NIBBLES & BITS

THE COMPREHENSIVE MONTHLY NEWSLETTER FOR ADAM USERS

Published by DIGITAL EXPRESS, INC.
1203 Northwoods Drive, Kings Mtn, NC 28086
(704) 739-9106

JULY 1986
vol: 1, nmb: 1
SINGLE ISSUE: $3.50

INSIDE THIS ISSUE:

This issue includes 9 SmartBASIC program LISTs, 6 tables (charts), and 3 assembly-language lists.

EDITOR'S NOTE:

The response to NIBBLES & BITS has been far greater than we expected! It is very encouraging to see so many active ADAMites.

In editing NIBBLES & BITS, I have several goals in mind. My first priority is to do everything that I can to help keep ADAM alive!! To accomplish this, our staff works diligently for YOU to pack as much ADAM info as possible into these 25 to 30 pages (every single month). If you have any questions, comments, or suggestions, please write to us. We sincerely want this newsletter to be what our readers want it to be.

It's been just over a year and a half since Coleco officially announced it was to drop the ADAM. We have inside information from a reliable source that Coleco manufactured in excess of a half million units. Indeed, there are a LOT of us ADAMites!!

Over the past two and half years there has been a tremendous improvement in the quality of software designed for ADAM. In late 1983 there was only a handful of programs available. Much of the early third party (non-Coleco) software was of poor quality. In fact, public domain (non-copyrighted) software being released today is vastly superior to the commericial software of only two years ago!

The problem back then was that, in effect, Coleco refused to relinquish any technical info regarding ADAM or SmartBASIC. It seems to me that their plan was to be the only software developer for ADAM. However, thanks to the persevering research of many devoted ADAM hackers a great deal of progress has been made toward discovering the 'secrets' of ADAM.

Commercial software developed within the past year certainly reflects this new technical knowledge. Every month new titles are released for ADAM. In fact, much of the latest third party software for ADAM is written entirely or (at least) partially in machine language. The quality of ADAM software has never been better!

This good news notwithstanding, several developers of software for ADAM have stopped releasing new titles because of a lack of sales. If we ADAMites aren't careful, this long-awaited availablity of software will soon begin to taper off.

The critical problem facing ADAM owners is the rampant piracy by small groups of wanton ADAM users. Piracy is prevalent throughout the software industry. However, with ADAM unlawful distribution of back-up copies may very well precipitate a virtual standstill in the development of new software!!

Most software developers for ADAM are small businesses that are VERY dependent on sales to maintain growth. When sales decline, these companies start reconsidering their position in the industry. PLEASE . . . if you want to help keep ADAM alive, DO NOT give in to the temptation to trade for or purchase back-ups from unauthorized distributors. If you have any doubts about a distributor's authority with regard to a particular software item, just write to the copyright owner and ask for a list of their authorized distributors.

Software PIRACY may well bring an end to ADAM! Let's all work together to keep ADAM alive!!! Overall, things are looking up for ADAM; let's keep it that way!!

NEWS/UPDATES:

→ Until 12-1-86 NIBBLES & BITS will offer the following discount to YOU for having a friend subscribe to our newsletter. For each friend that you have subscribe, we will extend your subscription by one issue (limit 12). If you get 12 friends to subscribe, you'll get an entire extra year (12 issues) of NIBBLES & BITS absolutely FREE. Just have them send their check or money order with a letter including your identification number (from your mailing label) and your name.

→ DATA DOCTOR the developer of several excellent software packages has discontinued their operation. We have been asked not to elaborate on their reasoning. However, DIGITAL EXPRESS, INC. (our publisher) has purchased the copyrights to all DATA DOCTOR software (both marketed programs and unreleased titles)! In a three tier agreement DEI will eventually purchase the remainder of DATA DOCTOR. In good faith DEI will honor all of the other company's warranties. If you have a matter of concern for DATA DOCTOR, please address it to:

DIGITAL EXPRESS, INC.
ATTN: SS
1203 Northwoods Drive
Kings Mountain, NC 28086

→ Several national department stores still carry the ADAM Computer System. These include: Circus World, Kabee Hobbies, Lionel Leisures, Toys R Us, and ZAYRES. Prices vary demographically ranging from 250 to 300 dollars.

→ American Design Components has a number of electronic parts for ADAM in stock -- including digital data drives for $9.95. To get their FREE catalog, call 1-800-524-0809. Some Radio Shacks also carry ADAM components.

→ We are currently working to gain permission from Lazer Microsystems to release SmartBASIC 2.0. At present, it is available in the software underground. This improved version of BASIC corrects most of V1.0's bugs (including the DATA and REM space bumps and the file handling bugs). It also includes commands for switching between standard memory and extended memory (using a 64K expander) and a MERGE command.

→ DIGITAL EXPRESS, INC. has released two new software titles: Intel-BEST and Intel-LOAD. Both of these are pure machine code utilities. To maintain the integrity of our staff reviews, we will not directly review DEI products. However, we will accept/print evaluations submitted by NIBBLES & BITS subscribers.

→ Several companies are now developing peripherals for ADAM. These include: CAPITAL SOFTWARE, EVE Electronics, JJ's Gourmet Hardware & Software Emporium, and Orphanware. EVE Electronics is currently working on a disk controller card that will permit ADAM to be connected to non-Coleco disk drives. The quality and price of peripherals vary widely from one company to another. For your own benefit, you should carefully compare before purchasing ADAM hardware.

→ Coleco has fulfilled their contract requirements for disk drives. It is unlikely that they will ever again restart manufacture of any ADAM product. Also, they will repair out-of-warranty disk drives for about $170.00. Obviously, this exorbitant price is intended to discourage repair requests (they are required by law to provide repair service).

→ Beginning with the September issue of NIBBLES & BITS we will start a new department: ADAM USERS FORUM. In this department, we will answer questions sent in by our readers. So . . . send in your questions. If you send a self-addressed, stamped envelope with your questions, we'll also mail the answers to you personally.

→ Please include your identification number on all correspondence to us. When you send in contributions (articles, questions, comments, reviews, etc.), please indicate whether or not you want us to include your name and address.

→ As you can see there is not a lot of new news this month. Most software companies curtail release of new products during the summer, in anticipation of the back-to-school business flurry. This common practice coupled with the fact that many ADAM software developers are hurt by declining sales (due primarily to widespread piracy) will no doubt insure a slow summer for ADAM.

→ One of the most exciting events for ADAM this summer will be the results of the hardware 'price war'. We should all benefit both by having a larger selection of peripherals to choose from and by the patent savings. But don't wait too long to expand your system, prices tend to rise sharply after these competitive battles for customers.

BIT BY BIT:

Common Questions:

→ What is a computer?

A computer is a machine that accepts information, processes it according to specific instructions, and can provide the results as new information. It can store/retrieve large quantities of data at very high speeds. And a computer can make logical decisions and comparisons.

→ What is meant by hardware and software?

Hardware (peripherals) refers to the computer itself and its physical devices: data drives, disk drives, printers, keyboards, etc. Software refers to the programs (instructions) which are entered into the computer's memory to make it perform specific tasks. In general, if you can touch it, it's hardware -- if you can't, it's software.

→ What is a program?

A program is a list of instructions which direct the computer to perform its operations. The computer never does its own thinking; rather, it merely repeats the operations stored in its memory. A program instructs the computer as to which operations it will perform and the sequence it will perform them in.

→ What are computer languages?

In order to program a computer, there must be some way to give instructions to the computer. Internally, all computers understand only one language -- machine code. Machine code is simply a set specially coded numbers that the computer understands. ADAM is primarily controlled by a single microprocessor unit (MPU, the brain of a computer). This primary 'brain' is a Z80 MPU. Each particular type of MPU uses its own specific machine code.

Programming in machine language requires an in-depth understanding of a computer and a meticulous attention to detail. For this reason, a variety of high level languages (languages which use words rather than numbers) have been developed.

One of the most widespread of these languages is BASIC (Beginners All-purpose Symbolic Instruction Code). ADAM's Smart-BASIC is an enhanced version of BASIC.

SmartBASIC acts as an interpreter translating BASIC commands into machine code as each command is read by ADAM. For this reason, SmartBASIC MUST be loaded into memory before a BASIC program can be executed (RUN).

Programming Perspectives:

SmartBASIC permits two programming modes. In immediate mode all commands are executed immediately, ie, as soon as [RETURN] is pressed. This mode is sometimes called calculator mode because it is frequently used by BASIC programmers to do arithmetic computations.

Programming or deferred execution mode requires the use of line numbers. With ADAM a line number may be any integer between 0 and 65535, inclusive. When a BASIC program is executed (RUN), ADAM reads the instructions in line number sequence. With this feature, you may enter program lines out of order -- ADAM will automatically sequence them for you. It is common practice among experienced BASIC programmers to use 100 as the first line number and increment subsequent lines by 10, ie, 100, 110, 120, etc. Using this technique, it is very easy to add program lines between existing ones when editing. As you will no doubt discover, editing can be a frequent task in BASIC programming.

While programming, you should not confuse the capital letter 'O' with the number '0' or the lower case 'l' with the number '1'. ADAM recognizes each of these characters differently. You'll get an error message if you use them improperly.

When learning to program you'll probably have many frustrating moments. It may help to remember that you are controlling ADAM, but within the guidelines of SmartBASIC. Sure SmartBASIC has its bugs (program errors), but most error messages are caused by typos. When something doesn't work the way you expect it to, examine it. Chances are that you accidently entered an incorrect character. While programming, you should also keep in mind that ADAM doesn't read anything that you type until you press [RETURN].

BASIC programming is very logical and fairly easy, but it does take a lot of effort on your part. Read, study, EXPERIMENT, practice!

Getting the most out of the PRINT command:

PRINT is one of the most frequently used commands in BASIC. It is used primarily to display words and numbers on the video screen. When BASIC was first developed by John Kemeny and Tim Kurtz at Dartmouth College in the early nineteen sixties, computers revealed data using the printer. The PRINT command of today is carry-over from that era (it is still PRINT; not DISPLAY).

PRINT is an output command. Not only can it display data on the screen; but, it can also be used to print characters using a printer and to store data in text files. We'll discuss these applications in a later issue.

There are two types of information that a computer can use: numbers and words. For practical purposes, the difference between these two is that numbers can be used in mathematical calculations and words can not. Simple, huh? The PRINT command can be used to display both numbers and words.

The following simple program demonstrates some of the features of the PRINT command:

```
10 PRINT "A simple test."
20 PRINT 8
30 PRINT 6*5
```

The data that follows the PRINT command is called its 'argument' or 'parameter'. When the parameter is a word or string of words, it must be enclosed in quotes. Line #30 will display the result of 6 times 5, ie, 30. As you develop your programming skills, you'll find this to be a very convenient feature.

ADAM automatically inserts a carriage return at the end of a PRINT command's parameter. You can bypass this by placing a semi-colon at the end of the parameter. If you include exactly 31 characters as a PRINT parameter, BASIC will insert an extra carriage return. To correct this bug, simply follow the parameter with a semi-colon.

The comma may also be used in a PRINT statement. It's purpose is the segregate data into two columns. There is a bug with the comma feature. The first element in the first column will not line up properly. For instance, enter the following program line:

```
10 PRINT 0,1,2,3,4,5
```

When executed, you'll see that the zero is offset to the left by one space. To correct this problem, simply insert a blank space before the first element. To demonstrate:

```
10 PRINT " "; 0,1,2,3,4,5
```

PRINT is used so frequently that a shortcut command for it has been implemented. When programming, you may substitute a question mark (?) for PRINT. When you LIST the program, you'll see that BASIC has converted the punctuation to the actual PRINT command. Also, if you enter the PRINT command without a parameter, ADAM will print a blank line.

LIST parameters:

The LIST command is used to display any or all of the BASIC program lines currently stored in memory. Without any parameters, LIST will display the entire program in numerical line number sequence. The LISTing will be formatted from the left margin and include numerous spaces that you probably did not enter.

If your LIST contains more information than one screen will display, you'll most likely want to slow down or pause the display. To slow the LIST, use the SPEED command. For example:

```
SPEED = 100  [RETURN]
LIST  [RETURN]
```

The value of the SPEED parameter may be any integer between 0 and 255, inclusive. The SPEED value is normally set to 255 (the fastest setting).

To pause the LIST, use the CONTROL-S feature. While the program is being LISTed, hold down [CONTROL] and press the 'S' key. To resume the LIST, just press any other key. To stop the LIST use CONTROL-C in the same manner. To restart the LIST after using CNTL-C (provided you haven't typed anything else), enter CONT [RETURN].

You may want to use LIST for certatin parts of your program. If you want to LIST just one line, type that line number after LIST. For example:

```
LIST 100  [RETURN]
```

To LIST all the program lines up to 100, do this:

```
LIST  , 100   [RETURN]          or,

LIST  - 100   [RETURN]
```

Note that either the comma or the hyphen may be used with the LIST command.

To LIST all the program lines beyond 100, do this:

```
LIST 100 -  [RETURN]
```

To LIST a specific range of program lines, do this:

```
LIST 100 - 150  [RETURN]
```

This vital command will only LIST program lines in numerical sequence. Therefore, when LISTing a range of lines, the second number must be greater than the first number.

BASIC line editing features and secrets:

There are numerous ways to edit BASIC program lines. The simplist method is to just retype your program line using the same line number -- the new line will replace the old one.

You can use a similar technique to delete a program line. Just type the line number and press [RETURN]. Let's cancel the following program line.

```
10 PRINT (5*2)+16
```

All you need to do is this:

```
10 [RETURN]
```

When you press [RETURN], ADAM reads your program line. BASIC will search for errors in punctuation and spelling of BASIC words alerting you to the mistake with error messages. It is your responsability, as a programmer, to determine and correct other errors. If you notice a mistake before you press [RETURN], just use the left arrow key or [BACKSPACE] to move the cursor back to the error and begin retyping from there.

Another method of line editing is to move the cursor across the entire program line making corrections as you go. However, with line numbers greater than 9999, another SmartBASIC bug is encountered. The cursor will not move to the left of the prompt (the right bracket). To overcome this problem, just press [ESCAPE] first (or any of the special function keys).

Some of the special [CONTROL] functions may also be of use to you in editing your programs. To implement any of these features, hold down [CONTROL] and press the appropriate key.

The CNTL-O function will delete the character directly above the cursor. It will also pull any characters to the right of the cursor left, thus inserting a blank space at the end of the screen line.

Th CNTL-N function is the opposite of CNTL-O. It inserts a blank space directly above the cursor. Doing so, it pushes the characters to the right of the cursor to the right deleting characters which are pushed beyond the right margin.

The CNTL-X function deletes the program line currently being typed. You will need to press [RETURN] after using this feature.

One of the more interesting editing features of SmartBASIC is the use of [CONTROL] with the arrow keys. Suppose you enter the following line:

```
10 PRINT " This is a test."
```

You decide after pressing [RETURN] that you want the line to print 'This is a simple test.'. Let's use the [CONTROL]-arrow key functions to insert the word 'simple':

Move the cursor underneath the first digit of the line number. Then move it to the right until you get to the first letter of 'test'. Now press CNTL-down arrow. Continue moving the cursor downward until you reach a clear part of the screen. Now release [CONTROL] and the arrow key. Next, type the word 'simple' plus a blank space. Now use CNTL-left arrow to move the cursor back to the first letter of 'simple'. Then use CNTL-up arrow to move the cursor back to where you started from -- underneath the first letter of the word 'test'. Release the [CONTROL] and up arrow keys. Finally, move the cursor to the end of the line using only the right arrow key. When finished, press [RETURN].

List line # 10. If you've done everything correctly, you'll see the word is inserted. Obviously, this technique of insertion is more vital when editing long program lines. Also, you may use CNTL-right arrow to pass over unwanted characters when editing.

SmartBASIC will permit you to enter 128 characters on one program line, ie, four screen lines plus four characters. It is a good practice, however, to limit your program lines to three screen lines. This is because BASIC adds many extra spaces. If you enter a program line that is 128 characters long, it may LIST to 180 characters with the added spaces. You should always leave room for editing.

Postscript:

When writing articles for the BIT BY BIT department, we assume that you have at least made some programming effort on your own. These articles will be benefit you most if you study them in conjunction with 'PROGRAMMING WITH ADAM' which came with your computer. Remember programming takes effort on your part: read, study, EXPERIMENT, and practice.

HAPPY PROGRAMMING . . .

BYTE-SIZED BASIC:

The ASCII code (PART I):

The ASCII code (American Standard Code for Information Interchange) is a system for using numbers to represent various characters. The code is standardized so that it may be used on any computer.

ASCII (pronounced 'as-key') represents all the letters of the alphabet, the special symbols, and control functions as numbers ranging from 0 to 255, inclusive. Page C-12 of 'PROGRAMMING WITH ADAM' lists the first 128 of these values and their corresponding characters. Below is a listing of some of the more esoteric ASCII values.

Understanding the arrangement of the values may be of some use to you. CNTL-A has an ASCII value of 1. The upper case letter 'A' has a value of 65 (1 + 64). The inverse letter 'A' has a value of 193 (1 + 192 or 65 + 128).

Test your understanding:
Using only your reasoning ability, determine the ASCII values of CNTL-E, capital letter 'E', and inverse capital letter 'E'.

When working with ASCII values, you will most likely use these BASIC commands: ASC, CHR$, GET and PEEK (64885). As an intermediate level BASIC programmer you have no doubt used these commands many times. On the next page are two simple demonstration programs using ASCII values.

<ASCII>	<KEYPRESS>	<ASCII>	<KEYPRESS>
127	CNTL - DELETE	154	SHIFT - MOVE/COPY
128	HOME	155	SHIFT - STORE/GET
129	I	156	SHIFT - INSERT
130	II	157	SHIFT - PRINT
131	III	158	SHIFT - CLEAR
132	IV	159	SHIFT - DELETE
133	V	160	UP ARROW
134	VI	161	RIGHT ARROW
137	SHIFT - I	162	DOWN ARROW
138	SHIFT - II	163	LEFT ARROW
139	SHIFT - III	164	CNTL - UP ARROW
140	SHIFT - IV	165	CNTL - RIGHT ARROW
141	SHIFT - V	166	CNTL - DOWN ARROW
142	SHIFT - VI	167	CNTL - LEFT ARROW
144	WILD CARD	168	RIGHT ARROW - UP
145	UNDO	169	RIGHT ARROW - DOWN
146	MOVE/COPY	170	LEFT ARROW - DOWN
147	STORE/GET	171	LEFT ARROW - UP
148	INSERT	172	HOME - UP ARROW
149	PRINT	173	HOME - RIGHT ARROW
150	CLEAR	174	HOME - DOWN ARROW
151	DELETE	175	HOME - LEFT ARROW
152	SHIFT - WILDCARD	184	SHIFT - BACKSPACE
153	SHIFT - UNDO	185	SHIFT - TAB

```
   10 REM ASCII demo
   20 PRINT " Press the space bar to end."
   30 PRINT " Press any other key and its"
   40 PRINT " ASCII value will be displayed.": PRINT
   50 GET key$
   60 IF key$ = CHR$(32) THEN  PRINT " THAT'S ALL!!": END
   70 PRINT " The value is:  ";ASC(key$)
   80 PRINT: PRINT: GOTO 20
```

```
   10 REM SmartKEY referencing
   20 DATA I,II,III,IV,V,IV
   30 DATA option #1,option #2,option #3
   40 DATA option #4,option #5,option #6
   50 FOR x = 1 TO 6: READ sk$(x): NEXT
   60 FOR x = 1 TO 6: READ choice$(x): NEXT
   70 flag$ = "off": lower(1) = 129: lower(2) = 137
 1000 TEXT: PRINT: PRINT " Which do you prefer?"
 1010 FOR x = 1 TO 6: le% = LEN(sk$(x)): VTAB 2*x+4: HTAB 2
 1020 PRINT sk$(x);SPC(5-le%);choice$(x): NEXT
 1030 VTAB 23: HTAB 1: GET key$
 1040 upper(1) = 134: sk% = ASC(key$): GOSUB 20000
 1050 IF flag$ = "on" THEN  GOSUB 20500: GOTO 1070
 1060 GOTO 1030
 1070 ON sk% GOTO 2000,3000,4000,5000,6000,7000
 2000 HOME: PRINT "SmartKEY I": END
 3000 HOME: PRINT "SmartKEY II": END
 4000 HOME: PRINT "SmartKEY III": END
 5000 HOME: PRINT "SmartKEY IV": END
 6000 HOME: PRINT "SmartKEY V": END
 7000 HOME: PRINT "SmartKEY VI": END
19999 END
20000 upper(2) = upper(1)+8
20010 IF sk% < lower(1) OR sk% > upper(2) THEN  RETURN
20020 flag$ = "on"
20030 IF sk% >= lower(1) AND sk% <= upper(1) THEN  RETURN
20040 sk% = sk%-8: RETURN
20500 flag$ = "off": sk% = sk%-128: RETURN
```

Each of the two preceding programs demonstrate applications of the ASCII code. The first program is very simple and straightforward. You may want to use it to experiment with various ASCII values.

Many ADAMites are interested in making use of the SmartKEYs in their own programs. The second program demonstrates this application. Line numbers 10 through 70 initialize the various values. Line numbers 1000 through 1070 make use of the SmartKEY routine. Line numbers 2000 through 7000 are set aside for user options. Line numbers 20000 through 20500 contain the actual SmartKEY routine.

As you'll notice, ADAM recognizes both unSHIFTed and SHIFTed SmartKEY inputs. The SHIFTed ASCII value is eight higher than the corresponding unSHIFTed value. The variables are:

choice$(dim)	= menu choices
flag$	= indicates SmartKEY input
le%	= length of menu choice
lower(1)	= lower unSHIFTed limit
lower(2)	= lower SHIFTed limit
upper(1)	= upper unSHIFTed limit
upper(2)	= upper SHIFTed limit
sk$(dim)	= Roman numerals
sk%	= ASCII value of input
x	= work variable

POKEs to play with (PART I):

In this section of NIBBLES & BITS, we'll reveal many of BASIC's interesting POKEs. Using these POKEs, you'll be able to customize BASIC after it's LOADed. Some of the most useful POKEs are those that change the various screen colors. This month we'll concentrate primarily on these.

Internally ADAM recognizes only one set of color codes. The GR and HGR codes are an arbitrary convention of SmartBASIC. Later in this article, we'll explain how to CORRECT the GR and HGR color tables so that you'll only need to memorize (or refer to) one list of 16 color values.

Unless otherwise mentioned, all POKEs are in reference to SmartBASIC V1.0 (version 79). PEEK (260) discloses your version of V1.0.

ADAM is controlled by several microchips. One of these is a 16K video chip. The information programmed on this chip controls everything that you see on your monitor (or TV) screen. This chip is capable of 15 independent colors. This is ADAM's master color code. It's the only one that you really need to know.

Numerical sequence:

0	= transparent
1	= black
2	= medium green
3	= light green
4	= dark blue
5	= medium blue
6	= dark red
7	= aqua/cyan
8	= medium red
9	= light red
10	= dark yellow
11	= light yellow
12	= dark green
13	= magenta
14	= gray
15	= white

Color sequence:

→ RED:	
dark red	= 6
med red	= 8
lght red	= 9
magenta	= 13
→ BLUE:	
dark blue	= 4
med blue	= 5
aqua/cyan	= 7
→ GREEN:	
dark grn	= 12
med grn	= 2
lght grn	= 3
→ YELLOW:	
dark ylw	= 10
lght ylw	= 11
→ MISCELLANEOUS:	
black	= 1
white	= 15
gray	= 14
trnsprnt	= 0

SmartBASIC V1.0 has nine addresses that control screen color. These are:

17059 = TEXT background
17115 = TEXT NORMAL
17126 = TEXT INVERSE

18607 = GR background
18633 = GR window
18711 = GR TEXT

25431 = HGR background
25471 = HGR window
25568 = HGR TEXT

Changing the background colors is very easy. Just select your color preference and POKE it's color code into the appropriate address.

```
POKE 17059, value
POKE 18607, value   or,
POKE 25431, value
```

Suppose you want the TEXT background to be dark red. The code for that color is six. All you need to do is:

```
POKE  17059,  6:  TEXT      [RETURN]
```

Changing the remaining colors is a little more complex. Let's use variables to facilitate the explanation. The variables are:

nl = normal letters
ns = normal screen
il = inverse letters
is = inverse screen
gw = graphics window

Just substitute your color code preferences for the variable names in the following equations. You may use these equations in immediate or programming mode.

TEXT NORMAL:
```
POKE 17115, (nl * 16) + ns
```

TEXT INVERSE:
```
POKE 17126, (il * 16) + is
```

GR window:
```
POKE 18633, (gw * 16) + gw
```

GR TEXT:
```
POKE 18711, (nl * 16) + ns
```

HGR window:
```
POKE 25471, (gw * 16) + gw
```

HGR TEXT:
```
POKE 25568, (nl * 16) + nl
```

Suppose you want the NORMAL TEXT letters to be black and the NORMAL TEXT screen to be aqua. You would just do this:

```
POKE  17115,  (1*16)+7:TEXT     [RETURN]
```

Points to consider:

→ You have to enter the corresponding graphics mode command (TEXT, GR, or HGR) after using the POKEs in order to implement the color changes.

→ Transparent is colored with respect to the current background color.

→ FLASH, in the TEXT mode, is the result of alternating between NORMAL and INVERSE colors.

→ You should not set corresponding letter and screen colors to the same value -- you will not be able to read anything.

→ You may want to PEEK these addresses to determine their original values before POKEing in new values.

From this point forward, we shall refer to the original value (when SmartBASIC is LOADed) as its 'default' value. This is common terminology among experienced programmers.

The value of the current GR COLOR is at address 16776. The value of the current HGR HCOLOR is at address 16777. These color values use ADAM's master color code listed on page 10 of this issue. One technique to use just one color table (not three) is to simply POKE your color code preferences into these addresses rather than using the COLOR and HCOLOR commands.

However, there is a superior technique for accomplishing this convenience: CORRECT the GR and HGR color tables. To do this, use the following lines at the beginning of your program. As an added bonus, this procedure even corrects the SCRN color table. Now you only need to learn one color table when working with SmartBASIC!!!

```
10 FOR x = 0 to 15
20 POKE 18765 + x, x
30 POKE 18781 + x, x
40 NEXT x
```

There are even more benefits when employing this technique. You can use transparent as a color. And you even have a wider range of colors to choose from as the default color tables omit some of the possible colors.

How would you like to be able to 'merge' BASIC programs directly from SmartBASIC? Sure SmartWriter provides a facility for this, but it takes a lot of unnecessary time switching between SmartWriter and SmartBASIC.

The BASIC LOAD command does four tasks. It executes NEW, CLEAR, searches for the filename, and then LOADs it. To merge programs in BASIC, all you have to do is disable the NEW function.

The machine language execution routine for NEW begins at address 6356. The default value at that address is 205 (Z80 machine code for CALL). To disable NEW, just POKE a 201 into 6356. This value is Z80 machine code for RETURN. As soon as ADAM tries to execute the NEW function, it thinks its finished without doing anything at all.

Now you can use the LOAD command to merge subprograms and routines directly from SmartBASIC. Be sure to POKE 205 back into 6356 when you're done.

SmartBASIC includes several undocumented commands. Two of these are BREAK and NOBREAK. They were intended to disable and enable the CNTL-C function. However, SmartBASIC doesn't make use of them.

A simple POKE, on the other hand, can permit these features. The ASCII value of CNTL-C is 3. Address 16134 contains the ASCII value that enables the CNTL-C function. You can change it to any ASCII value that you desire.

If you POKE a 255 into 16134, you'll effectively disable CNTL-C because there is no keypress that corresponds to ASCII value 255. You may find it very convenient, while programming, to change the BREAK value to 27, ie, [ESCAPE].

Trigonometric graphics (PART I):

By HPLOTing the graphs of trigonometric equations you can create an almost infinite variety of interesting graphics displays. You don't have to a whiz at trigonometry to make use of these principles, but it helps. The three programs on the next page illustrate some techniques that you can use in your own programs.

ADAM uses radians rather than degrees for trigonometric functions. One radian is 'pi' divided by 180 or 0.0174532925 degrees. You can let ADAM calculate the value of 'pi' with this equation: pi = ATN(1) * 4.

The first program draws a 3-dimensional bowl. Line numbers 100 through 130 cause the apparent simultaneous drawing in different directions. Experiment with different values and you can create quite a variety of designs with just this one algorithm (program module).

The next two programs draw circles, empty and filled, respectively. By transposing the trig functions in lines 110 and 120 and changing the parities in line number 130, you can alter the direction and starting point of the HPLOTing.

```
 10 REM 3-D HPLOTing #1
 20 HGR: PRINT: PRINT " Press CNTL-C to stop . . ."
 30 HCOLOR  = 7: pi = ATN(1)*4
 40 FOR w = 0 TO 15: FOR x = 1 TO 60: y = x/2
 50 z = y^2+(w*2)^2
 60 z = SIN(pi*SQR(z)/15)*25
100 HPLOT (128+x),(80+z+w)
110 HPLOT (128+x),(80+z-w)
120 HPLOT (128-x),(80+z+w)
130 HPLOT (128-x),(80+z-w)
140 NEXT x: NEXT w
```

```
 10 REM draw empty circle
 20 HOME: INPUT " Enter radius (5-75): ";rds%
 30 h1 = 128: v(1) = 80
 40 HGR: HCOLOR  = 7
 50 pi = ATN(1)*4: radian = pi/180
100 FOR point = 0 TO 2*pi STEP radian
110 h2 = rds%*SIN(point)
120 v(2) = rds%*COS(point)
130 HPLOT h1+h2,v(1)-v(2): NEXT point
```

```
 10 REM draw filled circle
 20 HOME: INPUT " Enter radius (5-75): ";rds%
 30 h1 = 128: v(1) = 80
 40 HGR: HCOLOR  = 7
 50 pi = ATN(1)*4: radian = pi/180
100 FOR point = 0 TO 2*pi STEP radian
110 h2 = rds%*SIN(point)
120 v(2) = rds%*COS(point)
130 HPLOT h1,v(1) TO h1+h2,v(1)-v(2)
140 NEXT point
```

HACKER'S DELIGHT:

Z-80 instructions and registers:

SmartBASIC is a collection of useful machine language routines. These routines are executed by dint of symbolic words (the BASIC keywords). As an extension of machine code, BASIC is obviously much more limited in its capabilities. Serious BASIC programmers eventually delve into machine code programming due to these restrictions.

You should find it relatively easy to make the transition to 'ml' programming if you start by creating machine code enhancements to BASIC. For this reason, we'll begin our study of ADAM's Z80 code by creating simple routines. Once we've mastered the fundamentals, we'll develop entire programs in machine code.

BASIC is essentially a 'sloppy' language. Machine code programming, on the other hand, requires elaborate precission and a thorough understanding of the computer. Furthermore, as an extension of a game system, ADAM is much more complex than most home computers.

ADAM's Z80 MPU directs the activities of the computer by interpreting a set of instructions called 'operation codes' or 'op codes'. These 'op codes' are permanently stored in the main memory of the Z80. This MPU contains a set of 'registers' which are internal memory locations used for data storage and manipulation. All machine language programming is accomplished by using these registers in various ways. Registers are completely independent of the 64K standard RAM and the 16K video RAM. However, the registers are used to store data on these two RAM chips.

It may augment your understanding if you consider each address of the 64K RAM as a post office box. Extending this analogy, registers would be the postal carriers delivering mail (data) and you, as a programmer, would be the postmaster general directing all of the various activities. When you've eperimented with machine code a little, reread this analogy and you'll see how applicable it really is.

It is very convenient (almost mandatory) in 'ml' programming to use an operating system. An 'OS' is a wide range of 'ml' routines which control the various peripherals connected to the system. ADAM is already equipped with an EOS (Elementary Operating Systyem) ROM. We will use the routines from this chip frequently.

What is assembly-language:

Assembly-language is not a true language as BASIC is; rather it is a pseudo-language. Assembly-language uses mnemonics (a system of symbols to assist the memory) to represent the various machine code values. For example: LD DE,nnnn is the mnemonic equivalent of 17,0,0 in machine code. The 'LD' stands for 'LOAD'. The DE is a pair of registers. Each pair of "n's" represents an integer in the range of 0 through 255. The comma indicates that the value of 'nnnn' will be LoaDed into the DE register pair. If you're still a little confused, don't worry. As we progress, it will make more sense to you.

High and low order bytes:

The highest value that a single address or register can contain is 255. Because of this limitation, a technique is commonly used that allows the Z80 to recognize values greater than 255. This method is referred to as using high and low order bytes. The principle is fairly simple.

Any number between '0' and '65535' can be represented as two integers between '0' and '255'. The high byte value is multiplied by 256 and added to the low byte value. Let's demonstrate.

Suppose we are given 100 as the high byte value and 16 as the low byte value. The result of these two is (100*256) + 16. This can be simplified to 25600 + 16 or 25616.

Suppose we are asked to convert 17115 to high and low byte values. We, simply, use the reverse of the previous principle. The high byte value equals INT(17115/256). This can be simplified to INT(66.86). Thus the high byte value for 17115 is 66. The low byte value equals 17115 - (66*256). This can be simplified to 17115 - 16896. Thus the low byte value of 17115 is 219.

Now we have determined that the high byte of 17115 is 66 and its low byte is 219. Let's verify these results.

```
(66*256)+219  =  16896+219  =  17115
```

Understanding the concept of high and low order bytes is ESSENTIAL to learning machine code or assembly-language programming. Listed below are a few more examples of high and low bytes. Practice with them until you have thoroughly mastered this concept.

```
high byte = 175
low byte  = 87
the value = 44887

high byte = 255
low byte  = 255
the value = 65535

high byte = 0
low byte  = 117
the value = 117

high byte = 211
low byte  = 0
the value = 54016

high byte = 87
low byte  = 214
the value = 22486
```

Instructions and registers revisited:

The Z-80 is an updated version of the 8080 MPU. The Z-80 includes 158 instruction types that give a very large number of total commands (op codes) with all the variations. The Z-80 includes 22 registers. In the next few issues we'll concentrate on eight of these, the general-purpose registers.

These general-purpose registers are labeled: A,F,B,C,D,E,H, and L. Each of these is an 8-bit register. However, they may be combined in pairs to form 16-bit pseudo-registers: AF, BC, DE, and HL.

The Z-80 also includes eight alternate general-purpose registers: A', F', B', C', D', E', H', and L'. This alternate set can not be used at the same time as the standard set. The alternate set is used primarily for interrupt processing -- we'll discuss this concept in a later issue.

The remaining registers include: the 16-bit stack pointer (SP), the 16-bit program counter (PC), a 16-bit index register (IX), another 16-bit index register (IY), the 8-bit interrupt register (I), and the 8-bit refresh register (R).

If you are new to machine language programming, some of this information may seem confusing -- just stay with it. Remember, BASIC seemed a little complex at first too. If you've been alert to what you've read thus far, you should have inferred a vital fact: 16-bit registers are used for high and low order bytes!

The hexadecimal number system:

The 'hex' code is used primarily by assembly-language programmers. It is a base 16 number system. It's digits are: 0, 1, 2, . . . 8, 9, A, B, C, D, E, F. Letters are used to represent the digits ten through fifteen.

```
 10 REM hex to deciaml chart
 20 hex$ = "0123456789ABCDEF": dec = 0
 30 FOR x = 1 TO 2: space$(2) = space$(2)+" ": NEXT
 40 FOR x = 1 TO 3: space$(3) = space$(3)+" ": NEXT
 50 PR #1: PRINT: PRINT space$(2);space$(3);space$(3);
 60 FOR x = 1 TO 16: PRINT MID$(hex$,x,1);space$(3);
 70 NEXT: PRINT: PRINT
100 FOR x = 1 TO 16: PRINT space$(3);MID$(hex$,x,1);" ";
110 FOR y = 1 TO 16: PRINT " ";
120 IF INT(dec/100) = 0 THEN  PRINT " ";
130 IF INT(dec/10) = 0 THEN  PRINT " ";
140 PRINT dec;: dec = dec+1
150 NEXT y: PRINT: NEXT x: PR #0
```

HEX/DECIMAL CHART

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

The simple program at the top of this page will generate the 'HEX/DECIMAL CHART'. The chart is a convenient utility for converting between the hex and decimal number systems.

You can easily determine the decimal equivalent of a 2-digit hex number by cross-referencing the lefthand column with the row on top. You can calulate the decimal value of a 4-digit hex number by multiplying the decimal equivalent of the left two digits by '256' and adding this result to the decimal equivalent of the right two digits.

To insure that you understand the technique of converting between the hex and decimal systems, let's try a few examples.

Suppose we are given the hex number 'A8' to convert to its decimal equivalent. Locate the 'A' in the lefthand column. Next, locate the '8' in the row on top. Follow the two lines of numbers to their intersection. Here you'll find the decimal number '168'. Thus, 'A8' in hex equals '168' in decimal.

Now suppose we want to convert the decimal number '250' to its hex equivalent. Just locate '250' inside the chart. The value in the lefthand column from '250' is 'F'. The value at the row on top is 'A'. Thus the hex equivalent of the decimal number '250' is 'FA'.

Understanding the hex system is critical to assembly-languge programming. Please examine the following examples.

1B (hex) equals
27 (decimal)

DD (hex) equals
221 (decimal)

104 (decimal) equals
68 (hex)

80 (decimal) equals
50 (hex)

FFFF (hex) equals
65535 (decimal)

61CB (hex) equals
25035 (decimal)

17115 (decimal) equals
42DB (hex)

The video chip:

Of ADAM's standard 80K of Random Access Memory only 64K is directly accessible from SmartBASIC and the Z-80. The remaining 16K is an indirect memory. It is this 16K video chip that controls all of ADAM's video output.

Machine code within the execution routines of the TEXT, GR, and HGR commands set up a variety of tables on the VDP (Video Display Processor). The tables established by the GR and HGR commands are similar. However, the information from these tables are used differently.

Initially we'll concentrate on the TEXT tables and how to modify them. The TEXT command only uses about half of the VDP RAM. In later issues we'll reveal how to employ this free RAM in your BASIC programs so that you'll have access to 34K rather than 27K.

In BASIC TEXT mode ADAM uses four VDP tables. The character definition table is a bit-mapped (an 'ml' technique for drawing) table which defines the shapes of each font (displayable character). Each font is defined by eight bytes. This table is set up for 256 fonts. Thus it uses 2048 bytes. On the VDP it uses addresses '0' through '2047'.

Two tables are used to store the screen image (everything that appears on the screen). Except for the cursor and any FLASHing characters, the tables are identical. The screen image alternates continuously between the two tables. FLASH works by storing NORMAL letters in one table and their INVERSE equivalents in the other table. Each table is 768 bytes in length (32 * 24). The first table uses addresses '2048' through '2815'. The second table uses addresses '6144' through '6911'.

The character color table uses addresses '8192' through '8223' (32 bytes). Each byte controls the color of eight characters. This is the simplist table to modify. Let's take a look at how it's mapped (arranged).

BASIC TEXT mode VDP color table:

DISP	VDP ADDR	ASCII code
0	8192	0 - 7
1	8193	8 - 15
2	8194	16 - 23
3	8195	24 - 31
4	8196	32 - 39
5	8197	40 - 47
6	8198	48 - 55
7	8199	56 - 63
8	8200	64 - 71
9	8201	72 - 79
10	8202	80 - 87
11	8203	88 - 95
12	8204	96 - 103
13	8205	104 - 111
14	8206	112 - 119
15	8207	120 - 127
16	8208	128 - 135
17	8209	136 - 143
18	8210	144 - 151
19	8211	152 - 159
20	8212	160 - 167
21	8213	168 - 175
22	8214	176 - 183
23	8215	184 - 191
24	8216	192 - 199
25	8217	200 - 207
26	8218	208 - 215
27	8219	216 - 223
28	8220	224 - 231
29	8221	232 - 239
30	8222	240 - 247
31	8223	248 - 255

In the table above the first column represents the displacement which is added to 8192. 8192 is the starting address of the BASIC TEXT mode color table in VRAM (Video Random Access Memory). We'll use this displacement value in our machine language routines to change the color of various TEXT mode fonts.

Z-80 instructions in detail:

You may wonder how an 'ml' programmer learns all the various Z-80 op codes. With all the variations of the 158 instruction types, there are over 700 documented op codes for this MPU. In reality, you will eventually memorize many of them simply by repeated use. In the beginning though, you'll need to refer to a listing of all the op codes. Most Z-80 instruction manuals include such a list. Two of the best Z-80 tutorials are: '8080/Z80 Assembly Language' by Alan R. Miller and 'Programming the Z80' by Rodnay Zaks.

We have compiled two lists of the Z80 op codes. Each list includes over 700 instuctions with the decimal and hex values and the assembly-language mnemonics. One list is numeric and the other is alphabetic. Each list is four double-sided pages in length. We will sell each list for one dollar (to cover processing expenses). Please include a business-size self-addressed, stamped envelope with your order.

OS routines:

Accessing the video chip requires precise timing. Of course, you can write an 'ml' routine 'from scratch' that will account for the time element (in microseconds). However, it is generally more practical to use the 'OS' routines provided by Coleco on the EOS ROM.

In BASIC, addresses 64560 (FC30) through 64860 (FD5C) constitute the EOS jump table. The table is consists of several dozen 3-byte jumps to the various OS routines. Thirteen routines are included for VDP access.

VRAM OS table:

64791 (23,253) (FD17)
put ASCII to VDP

64794 (26,253) (FD1A)
write to VRAM

64797 (29,253) (FD1D)
read from VRAM

64800 (32,253) (FD20)
write to VDP register

64803 (35,253) (FD23)
read from VDP register

64806 (38,253) (FD26)
write one byte to VRAM

64809 (41,253) (FD29)
init VRAM table

64812 (44,253) (FD2C)
put VRAM

64815 (47,253) (FD2F)
get VRAM

64818 (50,253) (FD32)
calculate offset

64821 (53,253) (FD35)
point to pattern position

64824 (56,253) (FD38)
load ASCII to VDP

64827 (59,253) (FD3B)
write to sprite attribute table

The table above lists the thirteen 'OS' VDP controls. The format is: RAM address, low and high byte equivalent, hex equivalent, and routine function. Note that in decimal the low byte precedes the high byte, but in hex the high byte precedes the low byte. This difference is VERY important.

When using these routines, specific registers must contain certain values. We'll show you how to use each routine. But for now let's take a look at the VDP registers.

The VDP registers:

The 16K video chip contains nine registers of its own. As with the Z-80 registers, these are essentially independent of RAM. However, the VDP registers are used in a different manner. Each of these registers is used to store certain screen controls. Several of the registers are pointers to the various VRAM tables.

Register seven controls the background screen color. In SmartBASIC you can change the background color with a particular POKE followed by the corresponding graphics mode command (refer to page 11 of this issue). Using these POKEs has one drawback: you have to clear the screen to implement a color change.

Let's create an 'ml' routine that will instantly change the background (without clearing the screen). SmartBASIC includes a lot of unused RAM below 27407, but many enhancements to BASIC use this space. For this reason we recommend that you store your 'ml' routines between 27600 and 28000. First you'll need to set LOMEM to 28000 and then you'll need to POKE the routine into this reserved RAM.

The routine at address 64800 controls a write to VDP register function. This routine requires that the Z-80 register 'B' contain the number of the VDP register to be written to. The routine also requires that the Z-80 register 'C' contain the value that is to be transferred. In assembly-language the set up looks like this:

```
LD B, VDP register
LD C, value to transfer
CALL $FD20
RET
```

In machine code it looks like this:

```
6, 7,
14, value to transfer,
205, 32, 253,
201
```

```
   10 REM instant background color
   20 REM works in TEXT, GR, or HGR mode
   30 REM any graphics mode command will reset
   40 LOMEM :28000
   50 DATA 6,7,14,0,205,32,253,201
   60 FOR x = 27600 TO 27607: READ code: POKE x,code: NEXT
   70 PRINT: PRINT " press CNTL-C to exit.": PRINT: PRINT
   80 INPUT " enter color code (1-15): ";cr%
  100 POKE 27603,cr%: CALL 27600
  110 PRINT: GOTO 70
```

```
   10 REM instant TEXT font color change
   20 REM any graphics mode command will reset
   30 LOMEM :28000
   40 DATA 62,0,17,0,0,33,0,32,205,38,253,201
   50 FOR x = 27608 TO 27619: READ code: POKE x,code: NEXT
  100 HOME: PRINT: PRINT " Press CNTL-C to stop.": PRINT
  110 PRINT: INPUT " enter letter color: ";lc%
  120 PRINT: INPUT " enter screen color: ";sc%
  130 PRINT: PRINT " enter the displacement for the";
  140 INPUT " first set to change: (0-31): ";fi%
  150 PRINT: PRINT " enter the displacement for the";
  160 PRINT " last set to change (";fi%;"-31):";
  170 INPUT " ";lt%
  180 col% = lc%*16+sc%: repeat% = lt%-fi%+1
 1000 POKE 27609,col%: POKE 27611,repeat%
 1010 POKE 27614,fi%: CALL 27608
 1020 GOTO 100
```

```
   10 REM solid colored INVERSE blocks
   20 REM any graphics mode command will reset
   30 LOMEM :28000: POKE 17115,241: TEXT
   40 DATA 62,0,17,1,0,33,0,32,205,38,253,201
   50 FOR x = 27620 TO 27631: READ code
   60 POKE x,code: NEXT
  100 FOR x = 0 TO 15: POKE 27621,x*16+x
  110 POKE 27626,x+16: CALL 27620: NEXT
 1000 FOR x = 0 TO 15: PRINT x;"  ";CHR$(x*8+133): NEXT
```

TITLE (asmb#1): INSTANT BACKGROUND COLOR

```
Decimal
value:                    Op-code:             Comment:
______                    ________             ________

  6,   7,                 LD   B,  $07         ; load VDP register
 14,   0,                 LD   C,  nn          ; load color code value
205,  32, 253,            CALL $FD20           ; CALL OS write to VDP register
201                       RET                  ; RETurn to BASIC
```

TITLE (asmb#2): INSTANT TEXT FONT COLOR CHANGE

```
Decimal
value:                    Op-code:             Comment:
______                    ________             ________

 62,   0,                 LD   A,  nn          ; load color code value
 17,   0,   0,            LD   DE, nnnn        ; load byte count
 33,   0,   0,            LD   HL, nnnn        ; load VRAM start address
205,  38, 253,            CALL $FD26           ; CALL OS write one byte to VRAM
201                       RET                  ; RETurn to BASIC
```

TITLE (asmb#3): SOLID COLORED INVERSE BLOCKS

```
Decimal
value:                    Op-code:             Comment:
______                    ________             ________

 14,  16,                 LD   C,  $10         ; load repeat counter
 62,   0,                 LD   A,  $00         ; restore accumulator to zero
185,                      CP   C               ; check if repeat counter is zero yet
200,                      RET  Z               ; if zero, RETurn to BASIC
 13,                      DEC  C               ; decrement repeat counter by one
121,                      LD   A,  C           ; set original color value = rpt cntr
135,                      ADD  A,  A           ; double color value
135,                      ADD  A,  A           ; double color value
135,                      ADD  A,  A           ; double color value
135,                      ADD  A,  A           ; double color value
129,                      ADD  A,  C           ; obtain final color value (x*16+x)
  6,   0,                 LD   B,  $00         ; set up for 'BC' addend
 33,  16,  32,            LD   HL, $2010       ; establish VRAM base address
  9,                      ADD  HL, BC          ; get VRAM address to put color value in
 17,   1,   0,            LD   DE, $0001       ; load byte count of 'one' for routine
197,                      PUSH BC              ; save repeat counter
205,  38, 253,            CALL $FD26           ; CALL OS write one byte to VRAM
193,                      POP  BC              ; retrieve repeat counter
 24, 229                  JR   $E5             ; jump back to LD   A,  $00
```

Assembly language notes:

→ In assembly-language, hex values are preceded by a dollar sign ($).

→ In the hex system, the high byte value precedes the low byte value.

→ When creating machine code routines, it helps to think in terms of assembly-language mnemonics. Then convert the mnemonic op-code and the operand (value following the op-code) to their machine language values.

→ In assembly-language listings, three variable names may be used as operands (to be replaced with the correct values when the program is ready to run). An 8-bit operand will replace 'nn'. A 16-bit operand will replace 'nnnn'. An 8-bit signed displacement will replace 'dd'.

→ An operand for an assembly-language instruction may consist of a value that is used directly, or it may refer to a location that contains the value. If it refers to a location that contains the value, then the portion of the operand following the comma will be enclosed in parentheses.

Program LISTing explanations:

The first program on page 20 demonstrates how to set up and use the instant background color change routine. This program is an extension of the discussion on VDP registers.

The second BASIC LIST on page 20 demonstates how to use the displacement values from the table on page 18. As you can see, it is possible to have 32 different font colors in BASIC TEXT mode! Displacements 16 through 31 are set up by SmartBASIC to be the inverse characters.

The third BASIC program on page 20 presents an interesting extrapolation from the previos one. It eliminates the INVERSE characters supplanting them with solid colored blocks.

Each of the assembly-language lists on page 21 correspond respectively to the BASIC programs on page 20. The first one is very simple.

The second one uses the VDP routine at 64806. For this routine, register 'A' (called the accumulator) contains the value that is to be transferred to VRAM (the color code value). The register pair 'DE' contains the number of times to transfer the value in 'A'. The register pair 'HL' contains the first VRAM address to transfer the value to. The routine at 64806 POKEs the value in 'A' into the specified VRAM address, then begins a repetitive process. It decrements 'DE' by one and increments 'HL' by one and POKEs the value in 'A' into this next VRAM address. The routine continues in this manner until 'DE' is decremented to zero.

The third assembly-language list on page 21 creates solid colored INVERSE blocks. This 'ml' routine is the exact equivalent of lines 40 through 110 in the third BASIC program on page 20. This correspondence of BASIC to machine language exemplifies three common factors. Machine coding uses many more instructions than its BASIC counterpart. However, the 'ml' routine actually occupies much less RAM. And the 'ml' routine is almost 200 times faster than the same routine in BASIC!

This particular 'ml' routine is much more complex than the other two. With the load (LD) instructions, the value following the comma is stored in the specified register or register pair.

Compare (CP) instructions compare the value in the specified register with the value in the accumulator (A). In BASIC the 'IF . . . THEN' command is used to make decisions. In machine code the CP instuction with a conditional return (RET) or a conditional jump (JP or JR) is used to make decisions (there are also unconditional RETs, JPs, and JRs). The 'F' register (the flags register) contains six flags, ie, bits that are set to '1' when certain conditions exist. These are C (carry), Z (zero), S (sign), P/V (parity/overflow), N (subtract), and H (half carry). The 'C' and 'Z' flags are typically used more frequently by programmers. In a comparison (CP), the 'Z' flag is set if the values are equal and the 'C' flag is set if the accumulator is smaller than the other register.

The Z-80 provides no direct instruction for multiplication or division. These operations are accomplished through repeated addition or subtraction respectively. The third 'ml' routine doubles the accumulator four times, ie, it multiplies the accumulator times two to the forth power (16).

The routine at 64806 uses the 'BC' register pair. Therefore we need to save it on the stack (PUSH) before using the routine and retrieve it when done (POP).

Jumps (JP or JR) are essentially the equivalent of the BASIC GOTO command. Relative jumps (JR) are used so that a routine is independent of its RAM address. This is very convenient when you want to use a particular routine at different locations in various programs. The operand of a relative jump is a signed displacement of a specific number of bytes. Without getting into a complicated explanation of signed displacements at this point, here's a simple way to calculate the displacement value. A positive (or forward) displacement is the actual byte count -- not to exceed 129 bytes. A negative (or backward) displacement can be determined by subtracting the actual byte count (not to exceed 126) from 256. We'll experiment more with displacements, decisions, and jumps in later issues.

WORD POWER:

Here are five words that we encounter everyday in magazines and newspapers. As a matter of fact, every one of these words was used at least once in this very newsletter. See how well you fair. The answers are listed below the quiz.

(1) abstruse:

A. to push forward
B. hard to understand
C. lacking in intellect
D. to refrain voluntarily

(2) concatenate:

A. to speak freely
B. to gain entry to
C. an official agreement
D. to connect in a series

(3) facilitate:

A. to make or manufacture
B. an exact copy
C. to make easy
D. marked by flippant humor

(4) meticulous:

A. precise about details
B. performed in systematic order
C. infested with ticks
D. completely untruthful

(5) simultaneous:

A. occurring at the same place
B. having similarity
C. marked by simplicity
D. occurring at the same time

The answers are: (1) B, (2) D, (3) C, (4) A, and (5) D.

PRODUCT REVIEWS:

Product:	The Coleco ADAM Entertainer
Manufacturer:	Osborne/McGraw-Hill
Media Types:	book
Graphics rating:	97
Value for money:	95
Instructions:	95
Overall rating:	highly recommended
Price:	$12.95
Rated by:	staff

This is an excellent collection of ready to type in SmartBASIC programs. The book includes over two dozen programs which make use of the GR and HGR modes. Many of the programs are games. Each program is fully explained. You can learn a lot about BASIC programming and graphics applications by studying these programs.

Product:	JKL Utilities
Manufacturer:	Overpriced Software
Media Types:	disk or DDP
Graphics rating:	93
Value for money:	75
Instructions:	90
Overall rating:	recommended
Price:	$49.95
Rated by:	staff

This is a very useful assortment of utilities. It is self-booting and written entirely in machine code. This program should be of great benefit to advanced programmers. It includes several functions: media editor, recover deleted files, remove deleted files from directory, copy utility, disassembler, change file attributes, change volume and file names, plus other advanced applications. It also includes a help screen which lists all the controls.

At nearly FIFTY dollars the listed price is very high. However, this is an outstanding package.

Product:	SMART GAMES PACK
Manufacturer:	NIAD
Media Types:	disk or DDP
Graphics rating:	95
Value for money:	98
Instructions:	93
Overall rating:	highly recommended
Price:	$9.95 or $11.95
Rated by:	staff

This is an excellent package of three graphics games: SPACE CHASE, TREASURE SEARCH, and MAZE ESCAPE. The programs are all controlled from a central menu. All inputs are made with the game controllers (supports one or two players). All of the games are nicely done and fun to play.

Product:	PaintMASTER
Manufacturer:	Strategic Software
Media Type:	DDP
Graphics rating:	99
Value for money:	99
Instructions:	95
Overall rating:	HIGHLY RECOMMENDED!!
Price:	$24.95
Rated by:	staff

This is the best graphics design program ever developed for ADAM. With this program you can create, edit, and save hi-res drawings. It also includes a program that will print your creations on the ADAM printer.

You can change paint brushes, paint colors, screen colors, and drawing thickness. You can erase and relocate parts of the drawing. You can also print words inside the graphics window.

The various options are selected with the joystick by moving a pointer to the desired icon (a technique of using pictures rather than words for menu selections). For instance, if you want to access a storage drive, you would move the pointer to the picture of a disk.

Product:	Charts & Graph Assembler
Manufacturer:	Extended Software Company
Media Types:	disk or DDP
Graphics rating:	93
Value for money:	65
Instructions:	93
Overall rating:	not recommended
Price:	$24.95
Rated by:	staff

This is a BASIC program that you can use to create bar graphs, line graphs, and pie charts. You can graph up to ten inputs. You can print bar graphs on the ADAM printer.

This is a well written program and the graphics are nicely done. All inputs must be typed in -- it provides no save/load feature. I feel that at the listed price, most ADAM users would be very disappointed with this software.

Review notes:

Our staff makes every effort to write OBJECTIVE reviews. If you disagree with a particular review, please submit your own.

The greatest compliment that you can give a software developer is to recommend their products to others. When submitting a review, please try to use our format. Also, include your name and subscription ID number (from your mailing label).

HACKER'S CONTEST:

The NIBBLES & BITS Hacker's Contest is a monthly competition. The winner of each contest is randomly selected from the correct responses postmarked within the specified dates. No individual shall be named the winner in three consecutive contests. The winner of each contest shall be awarded ten dollars and a free three month extension to his/her NIBBLES & BITS subscription term. Decisions of the judges are final.

Responses for this month's contest will be considered valid if, and only if, they are postmarked after June 30, 1986 and prior to August 1, 1986. The winner shall be announced in the September issue of NIBBLES & BITS.

Write a SmartBASIC program (it may include machine code in DATA statements), which will do the following on a GR or HGR screen: display the letters 'A', 'B', and 'C' each with different font and screen colors.

SOFTWARE EXCHANGE:

Our first two public domain software packages will be released August 1, 1986. Each PD library contains at least 70K of programs. These first two libraries will be SmartBASIC programs. We may begin PD libraries for other processor programs (CP/M, ADAMCALC, SmartLOGO, etc.) if enough readers request same.

All the programs in our BASIC libraries are speed-LOADed with a machine language central menu. To get a free copy of a specific library: (1) contribute a public domain program (not already in one of our libraries), (2) send a signed statement that the program is public domain and not copyrighted material, (3) send the program on a datapak, (4) request the specific library that you want in return.

Each library is also available on datapak for $7.95.

LOCAL USER GROUPS:

ALABAMA:

James E. Gilbert, 4608 Lakeview Drive, Huntsville, AL 35810

Victor L. Watford, P.O. Box 777, Russellville, AL 35653

ALASKA:

Richard Baines, 7210 Bulen Drive, Anchorage, AK 99507

ARKANSAS:

Danny Levitt, 4525 South White Pine, Tucson, AZ 85730

Robert R. Marentes, 9425 North 38th Avenue, Phoenix, AZ 85021

CALIFORNIA:

Harlod Alexander, 37 Catspaw Cape, Coronado, CA 92118

Sue Askew, 868 North 2nd Street - #242, El Cajon, CA 92021

Frank Fleich, 13381-19 Magnolia Avenue, Corona, CA 91719

George Havach, 550 27th Street - #202, San Francisco, CA 94131

Ann Quetel, 1154 North Mayfield Avenue, San Bernardino, CA 92410

Brian Stranahan, 8580 Buggy Whip Road, Alta Loma, CA 91701

James Turner, Jr., 20110 Avenue 19, Madera, CA 93637

COLORADO:

Jesse Thornhill, II, 1416 Lipan Street, Denver, CO 80204

Intel-BEST 3.3 by DIGITAL EXPRESS, INC.

In the fall of 1985 DATA DOCTOR developed SmartBEST V1.0. SmartBEST adds 27 enhancements to BASIC: sound, graphics, etc. SmartBEST was DATA DOCTOR's fastest selling software. It is a great program!

Having purchased the copyrights to all DATA DOCTOR software, DIGITAL EXPRESS, INC. has improved the program dramatically. Intel-BEST 3.3 makes over three dozen changes to SmartBASIC.

Nearly all BASICs permit the question mark as a shortcut for the PRINT command. Intel-BEST 3.3 includes five similar shortcuts: 'F' for FLASH, 'H' for HOME, 'I' for INVERSE, 'N' for NORMAL, and 'T' for TEXT. With these added shortcuts, it's a lot easier and faster to enter BASIC programs.

Intel-BEST even corrects the DATA and REM spacebump bug!!! It also corrects the COLOR, HCOLOR, and SCRN color tables so that you only have to use ONE color code chart. Intel-BEST permits you to enter up to 216 characters per program line (BASIC limits input to 128). Intel-BEST eliminates many of the unnecessary spaces added with the LIST command. And Intel-BEST resets the POKE limit to 65535.

Intel-BEST also includes several ready-to-use machine code routines. Included are a block read and a block write routine. Also included is a routine that will read the catalog (block one) without interrupting a RUNning program. Routines are also included that will instantly change the background color and the NORMAL and INVERSE colors without clearing the screen. Intel-BEST even includes a routine that will instantly change the graphics window color in GR or HGR mode without clearing the screen.

Intel-BEST includes a relative RESTORE command (LINE) that will let you RESTORE to any line number that you specify. Using this command gives you powerful control over your DATA. Also included is a new command (Ln8) that will increase the TEXT window in either graphics mode to eight lines instead of four.

Intel-BEST 3.3 also includes some powerful audio enhancements. ADAM will permit four simultaneous sounds (three voices and a noise). Intel-BEST makes it very easy for you to get the maximum benefit from ADAM's sound capabilities. With each voice you can use any of 1024 possible sounds. The noise generator is equipped with 8 built-in sound effects. Of the 1024 sounds possible with each voice only 48 correspond exactly to musical notes. Intel-BEST provides you with direct access to these musical notes so that you can easily enter music from songsheets. The extensive instruction manual fully explains how to enter these notes and how to create special noises and sound effects. To enter a 'C' note in the lowest octave (of the four standard octaves) in the first voice all you need to do is: T1 = 3 [RETURN]. Changing the volume is equally simple; just enter the volume setting (0 through 15). Intel-BEST includes nine easy-to-use music commands.

Intel-BEST 3.3 is the most elaborate enhancement to SmartBASIC ever developed for ADAM and it's also the least expensive. Intel-BEST will also RUN just about ALL SmartBASIC programs. Once you've tried Intel-BEST 3.3, you'll NEVER want to go back to SmartBASIC V1.0. Intel-BEST is a pure machine code program (1742 bytes) and it sets LOMEM to 27600.

ADDED BONUS: As a special introductory offer, we'll give you a 25% discount off Intel-BEST 3.3 as a NIBBLES & BITS subscriber provided your order is postmarked prior to September 1, 1986. Get Intel-BEST 3.3 NOW and SAVE!!!

DEI PRODUCT LIST:

Software:

Intel-BEST 3.3 (dynamic enhancement to SmartBASIC V1.0)
$18.95 (introductory special: $14.21) -- datapak only

Intel-LOAD (converts BASIC programs to LOAD up to 12 times faster)
$11.95 -- datapak only

HARDWARE:

DEI datapaks (DEI manufactured datapaks are as reliable as Coleco's datapaks)
$2.75 each
$24.95 (for ten)

ACCESSORIES:

adhesive labels (tractor-feed, fan-fold, 3 1/2 x 15/16)
$2.50 (500 labels)
$4.75 (1000 labels)

Z-80 instuction lists (over 700 instructions, decimal, hex, op-codes, and operands)
$1.00 (numerical -- SASE: no shipping charge)
$1.00 (alphabetical -- SASE: no shipping charge)

blank white paper (tractor-feed, fan-fold, 9 1/2 x 11, 20# wt., 250 sheets)
$5.95

NIBBLES & BITS SUBSCRIPTIONS:

$18.00 (one year -- 12 issues)
$12.00 (six months -- 6 issues)

SPECIAL NOTICE: All DEI datapaks are warrantied to be free from defects in material and workmanship. If the storage media proves defective, return it to DEI for a replacement. This warranty applies to datapaks purchased 'blank' and to those purchased with DEI program(s) stored on them. DEI shall, under no circumstances, be liable for consequential damages, if any.

PRODUCT ORDER FORM:

YOUR NAME: ______________________________

ADDRESS: ______________________________

CITY: ______________ STATE: __ ZIP: ________

PHONE NUMBER: ______________________________

SUBSCRIPTION ID NUMBER: ______________________

< ITEM/QUANTITY/MEDIA >	< PRICE >
<______________________________>	$<____.__>
<______________________________>	$<____.__>
<______________________________>	$<____.__>
<______________________________>	$<____.__>
<______________________________>	$<____.__>

send check or money order to:

DIGITAL EXPRESS, INC.
1203 Northwoods Drive
Kings Mtn., NC 28086

SUBTOTAL:	$____.__
SHIPPING:	2.50
TAX:	____.__ (NC residents only -- 4.5%)
OTHER:	____.__
TOTAL:	$____.__

SWIFT POLL BALLOT:

As a NIBBLES & BITS subscriber, you are invited to submit one SWIFT POLL ballot per month. This ballot must be cut out (not duplicated) and postmarked prior to August 1, 1986 in order to be verified as valid. The results for the current poll shall be tallied and made public in the October issue of NIBBLES & BITS. Please list your top ten software preferences for the month of JULY, 1986 in the order that you like them (best first, next best second, etc.).

YOUR NAME:__________________ SUBSCRIPTION ID NUMBER:__________________

1. ____________________	2. ____________________
3. ____________________	4. ____________________
5. ____________________	6. ____________________
7. ____________________	8. ____________________
9. ____________________	10. ____________________