CROMEMCO

3K CONTROL BASIC
INSTRUCTION MANUAL

(Models MCB-216 & CB-308)

CROMEMCO, INC.
280 Bernardo Avenue
Mountain View, CA 94043

Part No. 023-0023 April 1979

Copyright © 1977, 1979
By CROMEMCO, INC.
All Rights Reserved

This manual was produced in its entirety with the Cromemco Word Processing System and was printed on a Cromemco 3355 Printer with proportional spacing.

Table of Contents

CHAPTER 1

Introduction

Control Basic is a computer language developed by Cromemco specifically for process control, automated testing, data acquisition, and similar applications. Control Basic is also an excellent language for first-time users of microprocessor systems since it is both easy to use and very powerful. Some of the advanced features of Control Basic include:

1. Programs can be executed directly from ROM without first being transferred to RAM.

2. Programs can read and write to RAM locations directly by use of the GET and PUT commands.

3. Programs can input from and output to I/O ports directly by use of the IN and OUT commands.

4. Machine language subroutines can be called directly by use of the CALL command. Up to 60 arguments can be passed from Control Basic to the subroutine in a single call.

5. Extensive output formatting is available including soft and hard terminal width, tabs, numerical field width, over-printing, and decimal or hex number output.

6. Multiple commands can be included in each statement.

7. Numbers can be represented in decimal, hexadecimal, or ASCII format both within the program and as input to the program.

8. The INPUT command automatically prints the variable name as the prompt if the program does not provide one. Operators making a syntax error in response to an INPUT request will be prompted again.

9. All commands can be abbreviated to produce very compact source code.

10. String input, output, and string manipulations can be carried out.

11. Multiple program files can be created. These programs can be run independently or may call one another as subroutines.

12. Programs can be stored in 2708 EPROM in an 8K Bytesaver* II from Cromemco using the EPROM command. Similarly programs can be stored in 2716 EPROM in a 32K Bytesaver* from Cromemco using the SAVE command.

Even with all these features, the Control Basic Interpreter occupies just 3K bytes of memory. Two versions of the interpreter are available from Cromemco. The first version is supplied in three 1K-byte 2308 ROM memories. This version (model CB-308) is designed to be resident in memory locations E400H through EFFFH. The three ROMs can be used in the Cromemco Bytesaver II (model 8KBS) or 16K ROM (model 16KPR) memory cards.

The second version of Control Basic comes in two 2K-byte 2316 ROM memories. This two-ROM set (model MCB-216) also contains a 1K-byte Z80 Monitor program. The two-ROM set is designed to be resident in memory space from locations 0000H through 0FFFH. These ROMs can be used in the Cromemco Single Card Computer (model SCC) or in the Cromemco 32K Bytesaver (model 32KBS) memory card.

An unusual and very important feature of Control Basic in the two-ROM set (MCB-216) is the AUTORUN command. If this command makes up the first line of the Basic program, then execution of the program will begin automatically upon power-on or reset without user intervention. This eliminates the need for an operator to type "RUN" in order to begin program execution.

*Trademark of Cromemco, Inc.

CHAPTER 2

Getting Started with Control Basic

2.1 Installing Control Basic in Your System

The Control Basic Interpreter model CB-308 is located at hex E400-EFFF in ROM. The cold start for CB-308 is at E400H, while the warm start to prevent loss of stored programs is at E42FH. (More information on the Control Basic warm starts will be found in section 5.9.1.) The three ROMs supplied can be used in ROM positions 1, 2, and 3 of a Cromemco 8K Bytesaver board addressed at E000H in memory space. (ROM position 0 of the Bytesaver is normally reserved for the Cromemco ZM-108 Monitor program.)

Control Basic model MCB-216 is located at hex 0423-0FFF in ROM with its cold start at 423H and its warm start at 452H (see section 5.9.1). This version of Control Basic also comes with the Cromemco SCC Monitor program located at 0-422H. Model MCB-216 is supplied in two ROMs that are normally plugged into ROM sockets 0 and 1 of the Cromemco Single Card Computer. They may also be used in ROM sockets 0 and 1 of a 32K Bytesaver addressed at 0 in memory.

The reader is referred to Chapter 8 for more information on memory allocation for both models of Control Basic.

2.2 Entering Programs from the Console Device

Once you have loaded your model of Control Basic into the appropriate Bytesaver or Single Card Computer (see above and Chapter 1), the prompt will be given:

```
OK
>
```

and the program will be waiting for input. To display the stored program, type "LIST" (section 5.6.2). To delete that program, type "NEW" (section 5.6.1). To enter a new statement, type

the statement number and the text of the statement. To delete a statement, type the statement number and a carriage return (CR) only.

If you notice an error in typing before you hit the CR, you can delete the last character by using the RUBout or DELete key or delete the entire line with the ESCape key. Control Basic will echo a back-slash (\) for each deleted character. A deleted line is marked by a CR-LF and an up-arrow (^). The ALT-MODE key (ASCII 7DH) may be used in place of the ESC key to delete lines (useful with a teletype).

To correct a statement, you can retype the statement number and the correct commands. Control Basic will replace the old statement with the new one.

You can verify the corrections or insertions to your program by using the LIST command. LIST allows you to view either single statements or complete blocks of text.

2.3 Entering or Saving Programs with Other Devices

Control Basic may be used with devices other than the console terminal. A paper tape reader and punch for Control Basic are assumed (in the standard I/O drivers) to be part of and connected to the same port address as the console device, as is the case with a teletype. Control-O is used as an echo control for the input device. Thus, a program on paper tape may be read in without having it typed out on the console device by either the operator pressing CTRL-O, or by the tape containing the CTRL-O character itself in the leader. Users who change the I/O drivers in RAM (model CB-308 only) may desire to assign the punch and reader to a port address other than that used for the console device. In such a case the function of CTRL-O could be preserved to allow switching from one device to another. Control Basic model CB-308 could also be interfaced to a printer for those users not having teletypes by changing the I/O drivers in RAM (see section 8.1.2 for complete listing of present drivers).

Pressing Control-O complements the value stored in "OCSW" in RAM (see listings in section 8.1.1 or

8.2.1). Thus, this may also be changed within a Control Basic program rather than from the console to allow the inputting of a password or other invisible characters. An example will illustrate this:

```
10 PRINT "INPUT PASSWORD: ",; REM Prompt operator
20 PUT(%3E5)=XOR(GET(%3E5),%FF); REM Complement "OCSW"
30 INPUT $(0); REM Input the actual password
40 PUT(%3E5)=XOR(GET(%3E5),%FF); REM Allow echo again
```

This example is for model CB-308 Control Basic but could easily be changed for MCB-216. Refer to Chapter 5 for explanations of the commands used above and to Chapter 3 for an explanation of strings $(J), and the GET and XOR functions.

To produce a paper tape of a Control Basic program using the punch, one method is to type "LIST" (no CR), turn on the punch, type as many nulls as are desired for a leader, and then a carriage return. Nulls are produced on a teletype by Control-Shift-P and on a CRT by Control-@. When the listing is finished, type more nulls for a trailer and then turn off the punch.

A paper tape of the form just described may be read back by the method described earlier: First type "NEW" to clear any current program; then press Control-O and turn on the tape reader. When the paper tape is finished loading, turn off the reader and press Control-O to once again allow echoed input and output.

CHAPTER 3

Elements of the Control Basic Language

3.1 Numbers and Constants

In Control Basic all numbers are integers and must be less than 32767. Numbers are stored internally as 16-bit two's complement (low byte-high byte). For programming and input/output, numbers are expressed in decimal, hexadecimal, or the ASCII character code. Hexadecimal numbers are denoted by the character "%" followed by up to 4 hexadecimal digits. For example, %21 and 33 are both stored internally as ØØ1ØØØØ1 ØØØØØØØØ; %FFF4 and -12 are both stored internally as 11110100 11111111. An ASCII code (7-bit) is denoted by a colon (:) followed by an ASCII character. For example, :A has the value 65 (or 41H) and is stored internally either as one byte, Ø1ØØØØØ1, or as two bytes, Ø1ØØØØØ1 ØØØØØØØØ, depending on how it is used.

3.2 Variables

There are 52 variables denoted by letters A through Z and AØ through ZØ. There is also an array @(I). The dimension of this array (i.e., the range of the index I) is set automatically to make use of all the memory space that is allocated but left unused by the text of the current program (i.e., Ø through SIZE/2; see SIZE function below. Also see section 8.1.1 or 8.2.1 on memory allocation.). Each variable and each element of the array @(I) uses two bytes of memory stored low byte-high byte.

The same memory space that is used to store the array @(I) can also be accessed, one byte at a time, through &(J) (or as strings through $(J) where J runs from Ø through SIZE. That is, &(Ø) is the low byte of @(Ø), &(1) is the high byte of @(Ø), &(2) is the low byte of @(1), &(3) is the high byte of @(1), etc. Strings of up to 132 characters each can be stored in the same memory area and are accessed through $(J). The length N of the string $(J) is stored in &(J); the 7-bit ASCII characters of the string are stored in &(J+1) through &(J+N). Thus, $(J+N+1) can be another string.

The multiple naming of the same memory space is intentional. This not only gives the programmer the freedom to allocate available memory between arrays and strings but also is very convenient for packing and unpacking data, and for string manipulation.

3.3 Functions

Control Basic has 10 built-in functions, which are described below. The values returned by these functions will always be printed in decimal unless the user requests the answer in hex. This may be done by means of the format controls of the PRINT command, section 5.4.1. In the following x and y denote variable or constant expressions.

ABS(x)	gives the absolute value of x.
RND(x)	gives a random number between 1 and x (inclusive).
SIZE	gives the number of bytes allocated to, but left unused by the current program in the text area. See section 8.1.1 or 8.2.1 for a discussion of memory allocation.
SGN(x)	returns a 1 if x is positive (or zero) or a -1 if x is negative.
AND(x,y)	gives the 16-bit Boolean AND of x and y.
OR(x,y)	gives the 16-bit Boolean inclusive OR of x and y.
XOR(x,y)	gives the 16-bit Boolean exclusive OR of x and y. Note that XOR(x,%FFFF) gives the one's complement of x, and -x gives the two's complement of x. (The result z of this operation must be in the range $-32767 \leq z \leq 32767$. Hence there is no one's complement of 7FFFH nor two's complement of 8000H.)

GET(x)	gives the 8-bit contents (1 byte) of memory location x.
IN(x)	gives the 1-byte value input from port x, where Ø<=x<=255.
LOC	gives the absolute address of @(Ø), &(Ø), and $(Ø).

3.4 Arithmetic and Compare Operators

Control Basic allows the following operators in variable and constant expressions:

/	divide.
*	multiply, or logical AND.
-	subtract, or two's complement.
+	add, or logical OR.
>	greater than (compare).
<	less than (compare).
=	equal to (compare).
#	not equal to (compare).
>=	greater than or equal to (compare).
<=	less than or equal to (compare).

The operations +, -, *, and / result in a value between -32767 and 32767. Results outside this range will cause an error, and Control Basic will print "HOW?". All compare operators result in a 1 if true and a Ø if false (not true).

In the Logical context, a Ø is False and any non-zero value is True. Thus, the add operator (+) is also used as logical OR, and the multiply operator (*) is also used as logical AND. For logical complement, one may use Ø=. For example:

```
A+B  can mean  A OR B.
A*B  can mean  A AND B.
Ø=A  can mean  NOT A.
```

Note that if one wants the 16-bit-by-bit Boolean AND, one should use the function AND(A,B) instead of the logical AND just described. The same applies for Boolean OR and XOR.

3.5 Expressions

Expressions are formed from numerical elements with arithmetic or compare operators between them. Numerical elements include constants (decimal, hex, or ASCII), variables, two-byte array elements @(I), one-byte array elements &(J), functions, and other expressions in parentheses. A "+" or a "-" sign can also be used as a unary operator. The hierarchy for evaluating expressions is as follows: functions are evaluated first, then parentheses (from the innermost), then multiplication and division, then addition and subtraction, then the compare operations. Cascaded * and / or cascaded + and - are evaluated from left to right. Compare operators cannot be cascaded. Several commented examples of both legal and illegal expressions follow:

1+2*3	has the value 7, not 9.
2>1+3	is false and has the value Ø (> is used as a compare operator and 2 is not greater than 1+3).
2*5/3	has the value 3, while 2/3*5 is Ø due to truncation.
A+-3	is invalid, but A>-3 is allowed.
A<X<B	is invalid; (A<X) * (X<B) is the proper way of testing whether X is between A and B.
(A>B)*X + (A=B)*Y	is equal to X if A>B, is equal to Y if A=B, and is equal to Ø if A<B.
(&(8)>=:A)*(&(8)<=:Z)*2 + (&(8)>=:Ø)*(&(8)<=:9)	has a value of 2 if the character &(8) is a letter, a value of 1 if it is a digit, and a value of Ø otherwise.

CHAPTER 4

Control Basic Syntax

4.1 Control Basic General Syntax

There are a number of special definitions used throughout this manual as well as some general rules of syntax which should be followed when entering and editing Control Basic programs. The definitions of important terms follow:

command -
Applies to Control Basic keywords which may be used in either direct commands or statements.

direct command -
Commands which may be typed directly from the console in response to the Control Basic prompt.

statement -
A numbered line in a Control Basic program which is composed of one or several commands.

multiple-command statement -
A numbered line in a Control Basic program which is composed of several commands (on the same line) separated by semi-colons.

function -
One of ten control structures intrinsic to Control Basic which may be used in expressions. (Note that commands may not be used in expressions.)

current program -
The program which may be entered or edited directly without first LOADing it; the program which prints on the console when "LIST" is typed.

text area -
The area of memory reserved for storage of the current program and the arrays.

There are also a number of rules of syntax to be followed when entering and editing Control Basic programs. These are summarized below:

1. A Control Basic programs consists of one or more numbered statements. The statement number must be an integer between 1 and 32767, inclusive.

2. Multiple-command statements (see above) must use semi-colons (;) to separate the commands. There are three exceptions to this: GOTO, STOP, and RETURN cannot be followed by a semi-colon or other commands; they must be the last command on any given line.

3. When the direct command "RUN" is issued, the statement with the lowest line number is executed first, followed by the one with the next lowest line number, etc. The statements GOTO, RUN, STOP, GOSUB, and RETURN can alter this sequence, however, by causing control to branch to a specific line number (see Chapter 5).

4. Execution of the commands within a multiple-command statement is from left to right. The IF command is a special case in that if the condition tested is false, all commands to the right of that point will be skipped over and execution will continue with the next line.

5. Spaces (blanks) may be used or omitted freely but for the following exceptions: constants, and command and function keywords may not contain embedded blanks. They may be abbreviated, however; see section 4.2.

6. Execution of a running program or listing of any type of output may be aborted by pressing the Control-C key (usually the two keys, CTRL and C, depressed simultaneously) on the console input device.

7. Throughout this manual, portions of example programs which are underlined signify that those expressions are user-typed.

4.2 Abbreviations and Summary of Commands

Control Basic command and function keywords may be abbreviated if desired. Since the Interpreter stores programs as the actual ASCII characters which make up the statements and this requires one byte per character (see Memory Allocation sections), the abbreviated forms of commands greatly reduce their memory storage requirements. This is important if the program is to be stored in PROM for future use (see SAVE and EPROM commands). For example, the following command line:

```
10 FOR I=1 TO 1000 STEP 2 ; PRINT I , ; NEXT I
```

would perform exactly the same function if abbreviated to the form:

```
10 F.I=1TO1000S.2;P.I,;NE.I
```

This second line requires only about half the bytes for storage as the first line. However, since this second line is not very legible, it is recommended that a programmer develop a program using the full-word commands, and only translate it to the abbreviated form just prior to programming it into PROM.

The abbreviations are formed by truncating several of the trailing letters and replacing them with a period (.). For example, "P.", "PR.", "PRI.", and "PRIN." all stand for the word "PRINT". Command keywords cannot be truncated to a shorter length than the minimum abbreviations given below or Control Basic will print "WHAT?". Also note that the word "LET" in a LET statement may be omitted altogether. There are three columns in the table below: the first shows all Control Basic commands, the second shows only those which may be used in numbered statements (these may also be used as direct commands), and the third column shows the ten Control Basic functions. The shortest keyword abbreviations are:

Direct Commands	Statements	Functions
* AUTORUN=AUTORUN	* AUTORUN=AUTORUN	A.=ABS
C.=CALL	C.=CALL	AND=AND
E.=EPROM		G.=GET
F.=FOR	F.=FOR	IN=IN
G.=GOTO	G.=GOTO	L.=LOC
GOS.=GOSUB	GOS.GOSUB	OR=OR
IF=IF	IF=IF	R.=RND
IN.=INPUT	IN.=INPUT	S.=SIZE
=LET (implied)	=LET (implied)	SGN=SGN
L.=LIST		X.=XOR
LO.=LOAD		
LOCK=LOCK		
NEW=NEW		
NE.=NEXT	NE.=NEXT	
NU.=NULL		
O.=OUT	O.=OUT	
P.=PRINT	P.=PRINT	
PUT=PUT	PUT=PUT	
Q.=QUIT		
* R.=RDOS		
R.=RETURN	R.=RETURN	
REM=REM	REM=REM	
RUN=RUN	RUN=RUN	
S.=SAVE		
S.=STEP	S.=STEP	
ST.=STOP	ST.=STOP	
TO=TO	TO=TO	
W.=WIDTH		

* These commands are found in model MCB-216 only.

As defined in section 4.1, a command is part of a statement, which, in turn, is part of a program. However, when a command is typed at the console input device without a line number preceding it, it is a direct command. Direct commands are not stored by Control Basic but are executed immediately.

The commands listed in the second column above may be used both as statements of stored programs and as direct commands. Those not listed in column two but listed in column one may be used as direct commands only. There are three groups of special cases:

1. When used as direct commands, GOSUB, INPUT, and RUN cannot be followed by semi-colons and other commands.

2. AUTORUN, NEXT, RETURN, REM, STEP, STOP, and TO are meaningless as direct commands. However, NEXT, STEP, and TO may be used within a direct FOR command all typed on one line as shown in this example:

```
FOR I=Ø TO 255 STEP 4; PRINT #%,,I,; NEXT I
```

3. When used in statements, GOTO, RETURN, and STOP must be the last command on the line (cannot be followed by semi-colons and other commands).

4.3 Memory Organization of Control Basic

The text of the current program is stored in memory one ASCII character per byte. Line numbers are low byte-high byte 16-bit binary and are stored with the text (section 8.1.1 or 8.2.1). The amount of memory space available for the current program may be changed by means of the LOCK command (section 5.7.1). The number of bytes available to (by LOCK or by default), but left unused by, the current program may be obtained with the SIZE function (section 3.3). These unused bytes are used for storage of the two-byte, one-byte, and string arrays (section 3.2).

One may not use the LOCK command to set the boundary at any page that already has current program text in it, nor at any page below the bottom of the text area. If the LOCKed area is exceeded during the entering of a statement of the current program, Control Basic will print "SORRY" and will accept no more statement input until more space is made available, either by deleting some present lines or by unLOCKing more text area.

Control Basic allocates storage in "pages", or 256-byte units. Model CB-3Ø8 has roughly the following organization:

page Ø - not used
pages 1,2, and 3 - used for I/O routines, variables, and stack storage
pages 4 through 31 - text area for current program and arrays
pages 32 through 227 and 240 through 255 - used for SAVEd program files
pages 228 through 239 - storage of Control Basic program itself

Model MCB-216 has a slightly different organization:

pages Ø through 15 - storage of Control Basic program itself
pages 16 through 31 - used for previously-stored programs in PROM
pages 32 and 33 - used for variables and stack storage
pages 34 and 35 - text area for current program and arrays
pages 36 through 255 - used for SAVEd program files

Much more detailed information on memory allocation including listings of the data areas in RAM will be found in sections 8.1.1 (model CB-308) and 8.2.1 (model MCB-216).

CHAPTER 5

Control Basic Commands and Statements

5.1 Assignment Commands

5.1.1 LET Command

The LET command can be used to evaluate expressions and store the values in variables, the double byte array @(I), and the single byte array &(J). More than one variable or array element may be set by a single LET command. Some examples of the use of LET are:

```
10 LET A=234-5*6, @(A-3)=4*A, &(25)=18
20 LET U0=(A<B)+AND(ABS(X-1),%FF)+:*
30 LET A0=B0=0
40 LET @(0)=-1, &(0)=0
50 A=3, B=2*A
```

Note that statement 30 will not set both A0 and B0 to zero. The second equal sign (=) in this command is a compare operator (see section 3.4); thus only A0 will be altered according to whether B0 is equal or is not equal to 0.

Note also that after statement 40 is executed, the value of @(0) will not be -1. This is because &(2*I) is the same as the low order byte of @(I), and &(2*I+1) is the same as the high order byte of @(I). Thus &(1) will be 255 and @(0) will be -256.

The word "LET" is optional. Thus statement 50 sets A to 3 and B to 6.

5.1.2 PUT Command

The PUT command can be used to evaluate expressions and store the values (which must be between 0 and 255) into absolute memory locations. The form of PUT is

```
PUT(x) = a,b,c,...
```

where a, b, and c are expressions whose values are stored in consecutive memory locations beginning

with the address which is the value of the expression x. Either single or consecutive memory locations may be altered with the PUT command. An example of the use of PUT is

```
10 PUT(%800)=%11,AND(X+3,%FF),(X+3)/%100,%CD,%A4,%EC,%C9
20 PUT(X+3) = :H, :O, :W, :D, :Y, %D
```

where part of a machine language subroutine (LD HL,X+3; CALL ECA4H; RET) is PUT into memory locations %800 through %806, and the string of ASCII code for "HOWDY <CR>" is put into locations X+3 through X+8. A better way of unpacking the low and high bytes of X+3 in statement 10 is to initialize the two byte array @(I) to the address, X+3, and then use the one byte array, &(I) or &(I+1) as appropriate, to specify low or high byte:

```
 5 LET @(0)=X+3
10 PUT(%800)=%11, &(0), &(1), %CD, %A4, %EC, %C9
```

The counterpart of PUT is GET , but GET is a function as opposed to a command. GET(x) can be used in any expression and returns the contents of absolute memory location x.

5.2 Control Commands

5.2.1 IF Command

The IF command consists of the word IF followed by an expression, and then followed by one or more other commands. An example is:

```
10 IF A<B LET X=3; PRINT 'A IS LESS THAN B'
```

The command above tests the value of the expression A<B. If it is true (i.e., if it is not zero), the commands in the rest of the line of this statement will be executed. If the value of the expression is false (i.e., if it is zero), the rest of this statement will be skipped over and execution will continue with the next statement. Note that the word "THEN" is not used.

5.2.2 GOTO Command

The GOTO command consists of the word GOTO followed by an expression. An example is:

```
10 GOTO 120
```

This statement causes the program being executed to jump to statement 120. Note that a GOTO command cannot be followed by a semi-colon and other commands. It must end in a carriage return. The statement

```
10 GOTO A*10+B
```

causes the program to jump to a variable statement number as computed from the value of the expression.

Indiscriminate use of GOTO statements makes a program difficult to follow and should be avoided. On the other hand combining short and logically related commands in a single statement can make a program easier to understand and is highly recommended. Consider the following example of a game; this program uses many GOTOs but does not take advantage of multiple-command statements.

```
 10 PRINT "NUMBERS, NUMBERS, WHO CAN GUESS MY NUMBERS?"
 20 LET X=RND(100)
 30 LET N=0
 40 PRINT "I HAVE A NUMBER BETWEEN 1 AND 100"
 50 INPUT "YOUR GUESS" G
 60 LET N=N+1
 70 IF N>20 GOTO 130
 80 IF G>X GOTO 160
 90 IF G<X GOTO 180
100 PRINT "YOU GOT IT IN", N, " GUESSES"
110 PRINT "LET'S PLAY AGAIN"
120 GOTO 20
130 IF G=X GOTO 100
140 PRINT "NO, IT WAS", X
150 GOTO 110
160 PRINT "TRY A SMALLER NUMBER"
170 GOTO 50
180 PRINT "TRY A BIGGER NUMBER"
190 GOTO 50
```

This is a working program, but it could have been much cleaner and easier to understand with half of the GOTO statements deleted and with a few of the other statements combined:

```
 10 PRINT "NUMBERS, NUMBERS, WHO CAN GUESS MY NUMBERS?"
 20 LET X=RND(100), N=0
 40 PRINT "I HAVE A NUMBER BETWEEN 1 AND 100"
 50 INPUT "YOUR GUESS" G; LET N=N+1
 70 IF (N>20)*(G#X) PRINT "NO, IT WAS", X; GOTO 110
 80 IF G>X PRINT "TRY A SMALLER NUMBER"; GOTO 50
 90 IF G<X PRINT "TRY A BIGGER NUMBER"; GOTO 50
100 PRINT "YOU GOT IT IN", N, " GUESSES"
110 PRINT "LET'S PLAY AGAIN"; GOTO 20
```

5.2.3 FOR Command

The FOR and NEXT commands are used to set up loops in Control Basic. The FOR statement consists of the word FOR followed by a variable or a double byte array element, an equal sign, an expression, the word TO, another expression, and optionally the word STEP and a third expression. The variable or the two byte array element is called the control variable of the loop. It is set to the value of the first expression before entering the loop. The second expression is the limit of the control variable. It is evaluated and stored internally at entry. The third expression is the step size, and it is also evaluated and stored at entry. It can

be positive, negative, or zero. If the step size is not specified, it is assumed to be +1. After the step size is stored, execution of Control Basic continues with the next statement or the next command of a multiple-command statement.

5.2.4 NEXT Command

The NEXT command consists of the word NEXT followed by the control variable of the loop. The control variable is updated by the amount of the step size and is then compared with the limit of the control variable. If it is within the limit (inclusive), Control Basic will loop back to the command that follows the FOR command and repeat the loop. If the updated control variable is outside the limit, the internal storage for this loop is cleared and Control Basic proceeds to execute the statement which follows the NEXT statement. The following example will illustrate the use of FOR-NEXT loops with varying STEP sizes:

```
 10 PRINT "TEST",
 20 FOR T=1 TO 3
 30 PRINT T,
 40 NEXT T
 50 PRINT; PRINT 'COUNT DOWN',
 60 FOR C=100 TO 98 STEP -1
 70 PRINT C,
 80 NEXT C
 90 PRINT '    ...'; PRINT 'STEP',
100 FOR @(3)=10 TO -15 STEP -5
110 PRINT @(3),
120 NEXT @(3)

OK
>RUN
TEST      1      2      3
COUNT DOWN    100     99     98     ...
STEP     10      5      0     -5     -10     -15
```

5.2.5 Notes on FOR-NEXT Loops

The following are several unusual features of Control Basic FOR-NEXT loops which are worth mentioning. Numbers 4 through 7 below are not recommended if the user is attempting to write clean, easily debugged code.

1. The loop will be executed at least once no matter what the initial value, step size, and the limit of the control variable are.

2. If the step size is Ø, the loop will never end unless the control variable is altered inside the loop.

3. After the loop is finished, the control variable will have the final updated value, which is outside the limit.

4. Since Control Basic is interpreted rather than compiled, it is perfectly acceptable to put the NEXT statement physically before the FOR statement as long as there are GOTOs to make Control Basic "see" the FOR before the NEXT.

5. It is also acceptable to have unequal numbers of FORs and NEXTs, as long as there are IFs and GOTOs so that Control Basic will not "see" a NEXT without first encountering a FOR with the same control variable.

6. It is acceptable to have a GOTO out of the loop and later GOTO back into the loop. It is also acceptable to have a GOTO out of the loop and never come back into the loop. In the latter case the loop remains "active"; it takes some stack space but is harmless otherwise.

7. When a new FOR command having a control variable that is the same as an old but still active FOR loop is encountered, the old loop is purged.

8. FOR-NEXT loops can be nested as long as each level uses a different control variable. The depth is limited only by stack space.

9. If a NEXT commmand for the outer loop is encountered inside the inner loop when using nested FOR-NEXT loops, the inner loop is purged automatically.

Note that the above apply to Cromemco Control Basic but do not necessarily apply to other Basic-like languages. For example the following program does not work on many of them:

```
 10 REM  TRY THIS WITH BOTH A>0 AND A<0
 15 INPUT A
 20 FOR J=1 TO 3
 30 IF A<0 GOTO 50
 40 NEXT J
 50 PRINT J
 60 FOR I=1 TO 3
 70 FOR J=1 TO 3
 80 NEXT J
 90 NEXT I
100 GOTO 15
```

Although this program works with Control Basic, it's a bit obscure.

5.3 Subroutines

5.3.1 GOSUB Command

The GOSUB statement is similar to the GOTO statement except that (a) the current statement number and position within the statement are stored internally (this allows control to be transferred back by RETURN); and (b) other commands separated by semi-colons can follow it in the statement. The following are legal GOSUB statements:

```
10 GOSUB 120
20 GOSUB A*10+B
30 GOSUB 120; IF C<=0 GOTO 10
40 REM  IF C>0 THE GOTO ABOVE WILL NOT BE EXECUTED
```

Statement 10 will cause the execution to jump to statement 120; when a RETURN command (see following section) is encountered, control will transfer back to the next following statement or command of a multiple-command statement. Statement 20 will cause the execution to jump to a variable statement number as computed from the value of the expression A*10+B.

5.3.2 RETURN Command

A RETURN command must be the last command in a statement and followed by a carriage return (i.e., it cannot be followed by a semi-colon and other commands). When a RETURN command is encountered, it will cause the execution to jump back to the command following the most recent GOSUB command.

An active FOR-NEXT loop in the calling program is no longer active in the subroutine but will be restored to be active after the subroutine RETURNs to the calling program. Any active FOR-NEXT loop in the subroutine itself will be purged automatically when RETURN is encountered. Variables are always global; therefore subroutines can have no local variables.

GOSUBs can be nested. The depth of nesting is limited only by the stack space and can be no more than 24 levels deep. GOSUBs are also recursive with the limitation that all variables and arrays are global. For example, consider the function F(N) of positive integers N such that:

```
F(1)=1
F(N)=N*F(N-1) for N>1
```

One can write the following test program to illustrate the function F(N) using recursive GOSUBs (underlined words are user-typed):

```
 10 INPUT 'GIVE ME A SMALL POSITIVE INTEGER' N
 20 GOSUB 100
 30 PRINT 'F(', #1,N, ')=', F
 40 STOP
100 IF N=1 LET F=1; RETURN
110 N=N-1; GOSUB 100; N=N+1
120 F=N*F; RETURN

OK
>RUN
GIVE ME A SMALL POSITIVE INTEGER:6
F(6)=720
```

Note that F(N) as defined above is simply the factorial function.

5.3.3 RUN Command

The RUN command consists of the word RUN followed optionally by a page number. The RUN command is a subroutine call on a grand scale. If a page number is given, it calls the Control Basic program SAVEd (or EPROMed) on that page. If a page number is not given, it calls the Control Basic program in the active text area. The RUN and GOSUB commands differ in that a GOSUB command calls a subroutine that is within the same program as the calling routine, whereas a RUN command calls a subprogram which is in a separate file. A subprogram in a separate file has the advantage that it has its own set of line numbers. Since RUN may be used in a statement, the following is a legal program:

```
10 RUN 40
20 PRINT "END OF FACTORIAL"
```

This would run the program given above in section 5.3.2, assuming it had been SAVEd on page 40. Thus, it is easy to see how different programs could share the same set of debugged "library" subprograms.

The RUN command can be nested and is recursive. As it is for GOSUB, the depth of nesting is limited by stack space, and one should bear in mind that all variables and arrays are global.

5.3.4 AUTORUN Command

The AUTORUN command is a feature of model MCB-216 Control Basic only. If the first line in a Basic program stored on page 10H begins with the command AUTORUN, then that program (on page 10H) will run automatically each time the computer is reset.

The console serial port of the computer will automatically be set to 9600 baud. However, it is easy for the program to change the baud rate to another value.

For example, if the following program is stored on page 10H,

```
10 AUTORUN
20 OUT(0) = %84
30 RUN %11
```

the baud rate will be changed to 300 with one stop bit (see the SCC Input/Output Register Description in the SCC Instruction Manual), and then the program beginning on page 11H will be RUN. This process occurs every time the computer is hardware reset.

It should be noted that during normal execution (i.e., in all cases except after a hardware reset or power-on) the line beginning with the word "AUTORUN" is treated as a REMark.

5.3.5 STOP Command

The STOP command consists of the word STOP followed by a carriage return. It cannot be followed by a semi-colon and other commands on the same line. The STOP command goes with the RUN command in the same way that RETURN goes with GOSUB. The principle difference between the two sets of commands is that RUN-STOP may be used to call a subprogram in a separate file with its own set of line numbers.

STOP returns control from a subprogram to the calling program (which will continue RUNning); however, if the program which is RUNning when STOP is encountered is the current program in the text area, execution is terminated and the prompt is given. If the end of file is reached in a subprogram before a STOP statement is encountered, Control Basic will also return to direct mode instead of to the calling program. An example will illustrate the way STOP passes control from subprogram to calling program. The following is a program which has been previously stored (SAVEd or EPROMed) at page 32:

```
10 PRINT "THIS IS A PROGRAM ON PAGE 32"
20 RUN 33
30 PRINT "END OF PROGRAM ON PAGE 32"
40 STOP
```

and the next listing is a program which has been previously stored at page 33:

```
10 PRINT 'BUT THIS PROGRAM IS ON PAGE 33'
20 STOP
```

RUNning the program on page 32 will produce the following result:

```
OK
>RUN 32
THIS IS A PROGRAM ON PAGE 32
BUT THIS PROGRAM IS ON PAGE 33
END OF PROGRAM ON PAGE 32
```

Note that this will have no effect on the current program in the text area if there is one. This means that one does not need to LOAD a program to RUN it, a most useful feature (see section 5.3.3 for more information).

5.3.6 CALL Command

The CALL command is used to call a machine language subroutine. The word CALL is followed by an expression and then a list of arguments enclosed in parentheses. The value of the expression is the absolute address of the entry of the machine language subroutine. The argument list consists of none, or one or more arguments. Each argument is a variable, a one byte array element, or a two byte

array element. Arguments are separated from one another by commas, and the entire list is enclosed in a pair of parentheses.

The machine language subroutine can be stored in memory by a loader, an assembler, a monitor program, or by Control Basic itself. To store a machine language subroutine with Control Basic, one may use the LET or PUT commands. The example of section 5.1.2 illustrates this. We can now CALL this machine language routine to print the word "HOWDY" on the console with the statement:

```
30 CALL %800
```

[This example will work only if there is a console printing routine whose starting address is at ECA4H, as there is for model CB-308 Control Basic.] Note that no arguments were passed in this CALL statement.

Arguments are passed by addresses and these addresses are stored in the stack before Control Basic passes control to the machine language subroutine. The arguments are stored in a "Last In-First Out" or LIFO format; thus, the last argument on the statement line will be the first one POPed off the stack. The number of arguments is passed in the C register. The subroutine must POP the stack (C) times even if it does not need the arguments. Other than this arrangement of the stack, the subroutine can change all registers without any limitation. When the subroutine is finished, it uses a RET (Z80 instruction) to POP the stack once more, which also returns control to Control Basic. Another example illustrating argument passing is:

```
10 CALL %2000 (A, &(3*B+1), @(3))
```

In this example the subroutine at memory location 2000H is called and passed the three arguments, A, &(3*B+1), and @(3). Upon entrance to the machine routine, the C register contains the number 3. The stack pointer points to the address of @(3), which is followed by the address of &(3*B+1), the address of A, and finally the return address.

5.4 Input and Output Commands

5.4.1 PRINT Command

The PRINT command is used to print all types of output on the console device, including strings, constants in both decimal and hexadecimal, and the values of variables, arrays, and expressions. Its format is the word "PRINT" followed by a list of items separated by commas. The items in the list may include any of the following:

1. Constants (decimal, hexadecimal, ASCII).
2. Variables (A through Z and AØ through ZØ).
3. One-byte, two-byte, and string array elements.
4. Expressions formed from any of the above, arithmetic and compare operators, and the legal functions.
5. Strings surrounded by matched pairs of single or double quotes.
6. Format control characters described later in this section.

A sample PRINT command follows which shows several of these items. Note the use of both quoted and unquoted strings, a variable, and a constant expression:

```
10 PRINT 'THIS ', "OR THAT ", $(A+3), X, 2*3-5/4

OK
>RUN
THIS OR THAT PLUS     -9      5
```

The string array $(A+3) has been set elsewhere to the string "PLUS", and the variable X has been previously set equal to -9.

The items in the print list can also be format controls. The allowable characters to use as format controls are summarized briefly below, followed by a more detailed description and some illustrations.

#	specify leading spaces or hexadecimal
%	specify number of hexadigits
,	(preceding) specify additional leading spaces
,	(succeeding) specify no CR-LF at end of item
_	tab to a specific column

The character "#" followed by a number n means print leading spaces (if necessary) to make numbers at least n spaces wide. Spaces can also be added by means of extra commas between items. The format control "#%%" means print numbers in 4 hexadigits, whereas "#%" means print the low order byte of a number only, in 2 hexadigits. A format control stays effective until changed by another format control or until the PRINT command ends. The default format if no control is specified is #6. The following example will illustrate these points:

```
10 PRINT 1, -2, 345, -6789
20 PRINT #3, 1, -2, 345, -6789
30 PRINT #%%, 1, -2, 345, -6789
40 PRINT #%%, 1,, -2,, 345,, -6789
50 PRINT #%, 1,, -2,, 345,, -6789

OK
>RUN
     1    -2   345 -6789
  1 -2345-6789
0001FFFE0159E57B
0001 FFFE 0159 E57B
01 FE 59 7B
```

The underline or back-arrow character (ASCII 5FH) is used as a format control to indicate tabbing to a particular column on the console device. It should be followed by a number n which is the column number; this will cause the cursor or print head to move to that position. The print positions are numbered from left to right beginning with 1. If n is 1, the cursor (print head) will move to the left margin without a line feed. For model MCB-216 of Control Basic, the cursor will move to position n independent of its present position, i.e, it may move either left or right. Model CB-308, however, allows movement only to the right of the present position.

The PRINT command generates a carriage return and line feed at the end of the last item in the command. However, if there is a comma at the very end of the line, the CR-LF is not generated. The following example will illustrate both the tab feature just discussed and the suppressed CR-LF:

```
10 PRINT 'THIS',
20 PRINT ' IS ',
30 PRINT 'PRINTED ON ONE LINE'
40 PRINT 'WE HAVE', _10, 'O = O', _10, '- / /'

OK
>RUN
THIS IS PRINTED ON ONE LINE
WE HAVE  θ ≠ Ø
```

Note that statement 40 of this example is somewhat special purpose in that it will work correctly only with model MCB-216 and with a console device which will print non-destructive spaces such as a teletype.

5.4.2 INPUT Command

The INPUT command is used to prompt the operator to type in an expression or a string to be saved by the program in a variable, an array element, or a string array. The word INPUT is followed by a list of items (any of the above) separated by commas. The items may also be strings (quoted in single or double quotes) or tab controls for the terminal. These items look and behave exactly like those used in a PRINT statement (see section 5.4.1). For example, the underline or back-arrow character (ASCII 5FH) may be used to specify INPUT at a particular column on the console terminal. Following are two examples which will illustrate some of the features of INPUT statements. The first example shows tab control and the second shows how to input character strings (underlined sections are to be user-typed):

```
10 INPUT "INPUT 'A' HERE" A, _30, "AND 'B' THERE" B
20 PRINT #0, A, _30, B

OK
>RUN
INPUT 'A' HERE:123
                                    AND 'B' THERE:456
123                                 456
```

Note that the quoted letters 'A' and 'B' are considered part of the strings shown because they are in single quotes and the strings are in double quotes. This could be reversed (letters in doubles; strings in singles); however, if both the strings and the letters are quoted with the same symbol, Control Basic will not understand and will print "WHAT?" (see Chapter 6).

```
10 INPUT "GIVE ME A STRING: ", $(23)
20 PRINT "YOU TYPED '", $(23), "'"
30 INPUT X, @(11)
40 PRINT X, @(11)

OK
>RUN
GIVE ME A STRING: THIS IS A STRING.
YOU TYPED 'THIS IS A STRING.'
X:3*5+5*6
@(11):X+1
     45      46
```

In this example the operator answered the first input inquiry by typing "THIS IS A STRING.", answered the second one by "3*5+5*6", and answered the third one by "X+1". Note that in statement 30 where a variable and an array element are the input items, the names of the variables, "X" and "@(11)", are also printed automatically by Control Basic as prompts. If the operator makes an error in response to the prompt, Control Basic will also retry that part of the INPUT and reprint the prompt. When the input item is a string array as it is in statement 10, there is no automatic prompt and whatever the operator types is always accepted. We now rerun the same program giving several unusual inputs to see their effect:

```
OK
>RUN
GIVE ME A STRING: %?#!*
YOU TYPED '%?#!*'
X:%?#!*
WHAT?
X:2/Ø
HOW?
X:@(8ØØØ)
SORRY
X:2+1
@(11):(3+4)*5)
WHAT?
@(11):(3+4)*5
      3    35
```

Programmers not wishing to take advantage of the automatic variable name prompt can supply their own in a pair of single or double quotes preceding the variable or array name without a comma between them. The next example will illustrate:

```
10 INPUT "PLEASE GIVE ME YOUR ","WEIGHT (IN LBS)" X
20 LET Y=(10*X+11)/22
30 PRINT 'THAT IS ABOUT ', #Ø, Y, " KILOGRAMS"
```

```
OK
>RUN
PLEASE GIVE ME YOUR WEIGHT (IN LBS):HUNDRED AND FORTY
WHAT?
WEIGHT (IN LBS):14Ø
THAT IS ABOUT 64 KILOGRAMS
```

Note that only the prompt "WEIGHT (IN LBS)" is repeated; the string "PLEASE GIVE ME YOUR " is separated from the input item by a comma and is thus not repeated following an incorrect response. The expression in statement 20 handles the unit conversion and rounding in integer arithmetic.

5.4.3 OUT Command

The OUT command is used to output a byte of data to any I/O port. The format of the command is

```
OUT(x) = y
```

where x is the port number and is an expression with a value between Ø and 255 (Ø-FFH), and y is an expression also with a value between Ø and 255, which is the actual byte of data to be output.

The counterpart of OUT is IN, but IN is a function rather than a command. The input function IN(x) is used to read data or status information in from an I/O port. The port number x is again an expression with a value between Ø and 255 (Ø-FFH). This function, like any other function, cannot stand for a command by itself. Thus in the following example illustrating the use of the OUT command and the IN function, statement 1Ø is invalid while the other statements are legal.

```
1Ø IN(3)
2Ø PRINT #%, IN(3),, SIZE,, LOC
3Ø LET A=AND(IN(B+1),%4Ø)
4Ø OUT(B+1)=A
```

5.5 Non-Executable Commands

5.5.1 REM Command

The word "REM" may be used in a Control Basic statement to denote a REMark line. This "command" is non-executable and everything following the REM on that statement line (preceding the CR) is ignored by the Interpreter. REM may also appear on the same line with other commands in a multiple-command statement; everything preceding the REM command will be executed but the rest of the line is again ignored. There are several commands which must be the last command on a line; these cannot be followed by even a REM command. They are: GOTO, RETURN, and STOP. [Also note that the word "REMARK" may be used instead of "REM" if you wish since the letters "ARK" are ignored anyway.] The following example illustrates the use of REM:

```
10 REM
20 REM   *** Title of Program ***
30 REM
40 PRINT "RECIPE"; REM PRINT "COOKBOOK"
```

In this example statements 10, 20, and 30 will be ignored during execution of the program; statement 40 will cause Control Basic to print the word "RECIPE" but not the word "COOKBOOK".

5.5.2 AUTORUN Command

Note that AUTORUN is treated exactly like a REMark in every case except when the computer hardware reset is pressed, in which case AUTORUN causes the Control Basic program at page 16(%10) to begin executing. AUTORUN is a feature of the MCB-216 model of Control Basic only. For more information see section 5.3.4.

5.6 Editing Commands (direct only)

5.6.1 NEW Command

The NEW command is used to delete the current program stored in the text area. It is a direct command which is executed by typing "NEW <CR>" from the console device. NEW is executed automatically upon entering Control Basic initially and upon issuing a LOAD command (section 5.7.4).

5.6.2 LIST Command

The LIST command is used to print some or all of the statements of the current program in the text area in numerical order on the console device. It is used frequently in entering, editing, and verifying Control Basic programs. There are several legal forms of LIST; the simplest is to type "LIST <CR>", which will then list the entire current program from beginning to end. A more complete format is

```
LIST x,n
```

where x is the beginning line number (a constant) and n is the number of lines to be LISTed. Both these values are optional, and the comma (,) should be omitted if n is. The following examples will illustrate these points further:

LIST 120	prints all statements beginning with line number 120.
LIST 120,3	prints at most (i.e., fewer will be printed if fewer are present) 3 statements beginning with line 120.
LIST ,20	prints at most 20 statements starting with the beginning line of the current program.
L.90,1	prints only statement number 90, if there is one. Note the abbreviated form of LIST (Chapter 4).

5.7 Storage and File Commands (direct only)

5.7.1 LOCK Command

The LOCK command is used to change the boundary between the current program text area and the area reserved for SAVEing program files. A sample LOCK command is

```
LOCK 40
```

which will set the boundary between the text area and the SAVE area at 2800H (page 40D=page 28H). Thus pages 4 through 39 (CB-308) or 34 through 39 (MCB-216) become allocated to the current program text and pages 40 through 255 can be used to SAVE files. The LOCKed value (see "TXTLMT" in Memory Allocation, section 8.1.1 or 8.2.1) is set when Control Basic is cold started to 32 (20H) for model CB-308 and 36 (24H) for model MCB-216. If these values are lowered by means of the LOCK command after a program has been entered into the text area, Control Basic will print "SORRY" if the page number specified would include part of that current program. For example, suppose the current program resides in pages 4 through AH. The command "LOCK %A" will print "SORRY" but the command "LOCK %B" will be accepted and executed.

IF the LOCK command is used to increase the size of the current program area, it is up to the user to make sure that this does not endanger any previously SAVEd files. The allowable range of the LOCK command is 5-255 (5H-FFH) for CB-308 and 35-255 (23H-FFH) for MCB-216.

5.7.2 SAVE Command

The SAVE command is used to save the current program in a particular page of memory. The page number given should be above the LOCKed text area and generally coincident only with read/write memory space. A sample SAVE command is

```
SAVE %43
```

where the current program will be saved in memory beginning at 4300H. When the SAVE command is finished, Control Basic will print a message indicating how many pages of memory were used. For

example the message

SAVED ON PAGE %43 TO %45

means that the current program now occupies all of pages 43H and 44H, and all or part of page 45H. If the copy does not compare with the original (for example, if there is no RAM at the specified page), Control Basic will print "SORRY". "SORRY" will also be printed if one attempts to SAVE a program in pages Ø through one less than the LOCKed number (i.e., Ø through 31 for CB-3Ø8 and Ø through 35 for MCB-216, as they are initially configured). This region is reserved by Control Basic for variables, stack, and the current text (see section 8.1.1 or 8.2.1).

The SAVE command may also be used in conjunction with a 32K Bytesaver to program 2716 (2516) PROMs. Remember to turn on the program power of the Bytesaver board before issuing the SAVE command. Used in this way, SAVE will take several minutes to execute due to the wait states required by the 2716's while programming.

5.7.3 EPROM Command

The EPROM command may be used to save the current program in blank or erased 27Ø8 PROMs. The PROMs must be loaded onto a Cromemco Bytesaver board residing in currently addressed memory. A convenient place to have the PROMs reside when using model CB-3Ø8 of Control Basic is from addresses FØØØH to FFFFH, or simply the top half of the Bytesaver which contains CB-3Ø8. A sample EPROM command is

EPROM 64

where the current program will be saved in PROMs addressed beginning at 4ØØØH (page 64D=page 4ØH). The page number must be divisible by 4 (since 27Ø8's are 1K bytes long, or 4 pages). This command will take several minutes to execute, where the amount of elapsed time will depend on the length of the program being saved.

When the EPROM command is finished, Control Basic will print a message indicating how many pages of memory were used (always a multiple of four pages).

If the copy does not compare with the original, Control Basic will print "SORRY". See the SAVE command, section 5.7.2, for information on programming 2716 (2516) PROMs.

5.7.4 LOAD Command

Programs which have been saved with the SAVE or EPROM commands may be copied back to the text area for editing by means of the LOAD command. This process performs an automatic "NEW"; in other words the current program in the text area is deleted. An example use of LOAD is

```
LOAD %43
```

where the file stored on page 43H (or at 4300H in memory) is copied into the text area. If no program is stored at page 43H, Control Basic prints "SORRY". Also, since the length of any given stored program is contained in the first two bytes of the first page (see section 8.1.1 or 8.2.1), Control Basic knows immediately if it will fit into the text area. If it will not fit, "SORRY" is printed and no LOADing takes place. The size of the text area may be increased (to allow for LOADing very long programs) by means of the LOCK command, section 5.7.1.

5.8 Console Terminal Commands (direct only)

5.8.1 WIDTH Command

The WIDTH command may be used to compensate for the non-standardization of console terminals by allowing the user to set both a soft and hard screen width or margin. A sample WIDTH command is

```
WIDTH 72,80
```

which sets the soft margin at column 72 and the hard margin at column 80. When the soft margin has been exceeded during output, Control Basic will generate a carriage return-line feed sequence after any space character (and thus, at the ends of words). When the hard margin has been exceeded, it generates an immediate CR-LF. The values above are good choices for an 80-column CRT. The default WIDTH for Control Basic model CB-300 is 60,60 and for MCB-216 it is 72,72. The allowable usable range for the first number is 0-129 and for the second is 0-255.

Since WIDTH is a direct command, it cannot be set by a statement in a program. However, it is sometimes useful to be able to change the screen width in a particular program. One method of doing this is to use the PUT command to alter the RAM locations where the soft and hard margins are stored (see "MARGN" in Memory Allocation, section 8.1.1 or 8.2.1). An example of this (for CB-300) is

```
10 PUT(%3E2) = 72,80
```

which has the same effect as the WIDTH example above. A similar command may be used to set the margins for MCB-216:

```
10 PUT(%21E3) = 72,80
```

5.8.2 NULL Command

The NULL command is used to set the number of nulls transmitted by Control Basic after any carriage return-line feed sequence. This allows the user to set carriage return time for hardcopy devices such as teletypes. A sample NULL command is

```
NULL 3
```

which causes Control Basic to transmit 3 null characters after every CR-LF. The default value for NULL upon entering Control Basic (either model) is Ø. The allowable, usable range of the value is Ø-128.

5.9 Monitor-Entrance Commands (direct only)

5.9.1 QUIT Command

The QUIT command is used to return control to a system monitor. For Control Basic model CB-308 this is assumed to be the Z80 Monitor (model ZM-108) located at E000H in memory. Typing "QUIT" will cause a jump from Control Basic to E008H, the warm start location for ZM-108. It is acceptable for programmers to use their own monitor if desired; however, there must still be a PROM containing a JP or CALL instruction at E008H to pass control to this monitor. One may use one of the Control Basic restarts to go back to CB-308 after finishing with use of the monitor. These are summarized below along with their entrance addresses:

name	address	function
initial start	E400H	initializes everything; moves I/O routines to RAM.
cold start	E406H	initializes everything except I/O.
new program warm start	E424H	warm start which does not re-initialize margins or EOL nulls, etc. but does clear program area; same effect as NEW command.
stored program warm start	E42FH	normal warm start for CB-308; program is preserved, but stack pointer is re-initialized.

Thus, a typical re-entrance to CB-308 after using the Z80 Monitor would be the command "GE42F <CR>" (Go to E42FH).

Control Basic model MCB-216 also has a QUIT command. However, in this case the Monitor has been merged with and is a part of Control Basic. Therefore, typing "QUIT" will pass control to the warm start of the Monitor, and typing "B <CR>" will pass control from the Monitor back to the Control

Basic warm start (see also section 8.2.4). The stored program will never be cleared because the Monitor always enters at the MCB-216 "stored program warm start." This is shown below along with the entrance addresses of the other restarts:

name	address	function
initial and cold start	423H	initializes everything and sets baud rate of console terminal.
new program warm start	447H	warm start which does not re-initialize margins or console baud rate, etc. but does clear program area; same effect as NEW command.
stored program warm start	452H	normal warm start for MCB-216; program is preserved but stack pointer is re-initialized. The Monitor "B" command causes a jump to this location.

5.9.2 RDOS Command

The RDOS command is a feature of model MCB-216 only. It is provided for users of this version of Control Basic who also have a 4FDC Disk Controller board in their system and one or more disk drives. Typing "RDOS" as a direct command causes control to pass from Control Basic to the warm start of the resident ROM monitor on the 4FDC known as RDOS. The disk read or write commands of RDOS may then be used to load programs into or save them from memory, respectively. See the RDOS Manual for more information.

To return to Control Basic after using RDOS, type "G452 <CR>" to enter at the "stored program warm start" listed in section 5.9.1 above. You may also return via one of the other MCB-216 restarts listed in that section.

CHAPTER 6

Error Reporting

Only three error messages are given by Control Basic. These are printed out on the console as soon as they are detected. The statement containing the error is printed below this with a question mark inserted at the point where the error is detected. If the error occurs in a RUNning program, execution is stopped. Since Control Basic is a pure interpreter, though, the presence of an error in a statement may not be detected until that line is actually encountered during execution. Error messages are also given for illegal direct commands as well as illegal statements in programs.

6.1 WHAT? Error Message

The error message "WHAT?" means that Control Basic does not understand you. It can occur for a wide variety of reasons, several of which are listed below as examples. Many other examples illustrating the cause of error messages will be found throughout this manual.

```
WHAT?
 210 P?TINT "THIS"      where "PRINT" is
                        mistyped

WHAT?
 260 LET C=(3+4?,X=4    where a ")" is missing
```

6.2 HOW? Error Message

The error message "HOW?" means that Control Basic understands you but does not know how to do it. For example, this message will occur in the factorial example of section 5.3.2 for any input number greater than 7. This is because the answer will then exceed the allowable maximum integer 32767. Other examples illustrating the cause of the "HOW?" message are:

```
HOW?
 310 LET A=B*C?+2        where B*C is larger than
                         32767

HOW?
 380 GOTO 412?           where statement 412
                         doesn't exist
```

6.3 SORRY Error Message

The error message "SORRY" means that Control Basic understands you and knows how to do it but it is still unable to do it because of memory or other physical limitations. If we replace statement 120 in the example of section 5.3.2 with

```
120 F=N+F; RETURN
```

the "SORRY" message will be given for any input number greater than 24. This is because recursive GOSUBs can be nested to no greater than 24 levels deep. Other examples of the cause of the "SORRY" message follow:

```
SORRY
 400 LET A=@(9000)?      where @(9000) is not
                         within the current
                         LOCKed text area

>SAVE 3
SORRY                    where page 3 is illegal
                         to use for SAVEing a
                         program
```

In general misusing, or using illegal, commands will cause Control Basic to print "SORRY". The message will also be given by programs which contain direct commands used as statements.

CHAPTER 7

Prime Number Generator - A Sample Program

The following Control Basic program calculates the prime numbers between 1 and 10,000, and is provided to illustrate the power and ease of use of Control Basic.

The "PRINT" statements in this program output the calculated prime numbers to a console or printer that is interfaced to the microprocessor system.

```
10 PRINT 1,
20 PRINT 2,
30 FOR N=3 TO 10000
40 FOR I=2 TO N/2
50 IF N=(N/I)*I GOTO 80
60 NEXT I
70 PRINT N,
80 NEXT N
```

The first two prime numbers are explicitly printed out by the first two statements of the program. Subsequent numbers, N, are checked for being prime by testing to see if they are exactly divisible by any integer, I. If any N is exactly divisible, it is discarded as not being a prime.

CHAPTER 8

Supplementary Information

8.1 Control Basic model CB-308

8.1.1 Memory Allocation (CB-308)

The listing which follows is a detailed outline of memory allocation for Control Basic model CB-308. The table on page 1 maps an entire 64K system. Of this, the minimum required is RAM from 180H to 1FFFH and the Control Basic Interpreter itself in ROM from E400H to EFFFH. The rest of memory (may be either ROM or RAM) except the unused portion from 0 to 17FH may be used to store program files and routines. Page 2 of the listing is a detailed breakdown of the RAM from 200H to 3FDH, which contains Control Basic variables, stack, input buffer, and console parameters.

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0001
3K Control Basic Variables and Memory Allocation

                0003 ;
                0004 ;       Copyright (c) 1976, 1979 Cromemco, Inc.
                0005 ;
                0006 ;
                0007 ;  *** Memory Usage (probably RAM unless otherwise noted) ***
                0008 ;
                0009 ;  0000-017F are not used
                0010 ;  0180-01FF are for the I/O routines
                0011 ;  0200-03FF are for variables, input line, and stack
                0012 ;  0400-1FFF are for Control Basic text & array @(I)
                0013 ;  2000-DFFF are not used but may be used for saved files
                0014 ;  E000-E3FF are for system monitor and not used here (ROM)
                0015 ;  E400-EFFF are for Control Basic Interpreter (ROM)
                0016 ;  F000-FFFF are not used but may be used for saved files


(0180)          0020 IORAM   EQU     180H            ; START OF I/O ROUTINES IN RAM
(0200)          0021 BOTRAM  EQU     200H            ; "BOTTOM OF RAM" FOR VARIABLES, ETC.
(0400)          0022 TXTRAM  EQU     400H            ; BOTTOM OF "TEXT RAM"
(2000)          0023 TOPRAM  EQU     2000H           ; "TOP OF RAM" USED BY CONTROL BASIC
(E400)          0024 BOTROM  EQU     0E400H          ; "BOTTOM OF ROM" CONTAINING CB
(F000)          0025 TOPROM  EQU     BOTROM+0C00H    ; FIRST FREE LOCATION PAST "TOP OF
                0026                                 ;   ROM" (USED BY RND FUNCTION)
(E008)          0027 MONITR  EQU     0E008H          ; MONITOR WARM-START POINT
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0002
3K Control Basic Variables and Memory Allocation

0000'                0030          ORG     BOTRAM
0200   (0068)        0031 VARBGN:  DEFS    2*(26*2)      ; VARIABLE A-Z AND A0-Z0 STORAGE
0268   (0001)        0032          DEFS    1             ; EXTRA BYTE FOR INPUT BUFFER
0269   (0085)        0033 BUFFER:  DEFS    133           ; STORAGE FOR INPUT LINE BUFFER
       (02EE)        0034 BUFEND   EQU     $             ; END OF INPUT BUFFER
02EE   (0028)        0035          DEFS    40            ; EXTRA BYTES FOR STACK
0316   (00C8)        0036 STKLMT:  DEFS    200           ; SOFT LIMIT FOR STACK
       (03DE)        0037 STACK    EQU     $             ; TOP OF STACK
03DE   (0002)        0038 IOINS:   DEFS    2             ; STORAGE FOR AN I/O INSTRUCTION
03E0   (0001)        0039 IORET:   DEFS    1             ; STORAGE FOR THE RETURN INSTRUCTION
03E1   (0001)        0040 NULLS:   DEFS    1             ; NUMBER OF NULLS AFTER CR-LF
03E2   (0001)        0041 MARGN:   DEFS    1             ; SOFT TERMINAL WIDTH
03E3   (0001)        0042          DEFS    1             ; HARD TERMINAL WIDTH
03E4   (0001)        0043 OCMG:    DEFS    1             ; COLUMN COUNTER FOR MARGINS
03E5   (0001)        0044 OCSW:    DEFS    1             ; SWITCH FOR VISIBLE OUTPUT
03E6   (0002)        0045 CURRNT:  DEFS    2             ; CURRENT LINE NUMBER STORAGE
03E8   (0002)        0046 STKRUN:  DEFS    2             ; SAVES STACK POINTER DURING 'RUN'
03EA   (0002)        0047 STKGOS:  DEFS    2             ; SAVES STACK POINTER DURING 'GOSUB'
       (03EC)        0048 VARNXT   EQU     $             ; TEMP. STOR. OF CURRENT 'NEXT' VAR.
       (03EC)        0049 CALDE    EQU     $             ; TEMP. STOR. FOR ADDR. FOR 'CALL'
03EC   (0002)        0050 STKINP:  DEFS    2             ; SAVES STACK POINTER DURING 'INPUT'
03EE   (0002)        0051 LOPVAR:  DEFS    2             ; ADDRESS OF 'FOR' LOOP VARIABLE
03F0   (0002)        0052 LOPINC:  DEFS    2             ; 'STEP' SIZE FOR 'FOR' LOOP
03F2   (0002)        0053 LOPLMT:  DEFS    2             ; LIMIT OF 'FOR' LOOP VARIABLE
03F4   (0002)        0054 LOPLN:   DEFS    2             ; TEMP. STOR. OF 'CURRNT' IN 'FOR'
03F6   (0002)        0055 LOPPT:   DEFS    2             ; TEXT POINTER TEMP. STOR. IN 'FOR'
03F8   (0002)        0056 RANPNT:  DEFS    2             ; ADDR. POINTER USED IN RND FUNC.
03FA   (0002)        0057 TXTBGN:  DEFS    2             ; STORAGE OF TEXT POINTER ADDR.
03FC   (0002)        0058 TXTLMT:  DEFS    2             ; TEXT AREA LIMIT (SAME AS 'TOPRAM'
                     0059                                ;   UNLESS CHANGED BY 'LOCK')


03FE                 0063          ORG     TXTRAM
0400   (0002)        0064 TXTUNF:  DEFS    2             ; POINTER TO UNFILLED TEXT AREA
0402   (0002)        0065 TEXT:    DEFS    2             ; START OF ACTUAL TEXT (STORAGE
                     0066                                ;   OF FIRST LINE NUMBER)
```

Also shown on page 2 is the beginning of the text area for the current program. The text area is organized as follows; any stored program (by means of the SAVE or EPROM commands) is organized this same way:

1. The first two bytes of the first page point to the first available byte after the program text relative to 400H.

2. The next two bytes contain the line number of the first line in binary (low byte-high byte). This is followed by the text of the first line and a single 0DH, or CR, at the end of that line.

3. Subsequent lines of text duplicate the form of this first line: 2-byte binary line number, ASCII text, CR-byte.

4. The final line of the program is followed by 2 additional bytes, xxH-FFH, where xx is an indeterminate value.

5. Following these are two bytes which contain the value of the array element @(Ø) (low byte-high byte binary). The value stored in TXTUNF (location 4ØØH) will be the address of the next byte following this array element. (In the case of a stored program file, this address is stored in the first two bytes of the first page of the file, which is equivalent to the location TXTUNF.)

Note that the foregoing structure of a program is the same regardless of whether it is a stored file or the current program in the text area. However, for a text area file only, subsequent values of the array @(I), where I=1,2,3,..., are stored in the text area following @(Ø) (two bytes per array element). If the array @(I) grows too large to fit in the text area, Control Basic will not allocate memory past the LOCKed top and will instead print "SORRY". Use the LOCK command to enlarge the text area in such a case.

Also, note that for stored program files the first two bytes point to the first available memory location relative to 4ØØH. Thus for example, the two bytes D7-Ø5 (low-high) stored at location 43ØØH (page 43H) mean that the first available memory location following that file is at 44D7H (5D7H-4ØØH+43ØØH). More information on arrays will be found in section 3.2.

8.1.2 I/O Drivers (CB-3Ø8)

The listing below is for the I/O drivers of Control Basic model CB-3Ø8 as found in RAM. These routines are stored within the Control Basic ROMs at E4ØØH and are moved to RAM at 18ØH as the first operation of CB-3Ø8. They are provided in RAM so that a user may change them using the Z8Ø Monitor or Control Basic itself to handle either unusual I/O devices or devices at different ports. An example follows.

Suppose the user would like to connect the I/O device (a CRT) to the computer through an I/O card whose base port address is 2ØH. The user's computer is set up to begin operation in the Z8Ø

Monitor program which initializes and uses the CRT at 20H. To enable Control Basic to use the CRT as well, the following steps must be taken:

GE400/E406	(Go into Control Basic and break after the I/O routines have been moved to RAM.)
SM 18B 1D7 00.20 SM 197 1DC 01.21	(Substitute in memory at those addresses which contain a port number.)
GE406	(Continue at Control Basic cold start.)

The user will now be operating Control Basic through a CRT connected to I/O port base address 20H.

Below are the complete Control Basic I/O Driver routines.

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0001
3K Control Basic I/O Routines in RAM

0000'                  0002          ORG     180H

      (0000)           0004 CSTAT    EQU     0               ; CONSOLE STATUS PORT (IN)
      (0001)           0005 CDATA    EQU     CSTAT+1         ; CONSOLE DATA PORT (IN/OUT)
      (0080)           0006 TBE      EQU     80H             ; TRANSMITTER-BUFFER-EMPTY MASK
      (0040)           0007 RDA      EQU     40H             ; RECEIVER-DATA-AVAILABLE MASK
      (007F)           0008 PSTRIP   EQU     7FH             ; PARITY-STRIP MASK

      (0003)           0010 CTRLC    EQU     03H             ; ASCII CONTROL-C CHARACTER
      (000A)           0011 LF       EQU     0AH             ; ASCII LINE FEED
      (000D)           0012 CR       EQU     0DH             ; ASCII CARRIAGE RETURN
      (000F)           0013 CTRLO    EQU     0FH             ; ASCII CONTROL-O CHARACTER
      (0020)           0014 SPC      EQU     20H             ; ASCII SPACE CHARACTER

      (03E1)           0016 NULLS    EQU     3E1H            ; NO. OF NULLS AFTER CR-LF (1 BYTE)
      (03E2)           0017 MARGN    EQU     3E2H            ; SOFT TERMINAL WIDTH (1 BYTE)
                       0018                                  ; HARD TERMINAL WIDTH (1 BYTE)
      (03E4)           0019 OCMG     EQU     3E4H            ; COLUMN COUNTER FOR MARGINS (1 BYTE)
      (03E5)           0020 OCSW     EQU     3E5H            ; SWITCH FOR VISIBLE OUTPUT (1 BYTE)
                       0021                                  ; (THIS IS CHANGED BY TYPING CTRL-O)
      (E42F)           0022 RSTART   EQU     0E42FH          ; CONTROL BASIC WARM-START LOCATION


0180  3E0D             0025 CRLF:    LD      A,CR            ; *** CR-LF ROUTINE ***
                       0026 ;
0182  F5               0027 OUTCHR:  PUSH    AF              ; *** OUTPUT CHARACTER ROUTINE ***
0183  3AE503           0028          LD      A,(OCSW)        ; CHECK VISIBLE/INVISIBLE SWITCH
0186  B7               0029          OR      A               ; IF BIT-7 IS SET, OUTPUT CHARACTER
0187  F2D401           0030          JP      P,ENDOUT        ; IF NOT SET, DON'T OUTPUT CHARACTER
018A  DB00             0031 OUTC10:  IN      A,CSTAT         ; CHECK CONSOLE STATUS
018C  E680             0032          AND     TBE             ; GET TRANSMITTER-BUFFER BIT
018E  CA8A01           0033          JP      Z,OUTC10        ; LOOP TO WAIT IF NOT READY
0191  F1               0034          POP     AF              ; RETRIEVE CHARACTER
0192  E5               0035          PUSH    HL              ; SAVE REGISTERS
0193  F5               0036          PUSH    AF              ; /
0194  E67F             0037          AND     PSTRIP          ; STRIP PARITY BIT
0196  D301             0038          OUT     CDATA,A         ; OUTPUT CHARACTER TO CONSOLE
0198  FE0D             0039          CP      CR              ; CHECK FOR CARRIAGE RETURN
019A  21E403           0040          LD      HL,OCMG         ; POINT TO COLUMN-COUNTER
019D  C2A201           0041          JP      NZ,OUTC20       ; SKIP IF CHARACTER IS NOT CR
01A0  3601             0042          LD      (HL),1          ; INIT. COLUMN-COUNTER IF NEW LINE
01A2  FE20             0043 OUTC20:  CP      SPC             ; CHECK FOR PRINTABLE CHARACTER .
01A4  DABB01           0044          JP      C,OUTC40        ; SKIP IF NOT PRINTABLE
01A7  7E               0045          LD      A,(HL)          ; GET COLUMN-COUNTER
01A8  34               0046          INC     (HL)            ;   AND INCREMENT IT FOR PRINTABLES
01A9  2AE203           0047          LD      HL,(MARGN)      ; H GETS HARD AND L GETS SOFT MARGIN
01AC  BC               0048          CP      H               ; CHECK FOR HAVING REACHED HARD MARGIN
01AD  CAB801           0049          JP      Z,OUTC30        ; FORCE OUTPUT CR-LF IF SO
01B0  BD               0050          CP      L               ; CHECK FOR HAVING REACHED SOFT MARGIN
01B1  FABB01           0051          JP      M,OUTC40        ; SKIP IF NOT YET REACHED
01B4  F1               0052          POP     AF              ; GET CHARACTER AGAIN
01B5  F5               0053          PUSH    AF              ;   AND PUT BACK
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0002
3K Control Basic I/O Routines in RAM

01B6  FE20        0054          CP      SPC             ; CHECK FOR SPACE CHARACTER (DO CR-LF
                  0055                                  ;   AT FIRST SPACE AFTER SOFT MARGIN)
01B8  CC8001      0056 OUTC30:  CALL    Z,CRLF          ; OUTPUT CR-LF IF FOUND
01BB  F1          0057 OUTC40:  POP     AF              ; RESTORE REGISTERS
01BC  E1          0058          POP     HL              ; /
01BD  FE0D        0059          CP      CR              ; CHECK FOR CARRIAGE RETURN
01BF  C0          0060          RET     NZ              ; RETURN TO CONTROL BASIC IF NOT CR
                  0061 ;
01C0  F5          0062          PUSH    AF              ; SAVE AF-REG.
01C1  3E0A        0063          LD      A,LF            ; IF CR, SEND LINE FEED ALSO
01C3  CD8201      0064 OUTNUL:  CALL    OUTCHR          ; (THIS IS RECURSIVE)
01C6  3AE103      0065          LD      A,(NULLS)       ; GET NUMBER OF NULLS
01C9  C5          0066          PUSH    BC              ; SAVE BC-REG.
01CA  47          0067          LD      B,A             ; PUT NULL-COUNT INTO B-REG.
01CB  97          0068 OUTC50:  SUB     A               ; CLEAR A-REG. TO A NULL
01CC  05          0069          DEC     B               ; DECREMENT NULL-COUNT
01CD  F48201      0070          CALL    P,OUTCHR        ; OUTPUT NULL IF COUNT NOT= 0
01D0  DACB01      0071          JP      C,OUTC50        ; C-FLAG SET BY OUTCHR IF LAST
                  0072                                  ;   CHARACTER WAS NULL
01D3  C1          0073          POP     BC              ; RESTORE REGISTERS
01D4  F1          0074 ENDOUT:  POP     AF              ; /
01D5  C9          0075          RET                     ; RETURN TO CONTROL BASIC


01D6  DB00        0079 CHKINP:  IN      A,CSTAT         ; *** CHECK FOR INPUT ROUTINE ***
01D8  E640        0080          AND     RDA             ; GET RECEIVER-DATA BIT
01DA  C8          0081          RET     Z               ; RETURN TO CONTROL BASIC IF NOT READY
01DB  DB01        0082          IN      A,CDATA         ; READ CHARACTER FROM CONSOLE
01DD  E67F        0083          AND     PSTRIP          ; STRIP PARITY BIT
01DF  FE0F        0084          CP      CTRLO           ; CHECK FOR CONTROL-O CHARACTER
01E1  C2EE01      0085          JP      NZ,CHKI10       ; IF NOT CTRL-O, DO FURTHER CHECKING
01E4  3AE503      0086          LD      A,(OCSW)        ; CONTROL-O TOGGLES VIS./INVIS. SWITCH
01E7  2F          0087          CPL                     ; TOGGLE BIT-7
01E8  32E503      0088          LD      (OCSW),A        ; STORE VISIBLE/INVISIBLE SWITCH BACK
01EB  C3D601      0089          JP      CHKINP          ; INPUT ANOTHER CHARACTER
                  0090 ;
01EE  FE03        0091 CHKI10:  CP      CTRLC           ; CHECK FOR CONTROL-C CHARACTER
01F0  C0          0092          RET     NZ              ; RETURN TO CONTROL BASIC IF NOT CTRL-C
01F1  C32FE4      0093          JP      RSTART          ; IF CTRL-C, ABORT PROGRAM AND WARM-
                  0094                                  ;   START CONTROL BASIC

      (01F4)      0096 FIRSTFREE EQU    $               ; FIRST FREE BYTE OF MEMORY

Errors            0
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0003
SYMBOL TABLE

CDATA   0001    CHKI10  01EE    CHKINP  01D6    CR      000D    CRLF    0180    CSTAT   0000
CTRLC   0003    CTRLO   000F    ENDOUT  01D4    FIRSTF  01F4    LF      000A    MARGN   03E2
NULLS   03E1    OCMG    03E4    OCSW    03E5    OUTC10  018A    OUTC20  01A2    OUTC30  01B8
OUTC40  01BB    OUTC50  01CB    OUTCHR  0182    OUTNUL  01C3    PSTRIP  007F    RDA     0040
RSTART  E42F    SPC     0020    TBE     0080
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0004
CROSS REFERENCE LISTING

CDATA   0005  0038 0082
CHKI10  0091  0085
CHKINP  0079  0089
CR      0012  0025 0039 0059
CRLF    0025  0056
CSTAT   0004  0005 0031 0079
CTRLC   0010  0091
CTRLO   0013  0084
ENDOUT  0074  0030
FIRSTF  0096
LF      0011  0063
MARGN   0017  0047
NULLS   0016  0065
OCMG    0019  0040
OCSW    0020  0028 0086 0088
OUTC10  0031  0033
OUTC20  0043  0041
OUTC30  0056  0049
OUTC40  0057  0044 0051
OUTC50  0068  0071
OUTCHR  0027  0064 0070
OUTNUL  0064
PSTRIP  0008  0037 0083
RDA     0007  0080
RSTART  0022  0093
SPC     0014  0043 0054
TBE     0006  0032
```

8.2 Control Basic model MCB-216

8.2.1 Memory Allocation (MCB-216)

The listing which follows is an outline of memory allocation for Control Basic model MCB-216 equivalent to that of section 8.1.1. The minimum memory requirements to run this model are RAM from 2000H to 23FFH and the Control Basic Interpreter and Monitor in ROM from 0 to FFFH, the exact configuration of the Cromemco Single Card Computer. Memory from 1000H to 1FFFH is usually ROM (as on the SCC) and may be used to store previously SAVEd programs. Locations 2400H to the end of addressable memory may be used for SAVEd or EPROMed programs and may be either ROM or RAM. Page 2 of the listing is a detailed breakdown of the RAM from 2000H to 21FEH, which contains Control Basic variables, stack, input buffer, and console parameters.

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0001
3K Control Basic Variables and Memory Allocation (MCB-216)

                        0003 ;
                        0004 ;       Copyright (c) 1976, 1979 Cromemco, Inc.
                        0005 ;
                        0006 ;
                        0007 ;  *** Memory Usage (probably RAM unless otherwise noted) ***
                        0008 ;
                        0009 ;  0000-0422 are for SCC Monitor (ROM)
                        0010 ;  0423-0FFF are for SCC Control Basic Interpreter (ROM)
                        0011 ;  1000-1FFF are for SAVEd user programs (usually ROM)
                        0012 ;  2000-21FE are for variables, input line, and stack
                        0013 ;  21FF-23FF are for Control Basic text & array @(I)
                        0014 ;  2400-FFFF are not used but may be used for saved files


(2000)                  0018 BOTRAM  EQU     2000H           ;."BOTTOM OF RAM" FOR VARIABLES, ETC.
(2400)                  0019 TOPRAM  EQU     2400H           ; "TOP OF RAM" USED BY CONTROL BASIC
(0423)                  0020 BOTROM  EQU     423H            ; "BOTTOM OF ROM" CONTAINING CB
                        0021 ; The following are used for interface to the SCC Monitor:
(003B)                  0022 MONITR  EQU     3BH             ; SCC MONITOR WARM-START LOCATION
(005C)                  0023 INITTUART EQU   5CH             ; TUART INITIALIZATION ROUT. LOCATION
```

```
CROMEMCO CDOS Z80 ASSEMBLER version 02.15                    PAGE 0002
3K Control Basic Variables and Memory Allocation (MCB-216)

0000'                   0029            ORG     BOTRAM
2000    (0068)          0030 VARBGN:    DEFS    2*(26*2)        ; VARIABLE A-Z AND A0-Z0 STORAGE
2068    (0001)          0031            DEFS    1               ; EXTRA BYTE FOR INPUT BUFFER
2069    (0085)          0032 BUFFER:    DEFS    133             ; STORAGE FOR INPUT LINE BUFFER
        (20EE)          0033 BUFEND     EQU     $               ; END OF INPUT BUFFER
20EE    (0028)          0034            DEFS    40              ; EXTRA BYTES FOR STACK
2116    (00C4)          0035 STKLMT:    DEFS    200-4           ; SOFT LIMIT FOR STACK
        (21DA)          0036 STACK      EQU     $               ; TOP OF STACK
21DA    (0004)          0037            DEFS    4               ; RESERVED FOR RDOS
21DE    (0001)          0038 BASFLGS:   DEFS    1               ; 0 FOR COLD-; NON-0 FOR WARM-START
21DF    (0002)          0039 IOINS:     DEFS    2               ; STORAGE FOR AN I/O INSTRUCTION
21E1    (0001)          0040 IORET:     DEFS    1               ; STORAGE FOR THE RETURN INSTRUCTION
21E2    (0001)          0041 NULLS:     DEFS    1               ; NUMBER OF NULLS AFTER CR-LF
21E3    (0001)          0042 MARGN:     DEFS    1               ; SOFT TERMINAL WIDTH
21E4    (0001)          0043            DEFS    1               ; HARD TERMINAL WIDTH
21E5    (0001)          0044 OCMG:      DEFS    1               ; COLUMN COUNTER FOR MARGINS
21E6    (0001)          0045 OCSW:      DEFS    1               ; SWITCH FOR VISIBLE OUTPUT
21E7    (0002)          0046 CURRNT:    DEFS    2               ; CURRENT LINE NUMBER STORAGE
21E9    (0002)          0047 STKRUN:    DEFS    2               ; SAVES STACK POINTER DURING 'RUN'
21EB    (0002)          0048 STKGOS:    DEFS    2               ; SAVES STACK POINTER DURING 'GOSUB'
        (21ED)          0049 VARNXT     EQU     $               ; TEMP. STOR. OF CURRENT 'NEXT' VAR.
        (21ED)          0050 CALDE      EQU     $               ; TEMP. STOR. FOR ADDR. FOR 'CALL'
21ED    (0002)          0051 STKINP:    DEFS    2               ; SAVES STACK POINTER DURING 'INPUT'
21EF    (0002)          0052 LOPVAR:    DEFS    2               ; ADDRESS OF 'FOR' LOOP VARIABLE
21F1    (0002)          0053 LOPINC:    DEFS    2               ; 'STEP' SIZE FOR 'FOR' LOOP
21F3    (0002)          0054 LOPLMT:    DEFS    2               ; LIMIT OF 'FOR' LOOP VARIABLE
21F5    (0002)          0055 LOPLN:     DEFS    2               ; TEMP. STOR. OF 'CURRNT' IN 'FOR'
21F7    (0002)          0056 LOPPT:     DEFS    2               ; TEXT POINTER TEMP. STOR. IN 'FOR'
21F9    (0002)          0057 RANPNT:    DEFS    2               ; ADDR. POINTER USED IN RND FUNC.
21FB    (0002)          0058 TXTBGN:    DEFS    2               ; STORAGE OF TEXT POINTER ADDR.
21FD    (0002)          0059 TXTLMT:    DEFS    2               ; TEXT AREA LIMIT (SAME AS 'TOPRAM'
                        0060                                    ;   UNLESS CHANGED BY 'LOCK')


        (21FF)          0064 TXTRAM     EQU     $
21FF    (0002)          0065 TXTUNF:    DEFS    2               ; POINTER TO UNFILLED TEXT AREA
2201    (0002)          0066 TEXT:      DEFS    2               ; START OF ACTUAL TEXT (STORAGE
                        0067                                    ;   OF FIRST LINE NUMBER)
```

Also shown on page 2 of the listing above is the beginning of the text area for the current program. The text area is organized as follows; any stored program (by means of the SAVE or EPROM commands) is organized this same way:

1. The first two bytes of the first page point to the first available byte after the program text relative to 21FFH.

2. The next two bytes contain the line number of the first line in binary (low byte-high byte). This is followed by the text of the first line and a single 0DH, or CR, at the end of that line.

3. Subsequent lines of text duplicate the form of this first line: 2-byte binary line number, ASCII text, CR-byte.

4. The final line of the program is followed by 2 additional bytes, xxH-FFH, where xx is an indeterminate value.

5. Following these are two bytes which contain the value of the array element @(Ø) (low byte-high byte binary). The value stored in TXTUNF (location 21FFH) will be the address of the next byte following this array element. (In the case of a stored program file, this address is stored in the first two bytes of the first page of the file, which is equivalent to the location TXTUNF.)

The foregoing structure of a program is the same regardless of whether it is a stored file or the current program in the text area. Also, this structure is identical to that given in section 8.1.1 for model CB-3Ø8 except that the current program area is relative to 21FFH instead of to 4ØØH. This means that users who want to use programs with MCB-216 which were previously EPROMed or SAVEd under CB-3Ø8 will have to change the ROMs slightly by adding an offset of 1DFFH (or, the difference 21FFH-4ØØH) to the first two bytes of each program file, i.e., the value stored low byte-high byte in "TXTUNF" above.

The storage of the array @(I) is the same for MCB-216 as for CB-3Ø8. This means that for the current program file stored in the text area, subsequent values of the array @(I), where I=1,2,3,..., are stored in the text area following @(Ø) (two bytes per array element). If the array @(I) grows too large to fit in the text area, Control Basic will not allocate memory past the LOCKed top and will instead print "SORRY". Use the LOCK command to enlarge the text area in such a case.

Also, note that for stored program files the first two bytes point to the first available memory location relative to 21FFH. Thus for example, the two bytes 5F-22 (low-high) stored at location 43ØØH (page 43H) mean that the first available memory location following that file is at 4360H (225FH-21FFH+43ØØH). More information on arrays will be found in section 3.2.

8.2.2 I/O Drivers (MCB-216)

The I/O drivers for model MCB-216 of Control Basic are not moved into RAM as they are for model CB-308. This was done to allow more space in the Single Card Computer RAM for user programs. The I/O drivers are, however, largely the same as those used by CB-308 except that relative jump (JR) instructions are used in place of absolute jump (JP) instructions. The character output routine "OUTCHR" begins at location E98H and the character input routine "CHKINP" begins at location EEBH in the MCB-216 ROMs (see CB-308 listing in section 8.1.2).

8.2.3 Interrupts

Special provisions have been made in Control Basic model MCB-216 for use with the Cromemco Single Card Computer (model SCC). Among these is the ability to take advantage of the SCC's interrupt structure. The Single Card Computer has 8 potential sources of interrupt, which are listed in the table below, and which are executed by Mode 0 of the three types of Z80 interrupts. Mode 0 interrupts execute one of the eight Z80 ReSTart instructions, RST 00H through RST 38H. However, since the addresses these instructions use are all part of the MCB-216 ROMs themselves and can thus not be changed by the user, this model of Control Basic has been programmed with JP instructions at each RST to fixed locations in on-board user programmable ROM, summarized (in order of priority from highest to lowest) in the table below:

Source of Interrupt	Vector Address
Timer 1 timeout	not available
Timer 2 timeout	10F2H
External int. req. ($\overline{\text{INT}}$)	10F4H
Timer 3 timeout	10F6H
Rec'r. data available	10F8H
Trans'r. buffer empty	10FAH
Timer 4 timeout	not available
Timer 5 or PI7	10FEH

Note that two of the Restarts (and thus two of the eight interrupts) cannot be used because their addresses are required for other system functions. Address 0 (the location of a vector for RST 00H) is

required for initial power-on and contains a Jump to the cold start of the Control Basic segment of MCB-216 (see section 5.9.1). Address 30H (and the instruction RST 30H) is used as a break-point trap for debugging with the SCC Monitor portion of MCB-216.

Also note that the rest of the interrupt vectors jump to locations in page 10H which are separated by only 2 bytes. It is up to the programmer to place Relative Jump instructions (JR) at these locations in ROM. The interrupt routines themselves must also be user-provided.

The interrupt scheme outlined above was chosen to enable maximum use of the SCC-provided interrupts while placing minimum constraints on the amounts of ROM and RAM required. Note in particular that the top portion of page 10H was chosen for the interrupt vectors, leaving the bottom portion of that page still free for the AUTORUN command, if desired (see section 5.3.4). Thus, for example, the routine of section 5.3.4 could be programmed into a portion of a 2716 (2516) to later be loaded at 1000H (ROM socket 2), and would occupy approximately 30H bytes. This would leave room in the upper portion of page 10H (and, of course, within the same ROM) for the interrupt vectors described above. If the user does not intend to make use of the SCC Interrupts, then the program of section 5.3.4 could have line 30 eliminated completely, and in its place the continuing text of the program which is to be AUTORUN.

8.2.4 Miscellaneous Features of MCB-216

Since MCB-216 is composed of two integrated parts, Control Basic and the SCC Monitor, this section deals briefly with the differences between this Monitor and model ZM-108 supplied by Cromemco in stand-alone form. The MCB-216 Monitor does not include the "U" (UART) or "N" (Null) commands of ZM-108. It does, however, have two additional commands, "B" and "E".

The "B" or Basic command is used to return control from the Monitor to Control Basic. It is described further in section 5.9.1.

The "E" or Examine command is used to examine the contents of input ports. The format to use is simply "E nn <CR>", where nn is a port number in hex ranging from Ø to FF. The current value present at that input port will be read and displayed in hex on the console terminal.

I N D E X