The Adam Technical Journal

BI MONTHLY NEWSLETTER

VOL 1 - NO. 1 2/85

CREATED FOR COLECO ADAM OWNERS

A TECHNICAL NEWSLETTER ON HOW TO OUT SMART "SmartBASIC"

SERENDIPITY PRODUCTIONS
P.O. BOX 07592
MILWAUKEE, WI 53207

INTRODUCTION

This is the first of a series of newsletters that discusses the capabilities of ADAM SmartBASIC. The first few newsletters will concentrate on the Video Display Processor. We will then continue into the SmartBASIC Interpreter and Operating System.

The Video Display Processor (VDP) is one of three microprocessors in the ADAM. The main processor is the Z80 CPU (Central Processing Unit). The Operating System and SmartBASIC are written in the Z80's assembly language. The Z80 controls the Inputs and Outputs (1/0) that control the TI SN76489AN Sound Processor and the TI TMS9918A Video Display Processor. For this reason the programs that we will discuss will be partially written in the Z80 Assembly Language.

The Video Display Processor controls all of the video screen displays. This includes:

1) How the characters to be displayed are defined
2) The color of the foreground and background of different sets of characters
3) The display or retrieval of a character an the screen
4) The definition and display of sprites
5) The ability to go into high resolution graphics.

In this newsletter, we will concentrate on displaying and retrieving characters from the screen. We will also discuss how you can define your own character sets and change the screen and character colors.

VDP MEMORY

The ADAM Computer has 80K (81920 bytes) of Random Access Memory (RAM). Only 64K of this memory is directly accessible by the main Z80 CPU. The remaining 16K is an indirect memory and can only be accessed through the VDP. This 16K of memory, which we will call VDP memory, contains a collection of tables used by the hardware which operates the display screen. For example, there are two segments of 768 bytes of memory used for the Screen Image Table. The Screen Image table holds the ASCII codes of text presently being displayed on the screen. Another example is the Character Definition Table. This table uses 2048 bytes of memory which describes what each displayable text character looks like. When you want to display the letter "A" on the screen, the ASCII code for "A" is placed in the first table. The hardware sees that code, looks it up in the character definition table to see what it looks like, then paints it on the screen.

The VDP is accessed through two I/O PORTS: PORT 191,190. The first port (191) is used to tell the VDP what to do with the data. Port 190 is used to send or receive the actual data. There are 4 basic operations that the VDP can perform.

1) Write to VDP memory
2) Read from VDP memory
3) Write to VDP write-only registers
4) Read the VDP status register

The first two operations are your only means of access to the 16K VDP memory. The third operation allows you to change the 8 internal registers of the VDP processor. The appropriate values in these registers will allow you to define where the different tables will reside. They also give the capability of changing the mode of the display. These modes are:

1) Hi-Res (each point, or pixel, an the screen is definable)
2) TEXT (gives you 40 columns of text displayed)
3) Graphics (gives you 32 columns, and is the mode the ADAM is normally in)
4) Multi-Color (each character appears as four color definable squares)

We will discuss these modes and how to obtain them in future newsletters.

As mentioned earlier the main topics of this issue are Character definition, Color code definitions, screen color and reading and writing to the screen. These issues will be discussed in the context of the 32 column by 24 row Graphics mode.

POKING AND PEEKING VDP MEMORY

In order to tap the exceptional power of this Video Display Processor, we must be able to modify VDP memory. Since the BASIC PEEK and POKE commands cannot reach VDP memory, we will have to use two small assembly language programs. Assembly listings (#2 and #3) for these programs are given at the end of this newsletter. They are for your information only since the actual routine is built for you by the BASIC Utility program (listing #1). Both programs listed at the end of this newsletter use port #191 to set up the address to poke into and port #190 to send the value you want poked into VDP memory.

The POKEVM program first saves the accumulator, status flags and register pair BC on the stack.

The low address byte is sent to PORT 191 first. The high byte must be offset by a value of 64 before it is sent to PORT 191. The offset indicates to the VDP which of the 4 operations described above are to be performed. A write to VDP memory requires an offset of 64, writing to a register requires a 128 and a read from memory requires no offset. Reading the status register is a slightly different operation and is not very useful until we get into some rather sophisticated graphics. The offset is added on by line 52000 of the BASIC program. The series of six "EX (SP),HL" instructions are used as a delay to give the processor time to set the address. This in instruction was chosen because it is one of the most time consuming, which is the effect we're after. The value you wish to transfer to VDP memory is then sent through Port 190. After each write to VDP memory the VDP's internal address buffer increments itself automatically.

Therefore, if the count-down register (BC) has not reached zero the program will loop through without having to tell the VDP what address to write to every time. After the count down loop has completed, the accumulator, status and BC registers are POPped back and the program returns to the calling BASIC program.

The BASIC subroutine at line 52000 does nothing more than take the destination address and character count, then decomposes them into 2-byte values. It then adds 64 onto the high byte of the address and pokes the four bytes into the POKEVM assembly language routine. lastly, the assembly subroutine is CALLed and the BASIC subroutine RETURNS to the main program.

The PEEK subroutine is similar to the POKE with the following exceptions. The count-down loop is no longer needed since we are only going to read one memory location. The high byte of the VDP address no longer needs the offset (actually, the offset is zero as discussed above). The OUT 191 is replaced by an IN 191. After the IN instruction, the accumulator is transferred into a CPU address. Since only one address is to be read the PUSH BC and POP BC instructions are also not present in the-PEEKV program.

The PEEK BASIC subroutine decomposes the address into two bytes the same way as the POKE subroutine. These two bytes are passed down into the PEEK VDP assembly routine. After the routine is executed, the contents of the VDP memory location will have been copied into the CPU memory 10 bytes beyond the PEEK VDP assembly routine. This CPU address is then peeked by the BASIC subroutine and stored in a variable.

WHERE TO PUT THE ASSEMBLY LANGUAGE

The assembly language is loaded into memory using subroutine starting at line #51000. This routine PEEKs at locations 16102 and 16101. These locations contain the high and low bytes of the address where the largest line number is stored in the line number table. Two locations below this address is the high and low bytes of the address where the tokenized code of the last line is stored. This address with the offset of 10 is where we are going to store the assembly language. Therefore to create a buffer in your basic program to store your assembly language you need a non-executable instruction, such an instruction is the REM with as many characters in it as needed for the buffer. This REM statement must be located with the largest line number in your program - Storing the assembly routine inside a REM statement of a basic program gains several advantages. one advantage is that assembly language becomes an integral part of your program. It can be saved and loaded onto a data pack right along with your BASIC program. Other reasons will become apparent in future issues. one of these topics will include a fast loader and saver that will decrease the time of saving and loading to about an eighth of the normal time. This topic will be in the issue that covers the topic of how the BASIC interpreter tokenizes code.

SCREEN DEFINITION TABLES

This contains the ASCII codes of what is presently appearing on the screen. Therefore, if you put, for example, a 65 into this table you see the letter "All appear on the screen.

There are actually two screen Image Tables, the operating system alternates these tables in order to cause a cursor to blink. If the ASCII 65 was only Put into one of these tables the screen display would alternate between what was originally there and the "A". To make the screen appear stable you must poke the same ASCII value into both tables.

Smart Basic has defined these tables in the VDP memory to start at locations 2048 and 6144. Each table contains 768 bytes. This corresponds to 24 rows times 32 columns. The memory locations that correspond to each row are as follows.

ROW	TABLE(1)	TABLE(2)
1	2048-2079	6144-61715
2	2080-2111	6176-6207
3	2112-2143	6208-6239
4	2144-2175	6240-6271
5	2176-2207	6272-6303
6	2208-2239	6204-6335
7	2240-2271	6336-6367
a	2272-2303	6368-6399
9	2304-2335	6400-6431
10	2336-2367	6432-6463
11	2368-2399	6464-6495
12	2400-2431	6496-6527
13	2432-2463	6528-6559
14	2464-2495	6560-6591

15	2496-2527	6592-6623
16	2528-2559	6624-6655
17	2560-2591	6656-6687
18	2592-2623	6688-6719
19	2624-2655	6720-6751
20	2656-2687	6752-6783
21	2688-2719	6784-6815
22	2720-2751	6816-6847
23	2752-2783	6848-6879
24	2784-2815	6880-6911

The basic "WRITE-TO SCREEN" subroutine starting at line 53000 calculates the addresses for you, given the row and column, and then executes the POKEMV routine.

The "READ-FROM-SCREEN" subroutine starting at line 56000 calculates an address only for Table (2) given the row and column ocations. The reason for PEEKing from only one table is what we had mentioned earlier. The two tables are duplicates of each other except for what is under the cursor.

CHARACTER DEFINITION TABLE

As we mentioned above in the VDP MEMORY section, the character definition table describes the appearance of each displayable character. This table consists of an 8 byte binary "picture" corresponding to each of the ASCII codes. The format of that picture will be described below and we will actually change the appearance of the character of your choice.

The character definition table starts in VDP memory at location 0 and goes to location 2048 allowing a maximum of 256 character definitions. To calculate which 8 bytes control the definition of a particular ASCII code you would multiply the ASCII code by 8. This will give you the beginning memory locations of the 8 consecutive memory locations that define the character.

A character is made up of an eight by eight grid. Each row of this grid, starting from the top, must be converted into a byte value to be poked into the a consecutive memory locations of the character you are redefining.

we will first explain how to convert the row into a byte value which ranges from a value of 0 to 255. Eight of these bytes that represent each raw could then be transferred directly into the table. Secondly, we will look at a shorthand way of representing the 8 bytes. This shorthand method is used in the BASIC subroutine which will be discussed later.

If you look at Fig. 1, you will see a grid of 8 rows by 8 columns. Each raw must be converted into a single byte. You will notice the columns are numbered left to right 128, 64, 32, 16, 8, 4, 2, 1. To obtain the byte value of the row, first check off the squares in the 8x8 grid that will make the grid appear as the object you wish. For each row, add up the values that appear at the top of the columns for the squares that are checked off. When you have completed all the rows, you will have the bytes necessary to poke into the table to redefine the character.

As an example, we will calculate the memory locations for the character "A" and determine the bytes to be poked into these locations to make it look like a jet.

To calculate the memory location simply multiply 8 times the ASCII code "A". This is: 8x65=520. Therefore, locations 520 through 527 hold the definition for the letter "A".

The values to he POKED into these locations are calculated and appear at the bottom of the grid in fig 2.

(GRAPHIC NOT INCLUDED)

ROW 1: 8 = 8
ROW 2: 16+8+1 = 25
ROW 3: 32+16+8+2+1 = 59
ROW 4: 128+64+32+16+8+4+2+1 = 255
ROW 5: 32+16+8+2+1 = 59
ROW 6: 16+8+1 = 25
ROW 7: 8 = 8
ROW 8: 0 = 0

Fig. 2

If you were to use the POKEVM subroutine that we discussed earlier to poke the calculated values into locations 520 through 527, all A's from that point on would appear on the screen as jets.

THE SHORTHAND METHOD

The shorthand method is used by the BASIC subroutine discussed later to pass a new character definition to the VDP by use of a single string variable. This is desirable for two reasons. First, it allows you to use one variable to describe the character rather than 8, a considerable savings in a large program. Second, it breaks up the character into twice as many pieces making it easier to modify later on.

The method involves decomposing the 8x8 grid into a 16 character string. This is done by separating each row into two 4 column rows (see Fig. 3). The 4 columns for each raw are converted into a hexadecimal value. Hexadecimal is a number system that is base 16 and the digits in the set are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Figure 4 lists the hexadecimal value for its particular 4 column pattern.

(GRAPHIC NOT INCLUDED)

To illustrate how a 16 character string can represent a character Pattern, we obtain the definition of the jet of Fig. 2. If you look at Fig.5 you will see that the 8x8 grid has been separated into two grids of 4 columns by 16 rows. Each 4 column row is then compared to Fig. 4 to obtain the corresponding pattern number. The pattern numbers are then combined into a single 16 character string.

(GRAPHIC NOT INCLUDED)

COLOR DEFINITION

The Color Definition Table contains the code which defines the foreground and background colors of the 32 character sets. The table starts at VDP memory location 8192 and contains 32 memory locations. Each byte defines one subset of a characters. For example, character set #10 would contain the ASCII codes 72-79 which are (H,I,J,K,L,M,N,O). The following table gives the character set # and VDP memory location that controls the colors of the characters in that set.

TABLE 2

SET LOCATION	VDP	ASCII CODES
1	8192	0- 7
2	8193	8-15
3	8194	16- 23
4	8195	24- 31
5	8196	32- 39
6	8197	40- 47
7	8198	48- 55
8	8199	56-63
9	8200	64- 71
10	8101	72- 79
11	8202	80- 87
12	8203	88- 95
13	8204	96-103
14	8205	104-111
15	8206	112-119
16	8207	120-127
17	8208	128-135
is	8209	136-143
19	8210	144-151
20	8211	152-159
21	8212	160-167
22	8213	168-175
23	8214	176-183
24	8215	184-191
25	8216	192-199
26	8217	200-207
27	8218	208-215
28	8219	216-223
29	8220	224-231

30	8221	232-239
31	8222	240-247
32	8223	248-255

The character set number will be used in the Basic subroutine starting at line 55000 to identify which set you would like to chance. The value you POKE into the Color Definition Table is listed in Table 3.

The chosen foreground and background color must be combined into a single byte to be poked into the appropriate character set memory location. This is accomplished by multiplying the foreground color by 16 and adding the background color code onto it. The BASIC subroutine at line 55000 does this for you. The background is normally transparent for all the character sets. This allows the screen color to be seen behind the characters.

TABLE 3

COLOR	CODE NUMBER
Transparent	0
Black	1
Med. Green	2
Light Green	3
Dark Blue	4
Light Blue	5
Dark Red	6
Cyan	7
Medium Red	8
Light Red	9
Dark Yellow	10
Light Yellow	11
Dark Green	12
Magenta	13
Gray	14
White	15

SCREEN COLOR

The screen color is the background color you would see behind the characters that have been defined with a transparent foreground or background color. The color code number used is the same as defined in Table 3.

Changing the screen color involves changing the value in VDP register #7. We will discuss how to calculate the value that must be poked to change VDP registers in a future issue. However, we have

included a BASIC subroutine starting at line 57000 to alter the Screen colors.

THE BASIC UTILITIES

As you can see, the utilities are a set of basic subroutines. By simply setting some reserved variables to the parameters you desire and doing a GOSUB to the particular subroutine, you will perform a graphics function.

The idea is to include the utilities in a BASIC program you create. This can be done in one of two ways. You can either LOAD the utility file from tape and key in the program above it or you can use SmartWRITER to append the utility file to an existing program.

Your program should not use any variables that start with the letter "u" other than for passing utility parameters. Your program must reside above the utilities, that is, have lower line numbers than the utilities. The following section describes how to use each utility.

SET UP UTILITY (GOSUB 51000)

This subroutine sets up the assembly language needed for all the other subroutines. Therefore GOSUB 51000 only has to be executed once at the beginning of your program. If you intend to list the entire program on the printer or transmit it aver a modem, it is advisable to re-enter the REM statement at line 65535 with a string of at least 60 characters. Since the remark statement contains a machine language program after performing a GOSUB 51000 and not printable characters, it may cause problems on certain rare occasions. If a data value poked into the routine corresponds to a screen or printer control byte, it will be executed. For example a value of 16 corresponds to a control-P which causes the contents of the screen to be sent to the printer.

Also, if you are repeatedly saving a program to tape (i.e. developing a new or large program) it is a good idea to re-enter the line anyway since SmartBASIC always inserts an extra character into the beginning of all REM and DATA statements when writing it to tape. Because of this you must GOSUB 51000 after every LOAD.

FORMAT:
10 GOSUB 51000

POKE VDP UTILITY (GOSUB 52000)

This utility will let you poke a value into consecutive VDP memory locations. The variables (ua,ub, and uc) are set by you to the starting address in VDP memory, the value to be poked and the number of copies to be poked. By doing a GOSUB 52000 the poking will be performed.

FORMAT:

10 ua=2048: ub=65: uc=5: GOSUB 52000

WRITING TO THE SCREEN (GOSUB 53000)

Placing a character or a series of the same character on the screen is performed by this subroutine. The variable (ur) is used to set the desired row. Variable (ul) contains the starting column location on the screen. Variable (ub) is used to hold the ASCII code to be displayed on the screen. The last variable you must define is (uc) which contains the number of consecutive copies you will have displayed on the screen. Finally, performing a GOSUB 53000 will put the character on the screen.

FORMAT:

10 ur=1: ul=1: ub=66: uc=5: GOSUB 53000

CHARACTER DEFINITION UTILITY (GOSUB 54000)

This subroutine will allow you to change the appearance of any character. All that must be done is to set the variables
(us) to the ASCII code of the character you wish to change and (us$) to the 16 character string that defines the pattern, as discussed in the shorthand method. Lastly, performing a GOSUB 54000 will change the appearances of the character.

FORMAT:

10 us=65: us$="08l93BFF3Bl90800": GOSUB 54000

CHARACTER SET COLOR UTILITY (GOSUB 55000)

Changing the foreground and the background colors of the character sets of table 2 is performed by this subroutine. If you set variable (us) to the character set code (table 1), and set (uf) and (ub) to the foreground and background color codes found in table 2, and lastly execute a GOSUB 55000 the characters in that set will change to the corresponding colors.

FORMAT:
10 us=9: uf=1: ub=15: GOSUB 55000

PEEK VDP MEMORY UTILITY (GOSUB 56000)

This utility will allow you to peek at any VDP memory location. If you set (ua) to the address you wish to peek and perform a GOSUB 56000 you Will have the contents of that address returned to you in the variable (ub).

FORMAT:

10 ua=2144: GOSUB 56000: va=ub

SCREEN COLOR UTILITY (GOSUB 57000)

This utility will let you change the background color of the screen. By setting (us) to the desired color code number found in table 3 and performing a GOSUB 57000 you will set the screen to the corresponding color.

FORMAT:
10 us=5: GOSUB 57000

READING FROM THE SCREEN (GOSUB 58000)

This subroutine allows you to read a character off the screen at any location. Variable (ur) is used to hold the row location and varialble (ul) is set to contain the column location on the screen you wish to obtain. After you execute GOSUB 58000 the variable (ub) will contain the ASCII code of that location.

FORMAT: 10 ur=10: ul=20: GOSUB 58000: va=ub

UNSCRAMBLING A MESS !!!

Probably the most important command in SmartBASIC is "TEXT". This command will undo anything you have done to the VDP memory and allows you to start with a clean slate. There will be times when the screen is totally blank, giving you the impression that the VDP hung up. If you have been accessing the VDP, its worth trying "TEXT" even if you don't think you did anything wrong. It resets all the table locations and content, including the character definition table.

BASIC UTILITES

This utility package should be included below all the example programs to follow. There are a large number of REM statements in the following program. If you would like to save yourself time and CPU memory you can leave them out as you type in the programs

Make certain the last REM statement in the utilities is not removed.

Listing #1

```
50990 REM Set up assembly routine
50995 REM Data for loading down machine code for POKEVM See: Assembly listing

51000 DATA 245, 197, 1, 0, 0, 62, 0, 211, 191, 62, 0, 211, 191, 227, 227, 227, 227, 227, 227, 62,
```

```
0, 211, 190, 11, 120, 177, 32, 247, 193, 241, 201
51005 REM Machine code for PEEPV
51010 DATA 245, 62, 0, 211, 191, 62, 0, 211, 191, 227, 227, 227, 227, 0, 0, 219, 190, 50, 0, 0, 241,
201, 0, 0, 0, 0, 0, 0, 0, 0
51015 REM Get address of last line number
51100 ux = PEEK(16102)*256+PEEK(16101)-2
51150 REM Get address of last line of tokenized code
51200 ux = PEEK(ux)+PEEK(ux+1)*256-10
51250 REM Load down machine code
51300 FOR ui = ux TO ux-56: READ ua: POKE ui, ua: NEXT ui
51350 REM Calculate and POKE down address for LD $0000,A of PEEKV assembly routine
51400 ut = ux-55: POKE ux-50, INT(ut/256): POKE ux+49, ut-INT(ut/256)*256
51500 RETURN
51990 REM POKEVM routine ua = address ub = data uc = number of copies
51995 REM Calculate high and low byte of address and POKE down data into POKEVM assembly
routine
52000 uw = INT(ua/256): uy = ua-uw*256: uw =uw+64: POKE ux+6, uy: POKE ux-10, uw: POKE
ux-20, ub: POKE ux-4, INT(uc/256)
52010 REM Set timing for VDP access
52020 POKE 17009,0
52100 POKE ux-3, uc-INT(uc/256)*256: CALL ux
52200 RETURN
52990 REM Screen write ur = row ul = column ub = data uc = number of copies
52995 REM Calculate VDP address for both screen tables
53000 us = ((ur-l)*32+ul-l)+2048: GOSUB 52000: ua = ua+4096: GOSUB 52000: RETURN
53990 REM Define character us=ASCII code us$=character definition shorthand code
54000 TDR ui = 1 TO 16 STEP 2
54005 REM Decompose shorthand string into 8 bytes and POKEVM into character table
54010 ul = ASC(MID$(us$, ui, 1))
54020 u2 = ASC(MID$(us$, ui+1, 1))
54030 IF ul < 60 THEN ul = ul - 48: GOTO 54050
54040 ul = ul-55
54050 IF u2 < 60 THEN u2 = u2-48: GOTO 54070
54060 u2 = u2-55
54070 ub = ul*16+u2: ua = (ui+1)/2+us*8-1: uc = 1: GOSUB 52000
54080 NEXT ui
54090 RETURN
54990 REM Color definition us = character set # uf = foreground color ub = background color
55000 ua = us+8191: ub = uf*16+ub: uc = 1: GOSUB 52000: RETURN
55990 REM PEEKV ua = address ub = incoming data
55995 REM Set timing for VDP access
56000 POKE 17009,0
56095 REM Decompose address into two bytes and PORE down into PEEKV assembly language
56100 uw = INT(ua/256): uy = ua-uw*256: POKE ux+33, uy: POKE uX+37, uw: CALL ux+31:
ub = PEEK(ux+55): RETURN
56990 REM Change screen color uc = color code
```

```
57000 ua = 18176+us: POKE ux+16, 24: POKE ux+17, 10: GOSUB 52000: POKE ux+16, 227:
POKE ux+17, 227: RETURN
57990 REM Character read from screen ur = row ul = column ub = data
58000 ua = ((ur-1)*32+ul-1)+6l44: GOSUB 56000: RETURN
65535 REM THIS IS A BUFFER FOR THE ASSEMBLY LANGUAGE ROUTINE
```

EXAMPLE PROGRAMS

This example demonstrates the POKEVM and PEEKV subroutines. The program changes the definition of characters 32-255 so that the characters appear upside down.

```
90 REM set up assembly language
100 GOSUB 51000
105 REM Loops through the character table starting with ASCII (32) times 8 which equals 256.
110 FOR I + 256 TO 2048 STEP 8
115 co = 0
116 REM Reads the 8 bytes that define the character in VDP memory.
120 FOR m = I TO i+7
130 ua = m: GOSUB 56000: te(co) = ub
150 co = co+1
160 NEXT m
170 co=0
175 REM Invert and POKEV the character definition back to VDP memory.
180 FOR m = I+7 TO I STEP -1
190 us = m: ub = te(co): uc = 1: GOSUB 52000
200 co = co+1
210 NEXT m
220 NEXT i
230 END
```

PUT UTILITIES HERE....

This example program illustrates how to define a character. This program changes the character "A" to look like a jet.

```
90 REM Set up assembly language
100 GOSUB 51000
110 us = 65: us$ = "08l93BFF3B190800":    GOSUB 54000
120 PRINT "A A A A A A A A A A A A A A"
140 END
```

PUT UTILITIES HERE....

This example program demonstrates several of the VDP utilities. It demonstrates redefining characters and placing that character on the screen. The program will create a little man and cause him to dance back and forth across the screen.

```
90 REM Set up assembly routine
100 GOSUB 51000
105 REM Shorthand character definition code for little man
110 a$ = "1898FF3D3CE40400"
120 b$ = "1819FFBC3C272000"
125 REM Redefine characters 0 and 1
130 us = 0: us$ = a$: GOSUB 54000
140 us = 1: us$ = b$: GOSUB 54000
145 REM Use screen write subroutine to clear screen
150 ur = 1: ul = 1: ub = 32: uc = 768: GOSUB 53000
160 b = 2: c = 31: e = 1: fl = 0
165 REM Loop to locate man on screen
170 FOR I = b TO c STEP e
175 REM Alternate between character 0 and 1
180 IF fl = 0 THEN fl = 1: GOTO, 210
190 fl = 0
200 REM Erase character behind little man. man is in row 10
210 ur = 10: ul = i-e: ub w 32: uc = 1: GOSUB 53000
212 REM Put man on screen in row 10
215 ur = 10: ul = i: ub = fl: uc = 1: GOSUB 53000
220 NEXT i
225 REM Mange directions
230 IF c = 31 THEN b = 31: c = 2: e = 1: GOTO 170
240 GOTO 160
250 END
```

PUT UTILITIES HERE.

This program demonstrates how to read characters from the screen. The program reads the screen into an array buffer and than prints the screen in reverse order.

```
90 REM List some TEXT on the screen
100 LIST 100-400
105 REM Set up assembly language
110 GOSUB 51000
120 DIM b$(24)
125 REM Read screen and put into array buffer
130 FOR ro = 24 TO 1 STEP -1
140 FOR co = 2 TO 31
150 ur = ro: ul = co: GOSUB 58000
160 b$(ro) = b$(ro)*CHR$(ub)
```

```
170 NEXT co
180 NEXT ro
185 REM Print the buffers back to the screen in reverse order
190 FOR ro = 24 TO 1 STEP -1
200 PRINT b$(ro)
210 b$(ro) = ""
220 NEXT ro
230 END
```

PUT UTILITIES HERE

This program flashes all the different color screens.

```
90 REM get up the assembly routine
100 GOSUB 51000
105 REM Loop through the different colors
110 FOR I = 0 TO 15
120 uc = i: GOSUB 57000
125 REM Delay loop
130 FOR i = 0 TO 500: NEXT i
140 NEXT i
150 GOTO 110
```

PUT UTILITIES HERE.....

The following program helps you obtain the shorthand code for defining your own characters.

When you RUN this program an 8 x 8 grid will be displayed for you. By pressing the "1" key a black square will replace the cursor and the cursor will move to the next position. Pressing the key "0" will leave the square blank. After the last square is defined the shorthand code string will be displayed along the real size character by itself and in a 3 x 3 square. If you make a mistake Before you define the last square you can use the right and left arrow key to position yourself to make the correction.

After the character code is displayed you have a choice to quit or to define another character. if you choose to continue the scram will be cleared and a new 8 x a grid will be displayed. The image of the last 8 x I grid can be recovered by Just using the arrow key to pass the cursor over the grid. Because of this if you have made an error an the previous character grid it can easily be corrected an the next grid.

This program was modified from a program originally written for the TI 99/4(A) home computer The TI 99/4(A) and the ADAM computers both have the same VDP processor. with the utilities we have provided in this newsletter and utilities of future newsletters you can modify many of the TI 99/4(A) programs to run on the ADAM system.

```
90 GOSUB 51000
95 REM Change color of screen
100 uc = 4: GOSUB 57000
103 REM Redefine characters 30, 100, 101
105 us = 30: Us$ = "FF81818181818181FF": GOSUB 54000
110 DIM b(8, 8)
120 us = 100: us$ = "000000000000000000": GOSUB 54000
130 us = 101: us$ ="FFFFFFFFFFFFFFFFFF": GOSUB 54000
135 REM Change color of set 13, 4, 16
140 us = 13: uf = 1: ub = 15: GOSUB 55000
145 us = 4: uf = 15: ub = 9: GOSUB 55000
147 us = 16: uf = 4: ub = 4: GOSUB 55000
148 REM Clear screen
150 ur = 1: ul = 1: ub = 32: uc = 768: GOSUB 53000
155 REM Create screen
160 m$ = "AUTO CHARACTER DEFINITION"
170 y = 3
180 x = 4
190 GOSUB 770
200 m$ = "12345678"
210 y = 8
220 GOSUB 770
230 GOSUS 820
240 m$ = "O=OFF=WHITE"
250 y = 22
260 x = 4
270 GOSUB 770
280 m$ = "1=ON=BLACK"
290 y = 23
300 GOSUB 770
310 FOR r = 1 TO 8
320 ur = 8+r: ul = 5: ub = 100: uc = 8: GOSUB 53000
330 NEXT r
335 REM Cursor control on 8 x 8 grid
340 FOR r = 1 TO 8
330 FOR c = 1 TO 8
360 ur = 8+r: ul = 4+c: ub = 30: uc = 1: GOSUB 53000
370 GET ka$: ke = ASC(ka$)
380 IF ke = 163 OR ke = 161 THEN 400
390 GOTO 420
400 GOSUB 870
410 GOTO 360
420 ke = ke-48
430 If (ke < 0) OR (ke > 1) THEN 370
440 b(r, c) = ke
450 ur = 8+r: ul = 4+c: ub = 100*ka: uc = 1: GOSUB 53000
```

```
460 NEXT c
470 NEXT r
475 REM Create shorthand definition
480 he$ = "0123456789ABCDEF"
490 m$ = ""
500 FOR r = 1 TO 8
510 lo = b(r, 5)*8+b(r, 6)*4+b(r, 7)*2+b(r, 8)+l
520 hi = b(r, 1)*8+b(r, 2)*4+b(r, 3)*2+b(r, 4)+1
530 m$ = m$+MID$(he$, hi, 1)+MID$ (he$, lo, 1).
540 NEXT r
545 REM Display new characters and shorthand code
550 us = 102: us$ = m$: GOSUB 54000
560 ur = 8: ul = 20: ub = 102: uc = 1: GOSUB 53000
570 FOR r = 0 TO 2
580 ur = 12+r: ul = 20: ub = 102: uc = 3: GOSUB 53000
590 NEXT R
600 y = 16
610 x = 13
620 GOSUB 770
630 m$ = "PRESS q To QUIT"
640 y = 18
650 x = 12
660 GOSUB 770
670 m$ = "PRESS ANY OTHER"
680 y = 19
690 GOSUB 770
700 m$= "KEY TO CONTINUE"
710 y = 20
720 GOSUB 770
730 GET ka$: ke = ASC(ke$)
730 IF ke <> 113 THEN 140
760 TEXT: END
765 REM Horizontal string display routine
770 FOR i = 1 to LEN(m$)
780 co = ASC(MID$(m$, I 1))
790 ur = y: ul = x+i: ub = co: uc = 1: GOSUB 53000
800 NEXT I
810 RETURN
815 REM Vertical string display routine
820 FOR i = 1 To LEN(m$)
830 co = ASC(MID$(m$, i, 1))
840 ur = y+i: ul = x: ub = co: uc = 1: GOSUB 53000
830 NEXT i
860 RETURN
865 REM Arrow keys routine
870 ur = 8+r: ul = 4+c: ub = 100-b(r, c): uc = 1: GOSUB 53000
```

```
880 IF ke = 161 THEN 960
890 c = c-1
900 IF c <> 0 then 1020
910 c = 8
920 r = r-1
930 If r <> 0 then 1020
940 r = 8
950 GOTO 1020
960 c = c+1
970 IF c <> 9 then 1020
980 c = 1
990 r = r+1
1000 IF r <> 9 THEN 1020
1010 r = 1
1020 RETURN
```

MISCELLANEOUS TID-BITS

The purpose of this column is to inform you of interesting little pieces of information that are not big enough subjects to warrant an article on its own. It may be a bug in the ADAM's software, or a 14 little-known feature of SmartBASIC, a useful system routine that can be CALL'ed from your Basic program, or a nifty little work-around for some nagging 'problem everyone seems to encounter. If you have a neat trick would benefit others, that you think send it to us and we'll make sure you got credit for it. Here's a couple I bet you aren't aware of....

If you have ever tried to POKE a value into a memory above 53630, you know that it doesn't work. First thought that it was Read-Only memory. Actually, it's only because SmartBASIC just won't do it for your protection. The limit is stored in 16149 and 16150. It you poke a value a 255 into both locations, you will be able to do a POKE anywhere you like!

Did you ever notice that every time you save a Basic program, all the REM and DATA statements grow by one character? SmartBASIC will always add a space after the words REM and DATA. YOU must be very careful with our Utilities since the assembly language routines are contained in DATA statements and are very long. Save it a few times you will first lose the 201 which is the machine code for RETURN. To fix this situation, list the-line on the screen, run the cursor over the line number, space over the DATA (or REM) until you can retype the word in front of the original data, then run the cursor over the rest of the line.

CLOSING REMARKS

SERENDIPITY hopes you found this issue enjoyable and educational. we would appreciate your comments regarding any aspect of this publication. if you would like to see specific topics covered, please let us know and we'll try to discuss them in future issues. lastly, if you have any information about the ADAM system you would like to share with our readers, please feel free to send it to us.

Listing #2

PROGRAM POKEVM

DATE December 27, 1984

PURPOSE Pokes a character into VDP memory as many times as desired. The destination address, character count and ASCII character code must be poked into this routine before execution.

Byte Count	Decimal Values	Op Code	Argument	Comments
1	245	PUSH	AF	Store registers that
2	197	PUSH	BC	we will be using
3	1 0 0	LD	BC, $00	Character count
6	62 0	LD	A, $00	Low byte of destination
8	211 191	OUT	$BF,A	Send it to VDP
10	62 0	LD	A, $00	High byte of destination
12	211 191	OUT	$BF, A	Send it to VDP
14	227	EX	(SP), HL	Time delay required by
15	227	EX	(SP), HL	by VDP for address
16	227	EX	(SP), HL	set up.
17	227	EX	(SP), HL	
18	227	EX	(SP), HL	
19	227	EX	(SP), HL	
20	62 0	LD	A, $00	Ascii code to be printed
22	211 190	OUT	$BE, A	Send data to VDP
24	121	DEC	BC	Decrement the count
25	120	LD	A, B	Load "B" into "A" for compare
26	177	OR	C	If count isn't zero then
27	32 247	JR	NZ,$F7	go back through loop
29	193	POP	BC	Else restore registers
30	241	POP	AF	and return to BASIC
31	201	RET		

Listing 03

PROGRAM PEEKV

DATE December 27, 1984

PURPOSE PEEKS a character from VDP memory. The destination address must be poked into this routine before execution.

Byte Count	Decimal Values	Op Code	Argument	Comments
1	245	PUSH	AF	Store registers
2	62 0	LD	A, $00	Low byte of destination
4	211 191	OUT	$BF, A	Send it to VDP
6	62 0	LD	A, $00	High byte of destination
8	211 191	OUT	$BF, A	Send it to VDP
10	227	EX	(SP), HL	Time delay required by
11	227	EX	(SP), HL	set up.
12	227	EX	(SP), HL	
13	227	EX	(SP), HL	
14	227	EX	(SP), HL	
15	227	EX	(SP), HL	
16	219 190	IN	A, $BE	Get data to VDP
18	50 0 0	LD($0000),A		Basic rtn. will supply this
21	241	POP	AF	Restore register and
22	201	RET		return to Basic

The ADAM TECHNICAL JOURNAL is published bi-monthly by Serendipity Productions. Subscription rates are $15.00 per year in the U.S. and Canada, $20.00 per year in any other country, payable by check or money order only. Single issues are available for $3.00. All inquiries and payments should be made to Serendipity Productions, P.C. Box 07592t Milwaukee, WX 53207.

Preview of the next issue:
==========

Making the most out of hi-res graphics.

Defining and moving graphics Sprites for animation.

Using the joysticks for graphics motion control

The second installation of the Basic Utility Library containing all the sprite control utilities

All concepts are demonstrated in an example program which does free-hand screen painting with the joystick.

More useful Tid-Bits of information.