Talk Together

Just bought an MC-10? Don't have any software for it? Oh, yes you do!

by William Barden, Jr.

LET'S FACE IT — like a lot of you out there, I'm a computer freak. Sure, I put good promotional copy on my book jackets — "Avocations are sailing, mathematical games, ham radio, Eastern European folk dancing, alligator wrestling, and driving Grand Prix race cars" — but, in fact, you'll see me in the wee hours hacking away on the Color Computer and other equipment. Lately, though, things have gotten somewhat boring. Even sophisticated Radio Shack equipment gets stale.

It was one of those boring mornings not long ago, when I forsook the computer room for brush-up steps on a Polish mazurka. I had just brought out my peasant costume when I happened to glance at the paper. "New for '83," the ad said. The Radio Shack MC-10 computer system!

Several hours later I was the apprehensive owner of an MC-10. Apprehensive, primarily because of its size. Shades of the Timex/Sinclair TS1000! Can this computer really *do* anything?

Inside the MC-10

An hour later I had hooked up the system, run some short benchmarks, and gotten my first impressions — the Basic isn't bad — not as powerful as the Color Computer, but not bad. The graphics were not too impressive — 64 by 32 elements in color is a far cry from the Color Computer. And no assembly language hooks such as USR and VARPTR!

Eight hours later I had revised my impressions. Shards of tin lay around me — I had cut off the shielding over the printed circuit board inside the MC-10. The wastebasket was stuffed with partial sketches and pieces of notepad paper. Bleary-eyed, I summarized what I had found:

- Item: The MC-10 uses a 6803 microprocessor. It's better than the older 6800, although not as powerful as the Color Computer's 6809. Believe it or not, it has a hardware multiply instruction and some other goodies.
- Item: The 8K ROM Basic is mounted on a socket. The socket has enough pins that a larger ROM can be installed!
- Item: The graphics are controlled by a 6847 Video Display Generator chip, the same chip that's in the Color Computer! A high-resolution 256 by 192 color graphics is possible with this chip, but not implemented in the Basic.
- Item: There are more Basic commands in the MC-10 than the manual describes. Among the existing commands are USR for machine-language linkage, EXEC for transferring control to machine-language subroutines, CLOADM for loading machine-language programs, and VARPTR for finding the location of variables.
- Item: The cassette recording technique appeared to be compatible with that used on the Color Computer!

Color Computer and MC-10 Cassette Compatibility

The last item interested me the most. Would it be possible to transfer programs from the MC-10 to the Color Computer and vice versa? It seemed plausible.

The first step was to try some tapes. I cut an MC-10 tape and loaded it into the Color Computer. There it was! It loaded fine. The Basic line numbers were correct, but the Basic commands were not as expected. (Why? We'll get to that in a second...) Going the other way around, cassette files from the Color Computer loaded into the MC-10. Again the line numbers and numbers of lines were all right, but the Basic commands were different.

Up to this point I had verified that the recording formats of the MC-10 and Color Computer were the same. Here's how each works: The Color Computer uses a digital-to-analog converter to produce a sine wave output of either 1200 or 2400 cycles per second (hertz), as shown in Figure 1. The MC-10 doesn't

Continued from page 49

use a "dac," but its output is a square wave of the same frequency. As a matter of fact, the Model III/IV in high-speed cassette mode also uses the same recording technique as the MC-10. The cassette tape file formats were also the same — the same header data for the file name, and so forth.

Now I knew I could eventually transfer files between the MC-10 and the Color Computer, using the cassette recorder as an intermediary device. This scheme is not too elegant compared to transferring data over RS-232C lines, but it would certainly be handy to use the editing, renumbering, and other features of the Color Computer on MC-10 Basic programs. And, as stoic as my constitution is, my cramped fingers could benefit from the larger keyboard of the Color Computer.

Tokenism

For those of you who are uninitiated into the mysteries of Microsoft Basic architecture, let me explain how I knew I could eventually get that supercilious Color Computer talking to his baby brother...

All Microsoft Basics are similar. Microsoft presumably cranks them out on an assembly-line basis in Bellevue on a Digital Equipment VAX system using a combination of automatic and manual methods. It's efficient to use the same structure, and certainly cost effective as far as development time. The structure of Microsoft Basic program lines is shown in Figure 2.

All Basic program lines are stored in a contiguous area in memory. This really means that the second line follows the first immediately, the third follows the second, and so forth. Although there could be gaps between the lines, there usually aren't. The first byte of the next line starts one byte after the last byte of the previous line.

Each Basic line has three parts. The first part is a header. This header consists of four bytes. The first two bytes are the starting address of the *next* line in binary. The last line of the program has a next line pointer of 0, to signify the end of the program. The next two bytes are the line number in binary.

The third part of the line is a terminating 0 byte. It defines the end of the line.

The second part of the line is the text of the line itself. For example, if you have the line:

```
100 FOR I = 1 TO 1000
```

Please turn to p. 52

Figure 1. Color Computer vs. MC-10 Cassette Wave forms

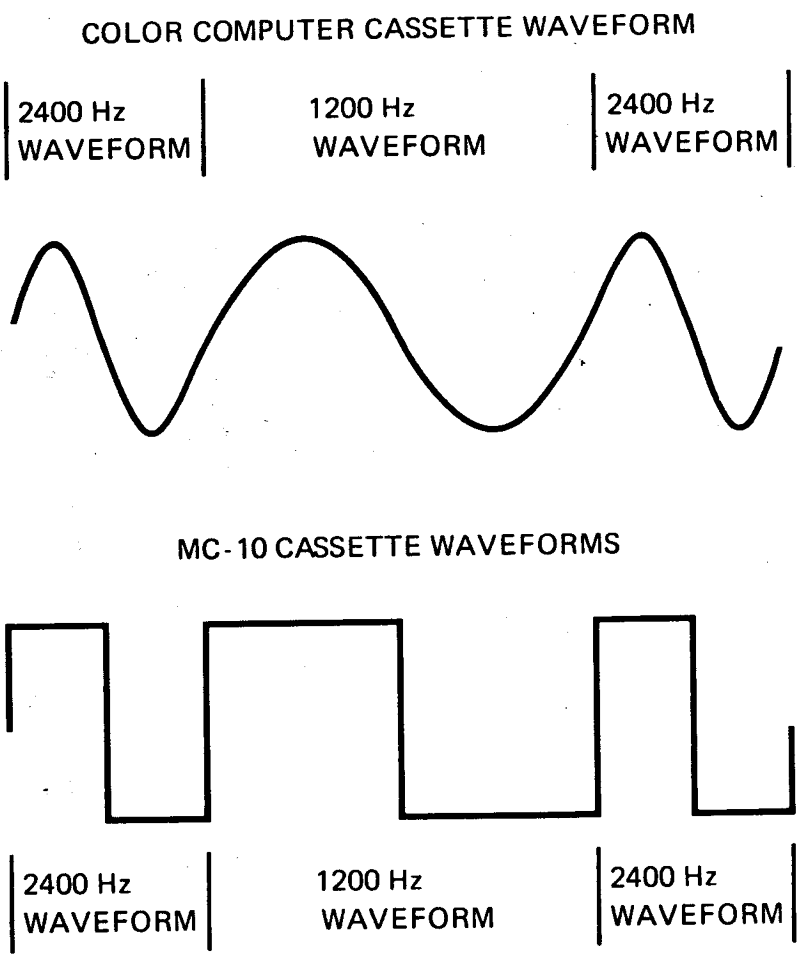
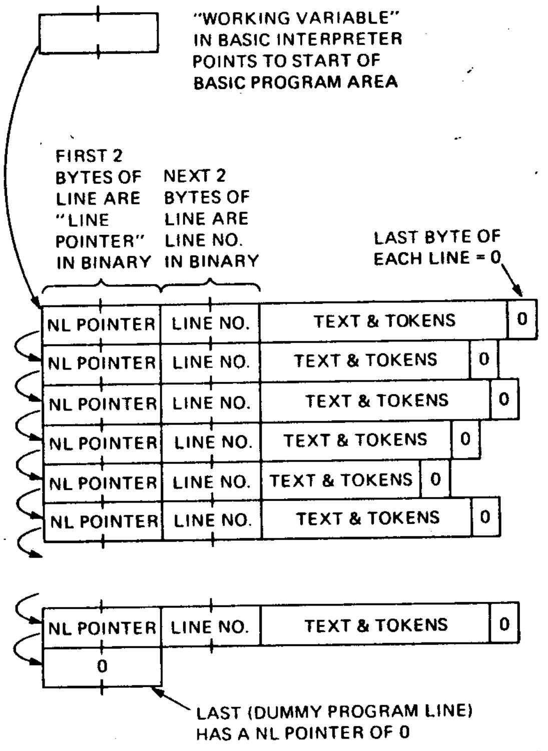


Figure 2. Microsoft Basic Program Line Structure


"WORKING VARIABLE" IN BASIC INTERPRETER POINTS TO START OF BASIC PROGRAM AREA
FIRST 2 BYTES OF LINE ARE "LINE POINTER" IN BINARY
NEXT 2 BYTES OF LINE ARE LINE NO. IN BINARY
LAST BYTE OF EACH LINE = 0
NL POINTER
LINE NO.
TEXT & TOKENS
0
LAST (DUMMY PROGRAM LINE) HAS A NL POINTER OF 0

Continued from p. 50

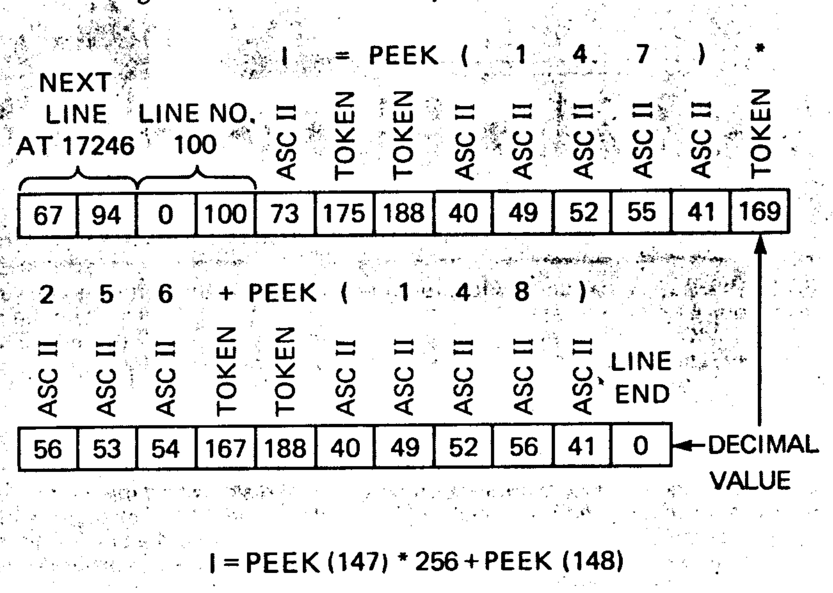
the "FOR I=1 TO 1000" would be in the second, or central, part of the line.

If you were a software house designing Basic interpreters, how could you make memory storage more efficient? One way is by "compressing" data. If you had text that frequently used 128 words, ranging from "aardvark" to "zebra," you might consider assigning a special code to each of the 128 words, say 128 through 255. That way, the eight letters of aardvark could be compressed into a single byte of, say, 153. You would have saved seven characters (seven bytes) each time "aardvark" was used.

So it is with Basic commands. There are only so many Basic commands, and each is assigned a special code, called a "token." In the Color Computer, for example, the token for the long Basic command RESTORE is a code of 143, saving six characters, or bytes.

If you look at a Basic program in memory, you'll find that each Basic command uses a token to save memory storage. You'll see the normal text interspersed with Basic tokens, as shown in Figure 3.

Figure 3. Normal Text Interspersed with Basic Tokens



One important point about the tokens, by the way: tokens are almost always values in the range of 128 through 255. The reason for this is that values in the range of 0 through 127 represent ASCII characters — normal printable or displayable characters of 0 – 9, A – Z, a – z, and special characters such as # or %.

The main problem, then, is to decode those Basic tokens in both the Color Computer and MC-10 so we can translate from one to another.

Disassembling Basic

How do you compile a list of tokens on a Microsoft Basic computer? One obvious way is to put in all the possible combinations of Basic commands, find the area in RAM where the Basic program is stored, and then display the tokenized version of the line, compiling a list of Basic tokens by manual methods. For example, suppose you started with a NEW command and then entered the Basic lines:

```
100  REM $#
110  FOR I=0 TO 65535
120  A$ = CHR$(PEEK(I)) + CHR$(PEEK(I+ 1))
130  IF A$ = "$#" THEN PRINT I
140  NEXT I
```

This short Basic routine searches all the memory space, ROM and RAM, for the occurrence of the two unique characters $#. Every time that particular combination is encountered, the address at which they are found will be printed out. It's a way of locating the start of the Basic program area.

This program might produce two or three locations at which this particular character string was found, but you could then further define the area by trying additional unique characters. Knowing where the character string was, you can work back one location to find the start of the first Basic line, the REM token, which you could define by a PEEK. From that point it's easy to substitute other Basic commands and compile a list.

This manual method is pretty tedious. I used a much nicer technique to find the tokens for the MC-10, one that I had used on the Color Computer and other Radio Shack systems before. The day was flying by, but I had to uncover the secrets of the MC-10. I executed this program:

```
100  FOR I=0 TO 65535
110  A=PEEK(I): IF (A<32) OR (A>127) THEN 130
120  PRINT CHR$(PEEK(I));
130  IF I-INT(I/256)*256=0 THEN PRINT I
140  NEXT I
```

This short piece of code will spew out all ASCII characters in memory on the screen, interspersed with every multiple of 256 memory locations. I ran it and saw a display of characters that looked like this:

```
FOGOTOGOSUREIDATPR
```

This represented the Token Table found in every Microsoft Basic. It's generally a sequential listing of all tokens to be found in the computer system, even undocumented tokens! In some cases this will be one large table. In other cases, such as the Color Computer, there may be two or more segments of the symbol table.

One unique feature of the Token Table: the first or the last character of every Basic keyword is set to an ASCII character plus 128. This is done to delimit or denote the end of each keyword, compacting the table. That's why the delimiting character will not print in the code above.

In the case of the MC-10, the Token Table (in my version) is located at ROM location 57413 ($E045 where $ stands for hexadecimal). (This area is also addressable by using addresses 8192 bytes lower in the $C045 area.) It's one large block of tokens, as shown in Table 1. You'll note that there are several undocumented tokens, such as USR and EXEC, that hold the promise of big plans for the small MC-10.

Table 1 not only shows the MC-10 tokens, but also lists the corresponding Color Computer tokens. Note that some of the Color Computer tokens are two values (two bytes) long. The Color Computer has a lot of tokens, and it was necessary to extend the token representation into two-byte values simply because the range of 128 through 255 couldn't hold them all. In the case of two-byte tokens, the Basic keyword is compressed into a two-byte value, the first of which is a value of 255 ($FF). The Color Computer tokens start at 43622 ($AA66) in my system and are shown in tabular form in Table 2, along with the corresponding MC-10 tokens.

Please turn to p. 54

Continued from p. 52

Cross-Referencing Tokens

At this point I had forgotten completely about my folk dancing. Dusk was settling in as I wrote down the token values. Only my allegiance to Radio Shack computer systems kept me going.

The next thing I did was to compile a cross-reference of Basic keywords and the token values for both the MC-10 and the Color Computer. This is shown in Table 3. I was glad I did this, for I detected several problems.

First, there are many Color Computer Basic commands that are not (yet) found on the MC-10. These would have to be detected in any translation program, of course.

There are also certain MC-10 tokens that have to translate into two-byte tokens. This poses certain problems. The proper way of doing this would be to expand the MC-10 program line by one byte to fit in the Color Computer token when going from MC-10 code to Color Computer code, and shrink the Color Computer program line by one byte when going from Color Computer code to MC-10 code. I'll show you how I circumvented this, shortly.

The next problem is one of GOSUBs and GOTOs. In the Color Computer a GOTO is represented by a token of 129 for GO, and a token of 165 for TO. In the MC-10, there is a single token for GOTO, a 129. In the Color Computer a GOSUB is represented by GO and SUB, for two tokens of 129 and 166. In the MC-10, GOSUB is a single token of 130. Here again, the problem is fitting two bytes into a one-byte area and vice versa in the translation.

After a hefty bite of kielbasa, left over from the beginning of the mazurka, I was ready to tackle the conversion. The results of my marathon all-night session are shown in Program Listings 1 and 2. Let's look at the Color Computer version first (Listing 1).

The Data statements in Program Listing 1 represent a duplicate of the data in Table 2. Rather than listing the Color Computer tokens, however, there are two tables, one for one-byte Color Computer tokens, and one for two-byte Color Computer tokens.

The one-byte tokens are stored in a 128-byte array called T1. The first entry in this array corresponds to Color Computer token 128, the next for token 129, and so forth. Each entry holds the corresponding MC-10 token, or a 0 value for "no MC-10 token."

The two-byte tokens are stored in a 128-byte array called T2. The first entry

Please turn to page 59

Table 1. MC-10 Basic Tokens

TOKEN	VALUE	COLOR COMPUTER VALUE	TOKEN	VALUE	COLOR COMPUTER VALUE
FOR	128	128	STEP	165	169
GOTO	129	—	OFF	166	170
GOSUB	130	—	+	167	171
REM	131	130	–	168	172
IF	132	133	*	169	173
DATA	133	134	/	170	174
PRINT	134	135	CARET	171	175
ON	135	136	AND	172	176
INPUT	136	137	OR	173	177
END	137	138	>	174	178
NEXT	138	139	=	175	179
DIM	139	140	<	176	180
READ	140	141	SGN	177	255,128
LET	141	186	INT	178	255,129
RUN	142	142	ABS	179	255,130
RESTORE	143	143	USR	180	255,131
RETURN	144	144	RND	181	255,132
STOP	145	145	SQR	182	255,155
POKE	146	146	LOG	183	255,153
CONT	147	147	EXP	184	255,151
LIST	148	148	SIN	185	255,133
CLEAR	149	149	COS	186	255,149
NEW	150	150	TAN	187	255,150
CLOAD	151	151	PEEK	188	255,134
CSAVE	152	152	LEN	189	255,135
LLIST	153	155	STR$	190	255,136
LPRINT	154	—	VAL	191	255,137
SET	155	156	ASC	192	255,138
RESET	156	157	CHR$	193	255,139
CLS	157	158	LEFT$	194	255,142
SOUND	158	160	RIGHT$	195	255,143
EXEC	159	162	MID$	196	255,144
SKIPF	160	163	POINT	197	255,145
TAB(	161	164	VARPTR	198	255,157
TO	162	165	INKEY$	199	255,146
THEN	163	167	MEM	200	255,147
NOT	164	168			

■■■

Table 2. Color Computer Tokens

TOKEN	VALUE	MC-10 VALUE	TOKEN	VALUE	MC-10 VALUE
FOR	128	128	RUN	142	142
GO	129	—	RESTORE	143	143
REM	130	131	RETURN	144	144
'	131	—	STOP	145	145
ELSE	132	—	POKE	146	146
IF	133	132	CONT	147	147
DATA	134	133	LIST	148	148
PRINT	135	134	CLEAR	149	149
ON	136	135	NEW	150	150
INPUT	137	136	CLOAD	151	151
END	138	137	CSAVE	152	152
NEXT	139	138	OPEN	153	—
DIM	140	139	CLOSE	154	—
READ	141	140	LLIST	155	153

Please turn to p. 58

Table 2 continued from p. 54

TOKEN	VALUE	MC-10 VALUE	TOKEN	VALUE	MC-10 VALUE	TOKEN	VALUE	MC-10 VALUE
SET	156	155	DEF	185	—	LEN	255,135	189
RESET	157	156	LET	186	141	STR$	255,136	190
CLS	158	157	LINE	187	—	VAL	255,137	191
MOTOR	159	—	PCLS	188	—	ASC	255,138	192
SOUND	160	158	PSET	189	—	CHR$	255,139	193
AUDIO	161	—	PRESET	190	—	EOF	255,140	—
EXEC	162	159	SCREEN	191	—	JOYSTK	255,141	—
SKIPF	163	160	PCLEAR	192	—	LEFT$	255,142	194
TAB(	164	161	COLOR	193	—	RIGHT$	255,143	195
TO	165	162	CIRCLE	194	—	MID$	255,144	196
SUB	166	—	PAINT	195	—	POINT	255,145	197
THEN	167	163	GET	196	—	INKEY$	255,146	199
NOT	168	164	PUT	197	—	MEM	255,147	200
STEP	169	165	DRAW	198	—	ATN	255,148	—
OFF	170	166	PCOPY	199	—	COS	255,149	186
+	171	167	PMODE	200	—	TAN	255,150	187
–	172	168	PLAY	201	—	EXP	255,151	184
*	173	169	DLOAD	202	—	FIX	255,152	—
/	174	170	RENUM	203	—	LOG	255,153	183
CARET	175	171	FN	204	—	POS	255,154	—
AND	176	172	USING	205	—	SQR	255,155	182
OR	177	173	* * * * *	* * * * *	* * * * *	HEX$	255,156	—
>	178	174	SGN	255,128	177	VARPTR	255,157	198
=	179	175	INT	255,129	178	INSTR	255,158	—
<	180	176	ABS	255,130	179	TIMER	255,159	—
DEL	181	—	USR	255,131	180	PPOINT	255,160	—
EDIT	182	—	RND	255,132	181	STRING$	255,161	—
TRON	183	—	SIN	255,133	185			
TROFF	184	—	PEEK	255,134	188			

[illegible]

[illegible]

Continued from p. 54

in this array corresponds to Color Computer token 255, 128, the next for 255, 129, and so forth. Each entry holds the corresponding MC-10 token, or a zero value for no MC-10 token.

Assume that the conversion program is at lines 1 to 99 and that the Color Computer program to be converted to MC-10 format is somewhere above in Lines 100 through 59999. The first action is to move the Data values into the two arrays, T1 and T2.

Now the starting address of the Basic program area is put into variable PL. PL always points to the next byte of the program area lines to be processed. The starting address of the Basic program area can be found in RAM location &H19 ($19 or decimal 25). Variable NL is set equal to PL. NL holds the next line pointer value from the current Basic program line.

The Process One Line subroutine processes one line of the Basic program. For each line, these actions occur:

- The next line pointer value is read from the first two bytes of the line and put into variable NL.
- The line number of the line is read from the next two bytes of the line and put into variable LN.

Please turn page

Table 3. Cross Reference Listing, Color Computer and MC-10 Tokens

TOKEN	COLOR COMPUTER	MC-10
[illegible]	131	—
+	171	167
−	172	168
*	173	169
/	174	170
CARET	175	171
<	180	176
=	179	175
>	178	174
ABS	255,130	179
AND	176	172
ASC	255,138	192
ATN	255,148	—
AUDIO	161	—
CHR$	255,139	193
CIRCLE	194	—
CLEAR	149	149
CLOAD	151	151
CLOSE	154	—
CLS	158	157
COLOR	193	—
CONT	147	147
COS	255,149	186
CSAVE	152	152
DATA	134	133
DEF	185	—
DEL	181	—
DIM	140	139
DLOAD	202	—
DRAW	198	—
EDIT	182	—
ELSE	132	—
END	138	137
EOF	255,140	—
EXEC	162	159
EXP	255,151	184
FIX	255,152	—
FN	204	—
FOR	128	128
GET	196	—
GO	129	—
GOSUB	—	130
GOTO	—	129
HEX$	255,156	—
IF	133	132
INKEY$	255,146	199
INPUT	137	136
INSTR	255,158	—
INT	255,129	178
JOYSTK	255,141	—
LEFT$	255,142	194
LEN	255,135	189
LET	186	141

Please turn the page

FOR QUALITY EDUCATIONAL SOFTWARE LOOK FOR THIS EMBLEM

SOFTWARE AVAILABLE FOR:
- ▲ TRS-80® COLOR COMPUTER
- ▲ ATARI® 400/800/1200
- ▲ COMMODORE 64®
- ▲ APPLE® &
- ▲ VIC 20®

A T C E PROGRAM

Send for Free Catalog

DEALER INQUIRES INVITED

T.C.E.

P.O. Box 2477 Gaithersburg, Maryland 20879 (301) 963-3848

Circle No. 33 on Reader Service Card

Continued from page 59

- If the next line pointer is 0, the last program line has been processed.
- If the line number of the current line is under 100, a line of the conversion routine itself is being processed and no conversion is done.
- The next byte of the line is tested.
- If this byte is a zero, the end of the line has been reached and the subroutine returns back to the main code.
- If this byte is less than 128, it is an ASCII character and not a token. In this case PL is bumped by one to point to the next byte of the line. A special case of a literal string ("THIS IS A LITERAL STRING") is detected also. All data within the string is ignored. (This is most important in the MC-10 version of the program, in which graphics characters can be embedded in strings; graphics characters are greater than 128 and would look like tokens if they were processed.)
- If this byte is not 255 or 129 (GO), the corresponding MC-10 token is found from array T1 and POKEd into the current byte, unless the corresponding token is a zero. If the MC-10 token is a zero, an error message of "NO CORRESPONDING MC-10 TOKEN" is displayed and the program stops.
- If the byte was a 255, an ASCII blank is POKEd into the token byte. The next value of the two-byte token is then picked up, and the corresponding MC-10 token is picked up from array T2 and POKEd into that byte. POKEing a blank character causes no problem, as Basic program lines can have extraneous blanks anywhere except in literal string values.
- If the byte was a 129 (GO), the action is similar to a two-byte Color Computer token — a blank is stored for the GO, the next token value is picked up, and the GOSUB or GOTO token for the MC-10 is stored in the second byte.

The conversion continues a line at a time until a next line pointer of zero (end of program) is found. For each line processed, a period is displayed.

To run the conversion program, the program to be converted must start at Line 100 or above. Simply Run it and all program lines will be converted. After the conversion you'll be able to list the converted lines, but they will have MC-10 tokens — garbage as far as the Color Computer is concerned. Delete all lines under 100 and then CSAVE the converted program to cassette. The result is a program in MC-10 format, loadable by

Listing continued from page 59

TOKEN	COLOR COMPUTER	MC-10
LINE	187	—
LIST	148	148
LLIST	155	153
LOG	255,153	183
LPRINT	—	154
MEM	255,147	200
MID$	255,144	196
MOTOR	159	—
NEW	150	150
NEXT	139	138
NOT	168	164
OFF	170	166
ON	136	135
OPEN	153	—
OR	177	173
PAINT	195	—
PCLEAR	192	—
PCLS	188	—
PCOPY	199	—
PEEK	255,134	188
PLAY	201	—
PMODE	200	—
POINT	255,145	197
POKE	146	146
POS	255,154	—
PPOINT	255,160	—
PRESET	190	—
PRINT	135	134
PSET	189	—
PUT	197	—
READ	141	140
REM	130	131
RENUM	203	—
RESET	157	156
RESTORE	143	143
RETURN	144	144
RIGHT$	255,143	195
RND	255,132	181
RUN	142	142
SCREEN	191	—
SET	156	155
SGN	255,128	177
SIN	255,133	185
SKIPF	163	160
SOUND	160	158
SQR	255,155	182
STEP	169	165
STOP	145	145
STR$	255,136	190
STRING$	255,161	—
SUB	166	—
TAB(	164	161
TAN	255,150	187
THEN	167	163
TIMER	255,159	—
TO	165	162
TROFF	184	—
TRON	183	—
USING	205	—
USR	255,131	180
VAL	255,137	191
VARPTR	255,157	198

Program Listing 1. Color Computer to MC-10 Program Converter

```
1 'COLOR TO MC10 CONVERTER
2 REM ONE-BYTE DATA
3 DATA 128,0,131,0,0,132,133,134
,135,136
4 DATA 137,138,139,140,142,143,1
44,145,146,147
5 DATA 148,149,150,151,152,0,0,1
53,155,156
6 DATA 157,0,158,0,159,160,161,1
62,0,163
7 DATA 164,165,166,167,168,169,1
70,171,172,173
8 DATA174,175,176,0,0,0,0,0,141
9 REM TWO-BYTE DATA
10 DATA 177,178,179,180,181,185,
188,189,190,191
11 DATA 192,193,0,0,194,195,196,
197,199,200
12 DATA 0,186,187,184,0,183,0,18
2,0,198
13 CLS: PRINT "CONVERTING"
14 DIM T1(127),T2(127)
15 REM INITIALIZE TOKEN ARRAYS
16 FOR I=0TO 58:READ T1(I):NEXT
```

Listing continued on page 62

the MC-10. Important note: If you do not delete the lines under 100, the MC-10 will go screaming bonkers attempting to list the invalid tokens of the Color Computer program!

Program Listing 2 shows the antithesis of this conversion, a program to convert an MC-10 program to Color Computer format. It operates similarly.

Here we have only one array, T1. The first entry holds the corresponding Color Computer token value for MC-10 value 128, the next the Color Computer token value for MC-10 value 129, and so forth. The negative values represent Color Computer tokens which are two-byte tokens.

Initialization and conversion proceed pretty much the same as in the Color Computer program. The exceptions are negative values and GOSUBs and GOTOs.

If the byte in a line is found to be a token and the token value for the Color Computer is a negative value, then a test is made for a blank preceding the MC-10 token. There *must* be a blank preceding the token value to convert, otherwise a "NO SPACE BEFORE TOKEN" message is displayed and the program stops. If there is a preceding blank, a 255 is stored, followed by the Color Computer token.

A similar situation exists for the GOSUB and GOTO. If the MC-10 value is a 129 or 130, a 129 (Color Computer GO) is stored in the preceding blank, followed by a TO or SUB value. Here again, there *must* be a preceding blank before the MC-10 GOSUB or GOTO, otherwise an error message is displayed and the program stops.

To run the MC-10 conversion, the program to be converted must start at or above Line 100. Run it, and you'll see periods displayed as each line is converted. At the end of the conversion, delete all lines under 100 and CSAVE to cassette. The result is a cassette file that is loadable by the Color Computer and is in Color Computer token format.

There is one thing that the MC-10 to Color Computer conversion program won't do; any keyboard-generated MC-10 graphics character will not be properly converted unless within a string literal. Normally, though, such graphics characters would be within a string literal, as in `100 A$ = "XXXX"`, where X is a non-printable graphics character.

By the time I had finished the two programs, the morning paper had been delivered and my wife was fixing coffee. Here I was 20 hours later with workable programs that made the MC-10 a little more usable. Unfortunately, however, the guitar player had left. Ah well, I could always provide my own accompaniment. Let's see, where had I put that peasant costume... ■ ■ ■

Circle No. 34 on Reader Service Card

SEND $2.00 FOR OUR 25 PAGE CATALOGUE **Refunded with first order**

THE DATAMAN

Box 431, Sta. B
Hamilton, Ontario
Canada L8L 7W2
1-416-529-1319

ALL PRICES IN CANADIAN DOLLARS

KEYBOARD OVERLAYS
Many programs are supplied with keyboard overlays to help you keep track of the various commands used by the program. Now you can add overlays to your own programs or to commercial programs that did not come with this feature. Die cut to fit the standard Color Computer keyboard. Dealer inquiries for blank or custom printed overlays are invited.
CAT. NO. HW002 99¢ each

COCOCOPY
This all M/L Program will copy BASIC or M/L programs including most Auto Start Programs. It will supply the beginning, ending and offset addresses and allow you to change the load address for M/L programs. I/O errors are ignored so that bad tapes can be corrected. Programs can be renamed and the motor/audio functions are controlled from the keyboard.
CAT. NO. DM004 16K Ext $12.95

BOOKS from
SYBEX ✲ BYTE ✲ OSBORNE ✲ RESTON

ADD 3% SHIPPING — MINIMUM 2^{50}

Attention Subscribers!

All correspondence concerning your subscription, i.e., changes of address, duplicate copies, etc. should be sent to Color Computer, PO Box 468, Hasbrouck Heights, NJ 07604. Please include your mailing label for speedy service. BACK ISSUES of Color Computer can be ordered by writing to Color Computer, Att: G. Purnell, Highland Mill, Camden, ME 04843. The price is $5 per copy and all are available except the April 1983 issue.

INTRODUCING

HI-RES SCREEN UTILITY
Featuring: Double Height Characters
On Screen UNDERLINING
Bell Character tone generator
Switchable Full Screen Reverse Video
True Upper & Lower case character set
Reverse Video [illegible]
Programable line lengths from 28 to 255 characters

28 Characters per line
32 Characters per line
36 Characters per line
42 Characters per line
51 Characters per line
64 Characters per line

Line lengths of 85,128 & 255 are unreadable
but; can be very useful for seeing display layouts

All functions are easily programable thru BASIC
Fully BASIC COMPATIBLE including CLS & PRINT @.

★ ★

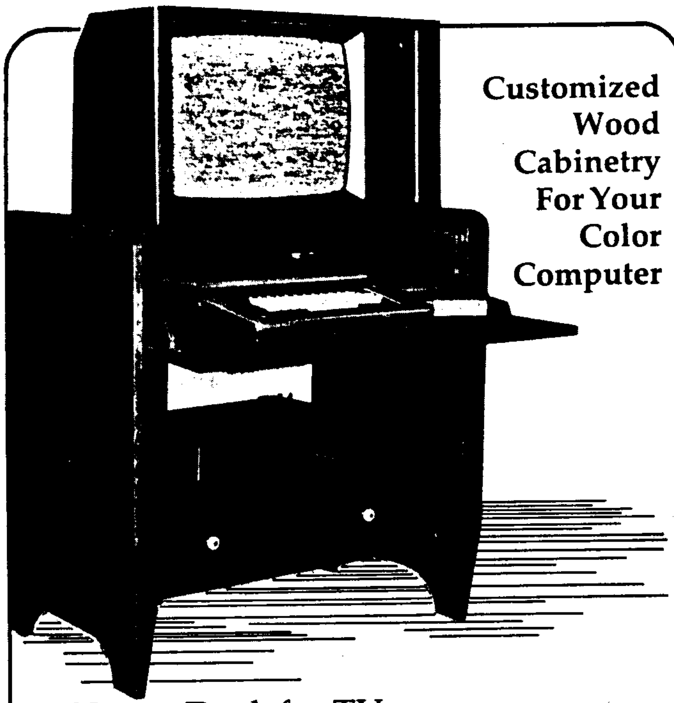
CER COMP

SPECIAL
$19.95

(702)452-0632
5566 Ricochet Avenue
Las Vegas, Nevada 89110

All Orders Shipped
From Stock
Add $2.50 Postage

Circle No. 35 on Reader Service Card



Customized Wood Cabinetry For Your Color Computer

- Upper Deck for TV
- Pull-Out Shelf for Keyboard and Cassette or Video Game
- Lower Deck for Printer
- Storage Compartment and Paper Tray

DIMENSIONS: Height 34″
Width 31½″
Depth 20″

Made of beautiful oak wood, the TV-Computer Combo will be attractive in any room in your home.

WOXALL WOODCRAFT
2nd & Walnut St., Green Lane, PA 18054
215-584-0522

Please send me ____ TV-Computer Combos. $248.00 plus UPS and handling. Less 10% unfinished.
PA residents please include 6% sales tax.
Amount enclosed $ ________
SHIPPED KNOCKED, MINOR ASSEMBLY REQUIRED
☐ Dark Finish ☐ Unfinished

Name ____________________

Address ____________________

____

Signature ____________________

MasterCard VISA Phone ____________________
Acct # ____________ Exp. ________

Woxall Woodcraft, 2nd & Walnut St., Green Lane, PA

Circle No. 36 on Reader Service Card

```
17 FOR I=0TO 29:READ T2(I):NEXT
18 PL=PEEK(&H19)*256+PEEK(&H1A):
 NL=PL
19 GOSUB 22: PRINT ".";: IF NL<>
0 THEN 19
20 PRINT "SUCCESSFUL TOKEN TRANS
LATION": STOP
21 ' SUBROUTINE: PROCESS 1 LINE
22 IF PL<>NL THEN A$="CATASTROPH
IC ERROR": GOTO 47
23 NL=PEEK(PL)*256+PEEK(PL+1): P
L=PL+2
24 LN=PEEK(PL)*256+PEEK(PL+1): P
L=PL+2
25 IF NL=0 THEN RETURN
26 IF LN<100 THEN PL=NL: RETURN
27 REM TRANSLATE TOKENS
28 ST=0
29 TN=PEEK(PL): PL=PL+1: IF TN=0
 THEN RETURN
30 IF TN=34 THEN ST=(ST+1)AND 1
31 IF ST=1 THEN 29
32 IF TN<128 THEN 29
33 IF TN=255 THEN 38
34 IF TN=129 THEN 42
35 TN=T1(TN-128): IF TN=0 THEN A
$="NO CORRESPONDING MC10 TOKEN":
 GOTO 47
36 POKE PL-1,TN: GOTO 29
37 REM TWO BYTE TOKEN HERE
38 POKE PL-1,32: TN=PEEK(PL): PL
=PL+1
39 TN=T2(TN-128): IF TN=0 THEN A
$="NO CORRESPONDING MC10 TOKEN":
 GOTO 47
40 GOTO 36
41 REM GOTO OR GOSUB HERE
42 POKE PL-1,32: TN=PEEK(PL): PL
=PL+1
43 IF TN=165 THEN POKE PL-1,129:
 GOTO 29
44 IF TN=166 THEN POKE PL-1,130:
 GOTO 29
45 A$="INVALID TOKEN AFTER GO":
GOTO 47
46 REM ERROR HERE
47 PRINT: PRINT A$
48 PRINT "LINE NUMBER ";LN
49 STOP
```

Program Listing 2.
MC-10 to Color Computer Program Converter

```
1 'MC10 TO COLOR CONVERTER
2 REM ONE BYTE DATA
3 DATA 128,165,166,130,133,134,1
35,136,137,138
4 DATA 139,140,141,186,142,143
```

```
5 DATA 144,145,146,147,148,149,1
50,151,152,155
6 DATA 0,156,157,158,160,162,163
,164,165,167
7 DATA 168,169,170,171,172,173,1
74,175,176,177
8 DATA 178,179,180
9 REM TWO-BYTE DATA
10 DATA -128,-129,-130,-131,-132
,-155,-153,-151,-133,-149
11 DATA -150,-134,-135,-136,-137
,-138,-139,-142,-143,-144
12 DATA -145,-157,-146,-147
13 CLS: PRINT "CONVERTING"
14 DIM T1(127)
15 RE INITIALIZE TOKEN ARRAY
16 FOR I=0TO 72:READ T1(I):NEXT
17 PL= PEEK(147)*256+ PEEK(148):
 NL=PL
18 GOSUB 21: PRINT ".";: IF NL<>
0 THEN 18
19 PRINT "SUCCESSFUL TOKEN TRANS
LATION": STOP
20 'PROCESS ONE LINE SUBROUTINE
21 IF PL<>NL THEN A$="CATASTROPH
IC ERROR": GOTO 44
22 NL= PEEK(PL)*256+ PEEK(PL+1):
 PL=PL+2
23 LN= PEEK(PL)*256+ PEEK(PL+1):
 PL=PL+2
24 IF NL=0 THEN RETURN
25 IF LN<100 THEN PL=NL: RETURN
26 REM TRANSLATE TOKENS
27 ST=0
28 TN= PEEK(PL): PL=PL+1: IF TN=
0 THEN RETURN
29 IF TN=34 THEN ST=(ST+1)AND 1
30 IF ST=1 THEN 28
31 IF TN<128 THEN 28
32 IF T1(TN-128)<0 THEN 36
33 IF TN=129 OR TN=130 THEN 39
34 POKE PL-1,T1(TN-128):GOTO 28
35 REM TWO-BYTE TOKEN HERE
36 IF PEEK(PL-2)<>32 THEN A$="NO
 SPACE BEFORE TOKEN": GOTO 44
37 POKE PL-2,255: POKE PL-1,-T1(
TN-128): GOTO 28
38 REM GOTO OR GOSUB HERE
39 IF PEEK(PL)<>32 THEN A$="NO S
PACE AFTER GOSUB  OR GOTO": GOTO
 44
40 POKE PL-1,129
41 POKE PL,T1(TN-128)
42 PL=PL+1: GOTO 28
43 REM ERROR HERE
44 PRINT: PRINT A$
45 PRINT "LINE NUMBER ";LN
46 STOP
```

INVEST IN YOUR CHILD

ADDITION THRILLS and SUBTRACTION ACTION

A colorful one and two digit addition and subtraction game. Up to four players. Cumulative scores. The better the players the harder the problems. Mistakes corrected. Educationally sound.

Grades 1-4   16K EXB $14.95

QUESTION and ANSWER EXCELLENT

Give your child a headstart in equalities, inequalities and equation theory. A five program developmental tutorial. Final program incorporates all concepts.

Grades K-3   16K EXB $17.95

INVE$T IN YOURSELF

THE PROGRAMERS TOOLKIT

BASIC AIDS
Superior full screen editing
Debugging made easier
Scroll programs screen by screen
List name & location of all variables
List source & destination of GOSUB & GOTO's
Merge 2 or more basic programs
Automatically load & run a series
of Basic programs

MACHINE LANGUAGE AIDS

Examine, change & print RAM contents

Six separate tools!
Each one no more than 1.1K
Complete with 11 pages of documentation

MORETON BAY SOFTWARE
a division of Moreton Bay Laboratory
316 Castillo Street
SANTA BARBARA, CA 93101
805 - 962-3127
Shipping add $2
CA Residents add 6%

Moreton Bay
Software

Circle No. 37 on Reader Service Card