11	BRIGHT RED	10	00	11	PURPLE
00	01	00	DIM GREEN	10	01	00	AQUA
00	01	01	DIM YELLOW	10	01	01	BLUISH WHITE
00	01	10	ORANGE	10	01	10	MAGENTAISH WHITE
00	01	11	ORANGISH-RED	10	01	11	PURPLISH WHITE
00	10	00	GREEN	10	10	00	CYAN
00	10	01	YELLOW GREEN	10	10	01	WHITE-ISH-CYAN
00	10	10	YELLOW	10	10	10	WHITE
00	10	11	ORANGE	10	10	11	PINK (reddish white)
00	11	00	BRIGHT GREEN	10	11	00	GREENISH CYAN
00	11	01	BRIGHT YELLOWISH GREEN	10	11	01	DEEP GREENISH WHITE
00	11	10	BRIGHT GREENISH YELLOW	10	11	10	GREENISH WHITE
00	11	11	BRIGHT YELLOW	10	11	11	BRIGHT YELLOWISH WHITE
01	00	00	DIM BLUE	11	00	00	BRIGHT BLUE
01	00	01	DIM MAGENTA	11	00	01	DEEP BLUE
01	00	10	PURPLE	11	00	10	VIOLET
01	00	11	DEEP RED	11	00	11	BRIGHT MAGENTA
01	01	00	DIM CYAN	11	01	00	AQUA-BLUE
01	01	01	DIM WHITE	11	01	01	DEEP BLUISH WHITE
01	01	10	DIM PINK	11	01	10	DEEP VIOLETISH WHITE
01	01	11	DEEP PINK	11	01	11	DEEP MAGENTAISH WHITE
01	10	00	BLUE-GREEN	11	10	00	DEEP CYAN
01	10	01	DEEP GREENISH WHITE	11	10	01	AQUAISH WHITE
01	10	10	DEEP YELLOWISH WHITE	11	10	10	BLUISH WHITE
01	10	11	DEEP ORANGISH WHITE	11	10	11	BRIGHT MAGENTAISH WHITE
01	11	00	BLUISH GREEN	11	11	00	BRIGHT CYAN
01	11	01	GREENISH WHITE	11	11	01	WHITISH CYAN
01	11	10	YELLOW-GREENISH WHITE	11	11	10	CYANISH WHITE
01	11	11	YELLOWISH WHITE	11	11	11	BRIGHT WHITE

+1	÷4
0 1	.25
1 2	.5
2 3	.75
3 4	1
4 5	1.25
5 6	1.5
6 7	1.75
7 8	2
8 9	2.25
9 A	2.5
A B	2.75
B C	3.
C D	3.25
D E	3.5
E F	3.75
F 16	4

TAKE NUMBER: 1000
÷2

1000
1000

1000
10111
1100

TAKE #1
NOT
MUL. BY 2

×2

000⊗111 = 111
001⊗101 = 000
010⊗011 = 001
011⊗001 = 010
100⊗110 = 010
101⊗100 = 001
110⊗010 =
111⊗000 =

IF (C+1)/2 = EVEN = ∅, ODD = 1

IF C+2/4 = EVEN = ∅, ODD = 1

23 67 10 11
34 78 1112

TAKE C, ADD 1

0	0	=0	=0
1		=1	=0
2		=3	+1
3		=2	-1
4		=6	+2
5		=7	-2
6		=5	-1
7		=4	-3
8		=12	+4
9		=13	+4
10		=15	+5
11		=14	+3
12		=10	-2
13		=13	0
14		=9	-5
15		=8	-7

1001
0100
00100
1010

1000
0111
01000

START AT 8, GO EVERY 16 ÷32
START AT 4, GO EVERY 8 ÷16
START AT 2, GO EVERY 4 ÷8
START AT 1, GO EVERY 2 - cycle ÷4

1 0 0 0	8			
1 0 0 1	14			
1 0 1 1	13			
1 0 1 0	10			
1 1 1 0	14			
1 1 1 1	10	15		
1 1 0 1	9	13		
1 1 0 0	12			
0 1 0 0	4			
0 1 0 1	6	5		
0 1 1 1	5	7		
0 1 1 0	6			
0 0 1 0	2			
0 0 1 1	2	3		
0 0 0 1	1	1		
0 0 0 0	0			

TAKE C: 0 1 2 3 4 5 6 7
SUB 1: 7 0 1 2 3 4 5 6
DIV. 4: 1.75 0 .25 .5 .75 1 1.25 1.5 .5 or greater = ∅

SUB 2: 6 7 0 1 2 3 4 5
DIV. 8: .5 or greater = ∅

SUB 4: 2 3 4 5 6 7 0 1
DIV. 16

TAKE NO: 10011010
NOT AND DIVIDE BY 2 01100101