BLOCK SHAPE ANIMATION X

DOS 3.3 O 0

In this finale to the exploration of block shape animations, an ampersand command interpreter for the block shape routines is presented. Now you can easily access the most commonly used routines without knowing their CALL addresses.

ProDOS O 0

by Robert R. Devine, Computers and You, Mellor Park Mall, 1855 North West Ave., El Dorado, AR 71730

In the first Graphics Workshop article on block shape animation (Vol. 4/No. 3), I introduced the block shape driver, a routine that we have built upon throughout the series. While the block shape driver is easy to use and creates exciting graphic routines, it is not without drawbacks. A program replete with CALLs executes very rapidly, but it can be difficult to debug, since it's not easy to remember what the CALLs actually do. In addition, many of the routines that use the block shape driver require that four or five POKEs be executed before the routine can be CALLed.

Wouldn't it be great if we could change a line such as this:

```
10  POKE 251,144: POKE 252,VT:
    POKE 253,VB: POKE 254,HR: POKE
    255,HL: CALL 37679: REM DRAW
    SHAPE #144
```

to:

```
10 &DRAU144,VT,VB,HR,HL
```

One simple statement would execute the proper CALL and pass along all the values that are needed. The second statement is a more comprehensible instruction, and it saves lots of memory. Moreover, instead of using the Applesoft interpreter to interpret six separate instructions, we could use the ampersand to jump directly into machine code and execute the entire statement at machine language speed.

When I set out to write this routine, which I call the GW.TRANSLATOR (**Listing 1**), I decided against using one of the many ampersand routines currently available; instead, I resolved to start from scratch and build my own routine.

The authors of ampersand routines usually tell you what their routines do, but not how or why they work. Building the translator taught me things about the Apple and how it treats Applesoft programs that I never knew before. Let's see what I found out, and how the translator was created.

DEFINING THE OBJECTIVE

The first step is to outline how we want to construct our commands:

1. Commands such as GOUP, GODN, MOVL, MOVR, INCU and INCD should be written in the format **&GOUP** or **&MOVL**.
2. Commands such as DRAU, DRWD, SCAN and REVD, which require values for SHNUM, VT, VB, HR and HL, should be written in the format **&DRAU SHNUM,VT,VB,HR,HL**.
3. Commands such as SFTL, SFTR, SFTD and SFTU, which require values for VT, VB, HR and HL, should be written in the format **&SFTL VT,VB,HR,HL**.
4. The parameters passed by the commands should allow the use of variables or constants interchangeably. For instance, **&DRAU SHNUM,VT,VB,HR,HL**, or **&DRAU 144,10,50,35,24**. . . , or even **&DRAU 144,VT,VB,35,24** should all be valid statements.

SETTING THE RULES

To be as efficient as possible, the parameter variables, SHNUM, VT, VB, HR and HL, are renamed S, T, B, R, and L, respectively. Also, to make translation as fast as possible, we'll change them all to the integer variables S%, T%, B%, R% and L%.

Finally, we don't want to waste time searching a variable table. Therefore, we will establish the integer variables as the first variables in the table, so we'll know exactly where to find them. This is done by making the first line in your program read:

```
5 S%=Ø: T%=Ø: B%=Ø: R%=Ø: L%=Ø
```

EXAMPLE 1: Applesoft Program Storage

```
Ø800-  ØØ  22  Ø8  Ø5  ØØ  53  25  DØ
       --  --  --  5   --  S   %    =
Ø808-  35  3A  54  25  DØ  31  35  3A
       5   :   T   %    =  1   5   :
Ø810-  42  25  DØ  33  34  3A  52  25
       B   %    =  3   4   :   R   %
Ø818-  DØ  31  33  37  3A  4C  25  DØ
        =  1   3   7   :   L   %    =
Ø820-  32  ØØ  3B  Ø8  ØA  ØØ  AF  44
       2   --  --  --  1Ø  --  &   D
Ø828-  52  41  55  53  2C  54  2C  42
       R   A   U   S   ,   T   ,   B
Ø830-  2C  52  2C  4C  3A  AF  47  4F
       ,   R   ,   L   :   &   G   O
Ø838-  55  5Ø  ØØ  54  Ø8  14  ØØ  AF
       U   P   --  --  --  2Ø  --  &
Ø84Ø-  53  43  41  4E  31  34  32  2C
       S   C   A   N   1   4   2   ,
Ø848-  31  3Ø  2C  32  32  2C  32  34
       1   Ø   ,   2   2   ,   2   4
Ø85Ø-  2C  31  35  ØØ  ØØ  ØØ  ØA  D3
       ,   1   5   --  --  --  --  S%
Ø858-  8Ø  ØØ  Ø5  ØØ  ØØ  ØØ  D4  8Ø
       --  --  5   --  --  --  T%  --
Ø86Ø-  ØØ  ØF  ØØ  ØØ  ØØ  C2  8Ø  ØØ
       --  15  --  --  --  B%  --  --
Ø868-  22  ØØ  ØØ  ØØ  D2  8Ø  ØØ  89
       34  --  --  --  R%  --  --  137
Ø87Ø-  ØØ  ØØ  ØØ  CC  8Ø  ØØ  Ø2  ØØ
       --  --  --  L%  --  --  2   --
```

HOW APPLESOFT STORES INSTRUCTIONS

You should note that since we won't be passing true variables to our ampersand routines, we can reference them in the ampersandcommands as S, T, B, R and L, with no need to use the percent sign (%) format. The percent sign format is only required within Applesoft's assignment statements to be sure that the variables are formatted by Applesoft as integers.

The best way to find out how Applesoft handles an instruction is to try something and then look at the results. Enter the following short program and run it.

```
5   S%=5: T%=15: B%=34: R%=137:
    L%=2
10 &DRAU S,T,B,R,L: & GOUP
20 &SCAN 142,10,22,24,15
```

It crashed, didn't it? But, of course, you knew it would, since **lines 10-20** are one big SYNTAX ERROR! When you ran it, though, you did set up the variable table, and the program is in memory, so let's look at it.

Enter the Monitor with CALL −151. Now enter **800.877**, and let's see how the program is stored. **Example 1** shows the hex bytes and their translation. You can see that most keyboard characters are stored in the listing using their ASCII values. Note that symbols like the equal sign (=) are replaced by a special token (in this case it's D0) that indicates a specific command. Also, every program line begins (or ends, depending on your point of view) with 00 followed by two bytes that point to the beginning of the next program line. Two bytes are reserved for each program line number.

Now enter **69.6A**, and your Apple should respond with 0069-57 08, which translates to the address 0856. If you type **857** to look at address 0857, you'll find D3 which represents S%, the name of the first variable in the variable table. Obviously, the values in locations 69 and 6A tells us where the variable table begins. Since we know exactly where each of our variables (S, T, B, R and L) are located in relation to the address shown, we can access them directly with no need to search a variable table.

Next, let's change a few things to see what happens. Change the statement S%=5 to S=5 in **line 5**, run the program, and enter **800.877** to display the program again. Finally, enter **69.6A** again. You'll notice that the variable table now begins at address 0856. Looking at address 0856, you'll note that the variable name has changed from D3 (with bit 7 set to one) to 53 (the true ASCII code for S). This is how Applesoft distinguishes between integer and floating point variables in the table.

You'll also note that when we set S%=5, we found the hex equivalent of five (05) in the table, all ready for us to use. However, when we set S=5, the value was represented as the two bytes 83 20. The 83 represents the exponent (and tells us the number is positive), and the 20 is the mantissa. Though we could translate these values to our original value of five, the use of integer variables simplifies the task.

Now that we know how and where our variables are stored, let's take a look at how the syntactically erroneous **lines 10 and 20** appear in memory. The first thing to note is that each statement ends with either a colon (the end-of-statement marker) or a 00 (the end-of-line marker). Therefore, we can detect the end of any statement by looking for either a 3A (:) or a 00. Since everything appears in memory in ASCII code, we can easily step through each ampersand command to get the proper translation.

The next step is to look at **line 20**, where we're trying to pass specific values, not variable names. Taking 142 as an example, how do we translate 31 34 32 to the hex value BE, which is what we really want? Notice that if we could remove the threes from 31, 34 and 32, we would end up with 01, 04 and 02. This can be accomplished by ANDing each byte with $0F; it is the first step in our translation.

. . .we could use the ampersand to jump directly into machine code and execute the entire statement at machine language speed.

Now let's think of how a decimal number is built. What is 142? When I went to school I was told that 142 = 2 + 4*10 + 1*100. If we start with $05, then add $28 (the hex notation of 40), we'll get $2D (42). Finally, if we add $64 (the hex notation of 100), we end up with $8E (142 decimal). This is how we'll go about translating the specific numbers that we want to pass.

Next, try changing the DRAU part of **line 10** to DRAW, and look to see how it is stored. You'll notice that Applesoft stored the ASCII codes for the four letters in DRAU, while DRAW was compacted to just one byte (94). The reason is that DRAW is a reserved word and is therefore assigned a token value.

THE PROGRAM POINTERS

The biggest difficulty with an ampersand translator is keeping track of just where we are in the Applesoft program so that we know exactly what part of which instruction we're looking at in any given instant. We already know that 69.6A points to the start of the variable table. Now let's meet TXTPTR.

TXTPTR is another two-byte pointer (located at B8.B9) that keeps track of important Applesoft information. In this case, it points to the byte which is currently being interpreted and executed by Applesoft.

THE TRANSLATION

Using TXTPTR, the first task of the translator is to step through the four bytes immediately following the ampersand to try to find a match in the command table. If a match is found, we'll assign a command number and modify the translator to point to the proper CALL address for the command. If no match is found, we'll let the program bomb with a SYNTAX ERROR message.

If the command found doesn't require parameters, TXTPTR is moved forward four bytes to point to the next Applesoft instruction, to avoid a SYNTAX ERROR. Then we'll execute the command and return to BASIC.

Here is the whole key to using ampersand routines: If we find a valid command, we step over it after translation so that Applesoft never sees or tries to interpret the instruction. If the translator doesn't find a legal command, we simply leave TXTPTR pointing to the invalid command. When Applesoft sees it, the interpreter does the dirty work, and a SYNTAX ERROR occurs. It's the old "what you don't see won't hurt you" trick. As long as we prevent Applesoft from seeing our ampersand commands, we can use any syntax we like.

TABLE 1: Translator Commands

Command	Function
SFTLT,B,R,L	Shift shape left one bit.
SFTRT,B,R,L	Shift shape right one bit.
SFTDT,B,R,L	Shift shape down one byte.
SFTUT,B,R,L	Shift shape up one byte.
DRAUS,T,B,R,L	Draw a shape in the normal posi
DRWDS,T,B,R,L	Draw a shape upside-down.
SCANS,T,B,R,L	Scan a shape into a shape table.
REVDS,T,B,R,L	Draw a shape reversed right-left.
GOUP	Subtract YINCR from $FB and $
GODN	Add YINCR to $FB and $FC.
MOVL	DECrement R%, L%, $FE and $
MOVR	INCrement R%, L%, $FE and $
INCU	DECrement T%, B%, $FB and $
INCD	INCrement T%, B%, $FB and $

If we find a legal command that needs to pass parameters, we start a search for the next colon (3A) or the next 00, which tells us the length of our command statement. When we find the end, we make a note of it so that we can reset TXTPTR to that address, again stepping past the SYNTAX ERROR.

Now we'll start working backwards through the list of variables, beginning with L and working down to S. If we encounter an L, R, B, T or S, we know that we need to go to the variable table to get what we want. We know that L is located 31 bytes past the address shown in 69.6A, while R, B, T and S are located at 69.6A + 24, 17, 10, and 3 bytes, respectively, so getting the Applesoft variables requires no search or translation. If we encounter a number (ASCII codes 30-39), then we know that constants, not variables, are being passed, so the appropriate steps are taken to isolate and combine the digits.

At this point, we keep moving leftward to see if the next byte is another number or if we've reached a comma so we can determine how many digits (one to three) the number contains. You should be aware that GW.TRANSLATOR is set up to properly handle only values between 0 and 199. Since you'll never use values greater than 199 with the block shape driver, there's no need to anticipate larger numbers.

RECAPPING THE STEPS

Let's stop here and recap by going step-by-step through what happens when the translator is called by the ampersand command. Upon entry to the translator at $8F37, 69.6A is pointing to the start of the variable table (at S%), and B8.B9 is pointing to the first character past the ampersand.

Now it steps through the first four characters it encounters in the program, looking for a match in the command table. If no match is found, program flow returns to BASIC and forces a SYNTAX ERROR. If a match is found, the address of the proper command is placed in addresses $8F73 and $8F74 in preparation for jumping to that location.

Next the routine checks to see if this command normally includes the passing of parameters that the driver routine will use. If no values are required, it checks to see if it's dealing with MOVL, MOVR, INCU or INCD, which require that T% and B% or R% and L% be updated. If so, it does the update, moves the program pointer and executes the command; otherwise, it simply moves the program pointer and executes the command.

If parameters are required, the routine jumps to START and finds the total length of the command sequence. If the command length is four (no variables or values included), it jumps back to reset the program pointer and executes the command. This allows you the option of setting command values elsewhere in the program, without the need to pass them through the ampersand command.

Now the routine works from right to left through the command sequence. If it encounters an L, R, B, T or S (in the proper order), it goes to the variable table to get the value. You should note that the translator expects to find the letters in the proper sequence. If it doesn't find the letter it expects, it assumes that the character is numeric and attempts to translate it into a numeric value. Therefore, incorrect syntax in your ampersand commands can have unexpected and potentially disastrous results.

If the expected letter is not found, the routine jumps to the section that translates the numbers it's using.

After going through all the variables or constants, translating them and putting them where the block shape driver will find them, the routine resets the program pointers, executes the command, and returns to BASIC.

ENTERING THE TRANSLATOR

To key in the translator, use an assembler to enter the source code shown in **Listing 1** and assemble it. You may also use the Monitor to enter the machine language code and save it on disk with the command:

BSAVE GW.TRANSLATOR,A$8E7E, L$1F8

For more information on entering machine code, see "A Welcome to New *Nibble* Readers" at the beginning of this issue.

TESTING THE TRANSLATOR

Table 1 is a list of the commands the translator recognizes. You may use S, T, B, R, L or specific values in each command. You'll notice that the last four commands modify not only the zero page values that the driver will use, but also ensure that the variables that represent those values are consistent with the zero page values. This allows you to use the CALL sequences described in earlier installments of the Graphics Workshop, if necessary.

To test GW.TRANSLATOR, enter and run the Applesoft program shown in **Listing 2**. In this test a box is created and then moved about the screen using GW.TRANSLATOR. You'll need to have a copy of the block shape driver (BLOCK.ROUTINES) on the same disk so that it can be BLOADed in **line 100** (see Vol. 6/No. 4). This test will try out most of the routines in the translator so you get a sense of how it works and can check to see that you've entered the translator properly.

For readers who may have missed that issue, the driver is reprinted in **Listing 3**. Use the Monitor to enter the machine code shown, and save it on disk with the command:

BSAVE BLOCK.ROUTINES,A$9076, L$58A

There are many other possible entry points in the block shape driver, and not all are included in the translator. The ones that I've included are those that are most often used. If you wish, you can add more commands to the translator or use the normal CALLs to access other parts of the driver.

Whether you work with the block shape driver or some other special utility, you can use the same techniques that we've looked at here to build your own ampersand command translator routines. They will extend your BASIC commands and make your Apple even more powerful.

LISTING 1: GW.TRANSLATOR

```
                 Ø9ØØ * GW.TRANSLATOR
                 Ø91Ø * BY BOB DEVINE
                 Ø92Ø * COPYRIGHT (C) 1985
                 Ø93Ø * BY MICROSPARC, INC.
                 Ø94Ø * CONCORD, MA Ø1742
                 Ø95Ø *
                 Ø96Ø * S-C ASSEMBLER
                 Ø97Ø * REQUIRES BLOCK.ROUTINES
                 Ø98Ø * TO BE IN MEMORY
                 Ø99Ø *
                 1ØØØ  .OR $8E7E
                 1Ø1Ø  .TF GW.TRANSLATOR
ØØFB-            1Ø2Ø SHNUM .EQ $FB
ØØFC-            1Ø3Ø VT .EQ $FC
ØØFD-            1Ø4Ø VB .EQ $FD
ØØFE-            1Ø5Ø HR .EQ $FE
ØØFF-            1Ø6Ø HL .EQ $FF
ØØØ8-            1Ø7Ø HLDR .EQ $Ø8
ØØØ8-            1Ø8Ø CNTR .EQ $Ø8
ØØØ9-            1Ø9Ø FLAG .EQ $Ø9
ØØB8-            11ØØ P .EQ $B8
ØØØ6-            111Ø HLDR2 .EQ $Ø6
ØØE3-            112Ø YINCR .EQ $E3
8E7E- AØ Ø4      113Ø START LDY #4 * POINT TO P+4
8E8Ø- B1 B8      114Ø J1 LDA (P),Y * GET A BYTE
8E82- FØ Ø7      115Ø  BEQ J2       * FOUND THE ØØ BYTE
8E84- C9 3A      116Ø  CMP #$3A     * IS IT A :?
8E86- FØ Ø3      117Ø  BEQ J2       * FOUND A COLON
8E88- C8         118Ø  INY          * POINT TO NEXT BYTE
8E89- DØ F5      119Ø  BNE J1       * TRY NEXT BYTF
8E8B- 84 Ø9      12ØØ J2 STY FLAG
8E8D- 88         121Ø  DEY          * BACK UP 1 BYTE
8E8E- CØ Ø3      122Ø  CPY #3       * PASSING VARIABLES?
8E9Ø- BØ Ø3      123Ø  BCS J15      * YES, TRANSLATE THEM
8E92- 4C 5D 8F   124Ø  JMP EXEC2    * NO, RESET AND EXECUTE
8E95- B1 B8      125Ø J15 LDA (P),Y * GET THE BYTE
8E97- C9 4C      126Ø  CMP #$4C     * IS IT L?
8E99- FØ Ø6      127Ø  BEQ GETL     * YES, TRANSLATE IT
8E9B- 2Ø E7 8F   128Ø  JSR TRNSNM   * GO TRANSLATE #
8E9E- 4C A8 8E   129Ø  JMP J3
8EA1- 84 Ø6      13ØØ GETL STY HLDR2
8EA3- AØ 1F      131Ø  LDY #31      * POINT TO VAR2
8EA5- 2Ø EØ 8F   132Ø  JSR LSRTST   * TRANSLATE L TO #
8EA8- 85 FF      133Ø J3 STA HL     * STORE L IN HL
8EAA- B1 B8      134Ø  LDA (P),Y    * GET THE BYTE
8EAC- C9 52      135Ø  CMP #$52     * IS IT R?
8EAE- FØ Ø6      136Ø  BEQ GETR     * YES, TRANSLATE IT
8EBØ- 2Ø E7 8F   137Ø  JSR TRNSNM   * GO TRANSLATE IT
8EB3- 4C BD 8E   138Ø  JMP J4
8EB6- 84 Ø6      139Ø GETR STY HLDR2
8EB8- AØ 18      14ØØ  LDY #24      * POINT TO VAR2
8EBA- 2Ø EØ 8F   141Ø  JSR LSRTST   * TRANSLATE R TO #
8EBD- 85 FE      142Ø J4 STA HR     * STORE R IN HR
8EBF- B1 B8      143Ø  LDA (P),Y    * GET THE BYTE
8EC1- C9 42      144Ø  CMP #$42     * IS IT B?
8EC3- FØ Ø6      145Ø  BEQ GETB     * YES TRANSLATE IT
8EC5- 2Ø E7 8F   146Ø  JSR TRNSNM   * GO TRANSLATE IT
8EC8- 4C D2 8E   147Ø  JMP J5
8ECB- 84 Ø6      148Ø GETB STY HLDR2
8ECD- AØ 11      149Ø  LDY #17      * POINT TO VAR2
8ECF- 2Ø EØ 8F   15ØØ  JSR LSRTST   * TRANSLATE B TO #
8ED2- 85 FD      151Ø J5 STA VB     * STORE B IN VB
8ED4- B1 B8      152Ø  LDA (P),Y    * GET THE BYTE
8ED6- C9 54      153Ø  CMP #$54     * IS IT T?
8ED8- FØ Ø6      154Ø  BEQ GETT     * YES TRANSLATE IT
8EDA- 2Ø E7 8F   155Ø  JSR TRNSNM   * GO TRANSLATE IT
8EDD- 4C E7 8E   156Ø  JMP J6
8EEØ- 84 Ø6      157Ø GETT STY HLDR2
8EE2- AØ ØA      158Ø  LDY #1Ø      * POINT TO VAR2
8EE4- 2Ø EØ 8F   159Ø  JSR LSRTST   * TRANSLATE T TO #
8EE7- 85 FC      16ØØ J6 STA VT     * STORE T IN VT
8EE9- B1 B8      161Ø  LDA (P),Y    * GET THE BYTE
8EEB- C9 53      162Ø  CMP #$53     * IS IT S?
8EED- FØ ØA      163Ø  BEQ GETS     * YES TRANSLATE IT
8EEF- C9 3A      164Ø  CMP #$3A     * IS IT A #?
8EF1- BØ ØF      165Ø  BCS J9       * NO, WE'RE DONE
8EF3- 2Ø E7 8F   166Ø  JSR TRNSNM   * GO TRANSLATE IT
8EF6- 4C ØØ 8F   167Ø  JMP J8
8EF9- 84 Ø6      168Ø GETS STY HLDR2
8EFB- AØ Ø3      169Ø  LDY #3       * POINT TO VAR2
8EFD- 2Ø EØ 8F   17ØØ  JSR LSRTST   * TRANSLATE S TO #
8FØØ- 85 FB      171Ø J8 STA SHNUM  * STORE S IN SHNUM
8FØ2- A5 Ø9      172Ø J9 LDA FLAG   * GET NEXT INSTRUCTION POINTER
8FØ4- 4C 61 8F   173Ø  JMP EXEC3
8FØ7- A2 ØØ      174Ø ENTER LDX #Ø  * POINT TO START OF TABLE  (ENTER AT $8FØ7)
8FØ9- 86 Ø8      175Ø  STX CNTR     * SET COMMAND COUNTER
8FØB- AØ ØØ      176Ø J1Ø LDY #Ø    * POINT TO START OF COMMAND GROUP
8FØD- 84 Ø9      177Ø  STY FLAG     * SET DETECTION FLAG
8FØF- BD 18 9Ø   178Ø J11 LDA TEST,X * GET CHARACTER FROM TABLE
8F12- D1 B8      179Ø  CMP (P),Y    * SAME AS PROGRAM COMMAND?
8F14- FØ Ø2      18ØØ  BEQ J12      * YES, WE'RE OKAY
8F16- E6 Ø9      181Ø  INC FLAG     * NOT A MATCH
8F18- E8         182Ø J12 INX       * POINT TO NEXT TABLE ELEMENT
8F19- C8         183Ø  INY          * POINT TO NEXT COMMAND ELEMENT
8F1A- CØ Ø4      184Ø  CPY #4       * ARE WE DONE WITH THIS GROUP?
8F1C- 9Ø F1      185Ø  BCC J11      * NO, CHECK NEXT ELEMENT
8F1E- A5 Ø9      186Ø  LDA FLAG     * CHECK DETECTION FLAG
8F2Ø- FØ Ø9      187Ø  BEQ J13      * WE FOUND COMMAND
8F22- E6 Ø8      188Ø  INC CNTR     * POINT TO NEXT COMMAND GROUP
8F24- A5 Ø8      189Ø  LDA CNTR     * CHECK COMMAND COUNTER
8F26- C9 ØE      19ØØ  CMP #14      * ARE WE AT END OF TABLE?
8F28- 9Ø E1      191Ø  BCC J1Ø      * TRY NEXT COMMAND GROUP
8F2A- 6Ø         192Ø J14 RTS       * NO LEGAL COMMAND FOUND, FORCE SYNTAX ERROR
8F2B- A6 Ø8      193Ø J13 LDX CNTR  * SET COMMAND POINTER
8F2D- BD 5Ø 9Ø   194Ø  LDA CLOW,X   * GET COMMAND ADDRESS LO BYTE
8F3Ø- 8D 43 8F   195Ø  STA EXEC+1   * SET JMP LO BYTE
8F33- BD 5E 9Ø   196Ø  LDA CHIG,X   * GET COMMAND ADDRESS HI BYTE
8F36- 8D 44 8F   197Ø  STA EXEC+2   * SET JMP HI BYTE
8F39- A5 Ø8      198Ø  LDA CNTR     * GET COMMAND POINTER
8F3B- C9 Ø8      199Ø  CMP #8       * NEED S,T,B,R,L?
8F3D- BØ Ø6      2ØØØ  BCS EXEC1    * NO, RESET P, EXECUTE COMMAND
8F3F- 4C 7E 8E   2Ø1Ø  JMP START    * YES, SET S,T,B,R,L
8F42- 4C FF FF   2Ø2Ø EXEC JMP $FFFF * EXECUTE COMMAND AND RETURN TO BASIC
8F45- 84 Ø6      2Ø3Ø EXEC1 STY HLDR2
8F47- C9 ØD      2Ø4Ø  CMP #13      * IS IT INCD?
8F49- FØ 4A      2Ø5Ø  BEQ INCD
8F4B- C9 ØC      2Ø6Ø  CMP #12      * IS IT INCU?
8F4D- FØ 39      2Ø7Ø  BEQ INCU
8F4F- C9 ØB      2Ø8Ø  CMP #11      * IS IT MOVR?
8F51- FØ 29      2Ø9Ø  BEQ MOVR
8F53- C9 ØA      21ØØ  CMP #1Ø      * IS IT MOVL?
8F55- FØ 18      211Ø  BEQ MOVL
8F57- C9 Ø9      212Ø  CMP #9       * IS IT GODN?
8F59- FØ 46      213Ø  BEQ GODN
8F5B- DØ 58      214Ø  BNE GOUP     * IT MUST BE GOUP
8F5D- A4 Ø6      215Ø EXEC2 LDY HLDR2
8F5F- A9 Ø4      216Ø  LDA #4       * INCREMENT PROGRAM POINTER
8F61- 18         217Ø EXEC3 CLC
8F62- 65 B8      218Ø  ADC P
8F64- 85 B8      219Ø  STA P        * RESET P-LO
8F66- A5 B9      22ØØ  LDA P+1      * GET POINTER HI
8F68- 69 ØØ      221Ø  ADC #Ø       * ADD ANY OVERFLOW
8F6A- 85 B9      222Ø  STA P+1      * RESET P-HI
8F6C- 4C 42 8F   223Ø  JMP EXEC
8F6F- AØ 1F      224Ø MOVL LDY #31
8F71- 2Ø D1 8F   225Ø  JSR DEC      * DEC L%     MODIFY
8F74- AØ 18      226Ø  LDY #24
8F76- 2Ø D1 8F   227Ø  JSR DEC      * DEC R%     L% AND R% OR
8F79- 4C 5D 8F   228Ø  JMP EXEC2
8F7C- AØ 1F      229Ø MOVR LDY #31              T% AND B%
8F7E- 2Ø CB 8F   23ØØ  JSR INC      * INC L%
8F81- AØ 18      231Ø  LDY #24                  TO CONFORM TO THE
8F83- 2Ø CB 8F   232Ø  JSR INC      * INC R%
8F86- DØ D5      233Ø  BNE EXEC2                CURRENT STATUS OF
8F88- AØ 11      234Ø INCU LDY #17
8F8A- 2Ø D1 8F   235Ø  JSR DEC      * DEC B%     $FE AND $FF OR
8F8D- AØ ØA      236Ø  LDY #1Ø
8F8F- 2Ø D1 8F   237Ø  JSR DEC      * DEC T%     $FC AND $FD
8F92- 4C 5D 8F   238Ø  JMP EXEC2
8F95- AØ ØA      239Ø INCD LDY #1Ø
8F97- 2Ø CB 8F   24ØØ  JSR INC      * INC T%
8F9A- AØ 11      241Ø  LDY #17
8F9C- 2Ø CB 8F   242Ø  JSR INC      * INC B%
8F9F- DØ BC      243Ø  BNE EXEC2
8FA1- AØ ØA      244Ø GODN LDY #1Ø * POINT TO T%
8FA3- B1 69      245Ø  LDA ($69),Y * GET T%
8FA5- 18         246Ø  CLC
8FA6- 65 E3      247Ø  ADC YINCR    * ADD YINCR
8FA8- 91 69      248Ø  STA ($69),Y * REPLACE T%
8FAA- AØ 11      249Ø  LDY #17      * POINT TO B%
8FAC- B1 69      25ØØ  LDA ($69),Y * GET B%
8FAE- 65 E3      251Ø  ADC YINCR    * ADD YINCR
8FBØ- 91 69      252Ø  STA ($69),Y * REPLACE B%
8FB2- 4C 5D 8F   253Ø  JMP EXEC2
8FB5- AØ ØA      254Ø GOUP LDY #1Ø * POINT TO T%
8FB7- B1 69      255Ø  LDA ($69),Y * GET T%
8FB9- 38         256Ø  SEC
8FBA- E5 E3      257Ø  SBC YINCR    * SUBTRACT YINCR
8FBC- 3Ø 9F      258Ø  BMI EXEC2    * IF RESULT <Ø ABORT OPERATION
8FBE- 91 69      259Ø  STA ($69),Y * REPLACE T%
8FCØ- AØ 11      26ØØ  LDY #17      * POINT TO B%
8FC2- B1 69      261Ø  LDA ($69),Y * GET B%
8FC4- E5 E3      262Ø  SBC YINCR    * SUBTRACT YINCR
8FC6- 91 69      263Ø  STA ($69),Y * REPLACE B%
8FC8- 4C 5D 8F   264Ø  JMP EXEC2
8FCB- B1 69      265Ø INC LDA ($69),Y * GET THE VARIABLE
8FCD- AA         266Ø  TAX          * TRANSFER TO X-REGISTER
8FCE- E8         267Ø  INX          * INCREMENT IT
8FCF- DØ Ø6      268Ø  BNE DEC2     * ALWAYS TAKEN
8FD1- B1 69      269Ø DEC LDA ($69),Y * GET THE VARIABLE
8FD3- FØ Ø6      27ØØ  BEQ ABORT    * IF L% OR B% ARE ZERO CANCEL OPERATION
8FD5- AA         271Ø  TAX          * TRANSFER TO X-REGISTER
8FD6- CA         272Ø  DEX          * DECREMENT IT
8FD7- 8A         273Ø DEC2 TXA      * BACK TO ACCUMULATOR
8FD8- 91 69      274Ø  STA ($69),Y * REPLACE THE VARIABLE
8FDA- 6Ø         275Ø  RTS
8FDB- 68         276Ø ABORT PLA     * PULL THE RETURN
8FDC- 68         277Ø  PLA          * FROM THE STACK
8FDD- 4C 5D 8F   278Ø  JMP EXEC2
8FEØ- B1 69      279Ø LSRTST LDA ($69),Y * GET THE VARIABLE
8FE2- A4 Ø6      28ØØ  LDY HLDR2    * RESTORE Y-REGISTER
8FE4- 88         281Ø  DEY          * POINT TO COMMA
8FE5- 88         282Ø  DEY          * POINT TO NEXT VARIABLE
8FE6- 6Ø         283Ø  RTS
8FE7- 29 ØF      284Ø TRNSNM AND #$ØF * MASK LEFT NIBBLE
8FE9- 85 Ø8      285Ø  STA HLDR
8FEB- 88         286Ø  DEY          * POINT TO NEXT BYTE
8FEC- B1 B8      287Ø  LDA (P),Y    * GET THE BYTE
8FEE- C9 2C      288Ø  CMP #$2C     * IS IT A ,?
8FFØ- FØ 22      289Ø  BEQ J7       * YES, WE'RE DONE
8FF2- C9 3A      29ØØ  CMP #$3A     * IS IS A #?
8FF4- BØ 1E      291Ø  BCS J7       * NO, WE'RE DONE
8FF6- 29 ØF      292Ø  AND #$ØF     * MASK LEFT NIBBLE
8FF8- AA         293Ø  TAX          * SET X-REG POINTER
8FF9- A5 Ø8      294Ø  LDA HLDR     * GET BYTE BACK
8FFB- 18         295Ø  CLC
8FFC- 7D 6C 9Ø   296Ø  ADC TABLE,X * ADD NEXT BYTE
8FFF- 85 Ø8      297Ø  STA HLDR
9ØØ1- 88         298Ø  DEY          * POINT TO NEXT BYTE
9ØØ2- B1 B8      299Ø  LDA (P),Y    * GET THE BYTE
9ØØ4- C9 2C      3ØØØ  CMP #$2C     * IS IT A ,?
9ØØ6- FØ ØC      3Ø1Ø  BEQ J7       * YES, WE'RE DONE
9ØØ8- C9 3A      3Ø2Ø  CMP #$3A     * IS IT A #?
9ØØA- BØ Ø8      3Ø3Ø  BCS J7       * NO, WE'RE DONE
9ØØC- A5 Ø8      3Ø4Ø  LDA HLDR     * GET BYTE BACK
9ØØE- 18         3Ø5Ø  CLC
9ØØF- 69 64      3Ø6Ø  ADC #$64     * ADD LAST BYTE HEX OF 1ØØ
9Ø11- 85 Ø8      3Ø7Ø  STA HLDR
9Ø13- 88         3Ø8Ø  DEY
9Ø14- 88         3Ø9Ø J7 DEY        * POINT TO NEXT BYTE
9Ø15- A5 Ø8      31ØØ  LDA HLDR     * GET CONVERTED BYTE
9Ø17- 6Ø         311Ø  RTS
9Ø18- 53 46 54
9Ø1B- 4C 53 46
9Ø1E- 54 52 53
9Ø21- 46 54 44
9Ø24- 53 46 54
9Ø27- 55         312Ø TEST .HS 5346544C5346545253465444534654 55   * Ø-3
9Ø28- 44 52 41
9Ø2B- 55 44 52
9Ø2E- 57 44 53
9Ø31- 43 41 4E
9Ø34- 52 45 56
9Ø37- 44         313Ø  .HS 44524155445257445343414E52455644      * 4-7
9Ø38- 47 4F 55
9Ø3B- 5Ø 47 4F
9Ø3E- 44 4E 4D
9Ø41- 4F 56 4C
9Ø44- 4D 4F 56
9Ø47- 52         314Ø  .HS 474F55Ø474F444E4D4F564C4D4F5652      * 8-11
9Ø48- 49 4E 43
9Ø4B- 55 49 4E
9Ø4E- 43 44      315Ø  .HS 494E4355494E4344                      * 12-13
9Ø5Ø- B5 ØE AA
9Ø53- EØ 2F 76
9Ø56- 61 E6 5E
9Ø59- 6D A1 BØ
9Ø5C- 15 DB      316Ø CLOW .HS B5ØEAAEØ2F7661E65E6DA1BØ15DB
9Ø5E- 91 92 9Ø
9Ø61- 9Ø 93 9Ø
9Ø64- 93 92 92
9Ø67- 92 91 91
9Ø6A- 91 9Ø      317Ø CHIG .HS 91929Ø9Ø939Ø9392929291919 19Ø
9Ø6C- ØØ ØA 14
9Ø6F- 1E 28 32
9Ø72- 3C 46 5Ø
9Ø75- 5A         318Ø TABLE .HS ØØØA141E28323C465Ø5A HEX OF Ø,1Ø,2Ø...9Ø

ØØØØ ERRORS IN ASSEMBLY
```

END OF LISTING 1

```
        KEY PERFECT 4.0
             RUN ON
        GW TRANSLATOR
==============================
  CODE        ADDR# - ADDR#
 ------       -------------
  2987        8E7E - 8ECD
  25A0        8ECE - 8F1D
  2A33        8F1E - 8F6D
  244E        8F6E - 8FBD
  2A42        8FBE - 900D
  27B3        900E - 905D
  0D34        905E - 9075
PROGRAM CHECK IS : 01F8
```

```
        KEY PERFECT 4.0
             RUN ON
        TRANSLATOR.DEMO
==============================
  CODE        LINE# - LINE#
 ------       -------------
  87D7          10 -   100
  C1BA         110 -   200
  54FA         210 -   250
PROGRAM CHECK IS : 035E
```

LISTING 2: TRANSLATOR.DEMO

```
10  REM  **********************
20  REM  *   TRANSLATOR.DEMO    *
30  REM  *    BY BOB DEVINE     *
40  REM  *  COPYRIGHT (C) 1985  *
50  REM  *  BY MICROSPARC, INC  *
60  REM  *  CONCORD, MA. 01742  *
70  REM  **********************
80 S% = 0:T% = 0:B% = 0:R% = 0:L% = 0: REM  S
     ETUP VARIABLES
90  POKE 1013,76: POKE 1014,7: POKE 1015,143:
     REM SET AMPERJUMP TO $8F07
100  PRINT  CHR$ (4)"BLOAD  BLOCK.ROUTINES"
110  PRINT  CHR$ (4)"BLOAD GW.TRANSLATOR"
120  HIMEM: 35328: CALL 37799: REM   PROTECT
     AND SETUP YTABLE. HIMEM MAY BE 36478 UND
     ER DOS 3.3
130  HGR : HCOLOR= 3: FOR X = 1 TO 13: HPLOT
     X,1 TO X,14: NEXT X
140  & SCAN142,0,15,1,0: REM  CREATE SHAPE #1
     42
150  HGR : REM  CLEAR THE SCREEN
160 S% = 142:T% = 0:B% = 15:R% = 1:L% = 0: REM
     SET VARIABLE VALUES
170  & DRAUS,T,B,R,L: REM  DRAW THE SHAPE
180 R% = R% + 1: REM  PUT AN EMPTY BYTE AHEAD
190  FOR X = 1 TO 35: FOR Y = 1 TO 7: & SFTRT
     ,B,R,L: NEXT Y: & MOVR: NEXT X: REM  MOV
     E IT RIGHT
200  FOR X = 1 TO 140: & SFTDT,B,R,L: & INCD:
     NEXT X: REM MOVE IT DOWN
210  & MOVL: REM PUT EMPTY BYTE AHEAD
220  FOR X = 1 TO 35: FOR Y = 1 TO 7: & SFTLT
     ,B,R,L: NEXT Y: & MOVL: NEXT X: REM MOVE
     IT LEFT
230  FOR X = 1 TO 140: & SFTUT,B,R,L: & INCU:
     NEXT X: REM MOVE IT UP
240  IF  PEEK ( - 16384) < 128 THEN 190
250  POKE  - 16368,0: TEXT : HOME : END
```

END OF LISTING 2

LISTING 3: BLOCK.ROUTINES

```
9076- A9 00
9078- 85 FA A5 FC 85 06 20 91
9080- 93 A4 FE A2 00 A1 FA 51
9088- 26 91 26 88 18 E6 FA D0
9090- 02 E6 FB C0 FF F0 04 C4
9098- FF B0 EA E6 06 A5 06 C9
90A0- FF F0 06 C5 FD 90 D7 F0
90A8- D5 60 A5 FD C9 BD B0 2F
90B0- 85 06 20 91 93 A4 FE B1
90B8- 26 85 F9 20 04 F5 A5 F9
90C0- 91 26 20 D5 F4 88 18 C0
90C8- FF F0 04 C4 FF B0 E8 C6
90D0- 06 A5 06 C9 FF F0 04 C5
90D8- FC B0 D7 E6 FC E6 FD 60
90E0- A5 FC C9 01 90 33 E6 FD
90E8- 85 06 20 91 93 A4 FE B1
90F0- 26 85 F9 20 D5 F4 A5 F9
90F8- 91 26 20 04 F5 88 18 C0
9100- FF F0 04 C4 FF B0 E8 E6
9108- 06 A5 06 C9 BE F0 04 C5
9110- FD 90 D7 C6 FD C6 FD C6
9118- FC 60 A9 0E A6 FB 9D 6F
9120- 8F 60 A9 00 8D 54 C0 A9
9128- 40 85 E6 60 A9 00 8D 55
9130- C0 A9 20 85 E6 60 20 22
9138- 91 18 90 03 20 2C 91 20
9140- 0E 92 20 0E 92 20 7A 91
9148- 20 0E 92 20 0E 92 20 7A
9150- 91 A5 E6 C9 40 F0 E5 60
9158- 20 22 91 18 90 03 20 2C
9160- 91 20 B5 91 20 B5 91 20
9168- 8A 91 20 B5 91 20 B5 91
9170- 20 8A 91 A5 E6 C9 40 F0
9178- E5 60 A6 FB DE 6F 8F D0
9180- 08 20 AC 91 A9 0E 9D 6F
9188- 8F 60 A6 FB DE 6F 8F D0
9190- F8 20 97 91 18 90 ED A5
9198- FF C9 02 90 0E C6 FF C6
91A0- FE A5 FF C9 01 90 04 C6
91A8- FF C6 FE 60 E6 FF E6 FE
91B0- E6 FF E6 FE 60 A5 FD 85
91B8- 06 20 91 93 18 A4 FE A9
91C0- 00 85 08 85 09 85 07 90
91C8- 02 E6 08 B1 26 C9 80 90
91D0- 02 E6 09 A5 08 F0 07 B1
91D8- 26 09 80 4C E2 91 B1 26
91E0- 29 7F 6A 91 26 90 02 E6
91E8- 07 A5 09 C9 01 90 06 B1
91F0- 26 09 80 91 26 C4 FF F0
91F8- 08 88 A5 07 C9 01 4C BF
9200- 91 C6 06 A5 06 C9 FF F0
9208- 04 C5 FC B0 AC 60 A5 FD
9210- 85 06 20 91 93 18 A4 FF
9218- A9 00 85 08 85 09 B1 26
9220- 2A 91 26 B0 02 90 02 E6
9228- 08 C9 80 B0 02 90 02 E6
9230- 09 A5 08 D0 09 B1 26 29
9238- 7F 91 26 4C 44 92 B1 26
9240- 09 80 91 26 C4 FE F0 09
9248- C8 18 A5 09 C9 01 4C 18
9250- 92 C6 06 A5 06 C9 FF F0
9258- 04 C5 FC B0 B5 60 38 A5
9260- FC E5 E3 85 FC 38 A5 FD
9268- E5 E3 85 FD 60 18 A5 FC
9270- 65 E3 85 FC 18 A5 FD 65
9278- E3 85 FD 60 A9 00 8D 54
9280- C0 A9 40 85 E6 A5 FC C5
9288- E3 90 0F 20 6D 92 20 2F
9290- 93 20 5E 92 20 5E 92 20
9298- 2F 93 60 A9 00 8D 55 C0
92A0- A9 20 85 E6 20 6D 92 20
92A8- 2F 93 20 5E 92 20 5E 92
92B0- 20 2F 93 60 A9 00 8D 54
92B8- C0 A9 40 85 E6 20 5E 92
92C0- 20 2F 93 20 6D 92 20 6D
92C8- 92 20 2F 93 60 A9 00 8D
92D0- 55 C0 A9 20 85 E6 20 5E
92D8- 92 20 2F 93 20 6D 92 20
92E0- 6D 92 20 2F 93 60 A9 00
92E8- 85 FA A5 FD 85 06 20 91
92F0- 93 A4 FF A2 00 A1 FA C9
92F8- 7F F0 15 C9 01 90 11 86
9300- F9 4A 26 F9 E8 E0 07 90
9308- F8 4A A5 F9 90 02 09 80
9310- 91 26 C8 E6 FA D0 02 E6
9318- FB C4 FE 90 D6 F0 D4 C6
9320- 06 A5 06 C9 FF F0 04 C5
9328- FC B0 C3 20 61 93 60 A9
9330- 00 85 FA A5 FD 85 06 20
9338- 91 93 A4 FE A2 00 A1 FA
9340- 51 26 91 26 88 18 E6 FA
9348- D0 02 E6 FB C0 FF F0 04
9350- C4 FF B0 EA C6 06 A5 06
9358- C9 FF F0 04 C5 FC B0 D7
9360- 60 A9 00 85 FA A5 FD 85
9368- 06 20 91 93 A4 FE A2 00
9370- B1 26 81 FA 88 18 E6 FA
9378- D0 02 E6 FB C0 FF F0 04
9380- C4 FF B0 EC C6 06 A5 06
9388- C9 FF F0 04 C5 FC B0 D9
9390- 60 A4 06 B1 CE 85 26 A5
9398- E6 C9 40 D0 05 B1 DE 85
93A0- 27 60 B1 EE 85 27 60 A9
93A8- 80 85 CE A9 94 85 CF A9
93B0- 40 85 EE A9 95 85 EF A9
93B8- C0 85 DE A9 93 85 DF 60
93C0- 40 44 48 4C 50 54 58 5C
93C8- 40 44 48 4C 50 54 58 5C
93D0- 41 45 49 4D 51 55 59 5D
93D8- 41 45 49 4D 51 55 59 5D
93E0- 42 46 4A 4E 52 56 5A 5E
93E8- 42 46 4A 4E 52 56 5A 5E
93F0- 43 47 4B 4F 53 57 5B 5F
93F8- 43 47 4B 4F 53 57 5B 5F
9400- 40 44 48 4C 50 54 58 5C
9408- 40 44 48 4C 50 54 58 5C
9410- 41 45 49 4D 51 55 59 5D
9418- 41 45 49 4D 51 55 59 5D
9420- 42 46 4A 4E 52 56 5A 5E
9428- 42 46 4A 4E 52 56 5A 5E
9430- 43 47 4B 4F 53 57 5B 5F
9438- 43 47 4B 4F 53 57 5B 5F
9440- 40 44 48 4C 50 54 58 5C
9448- 40 44 48 4C 50 54 58 5C
9450- 41 45 49 4D 51 55 59 5D
9458- 41 45 49 4D 51 55 59 5D
9460- 42 46 4A 4E 52 56 5A 5E
9468- 42 46 4A 4E 52 56 5A 5E
9470- 43 47 4B 4F 53 57 5B 5F
9478- 43 47 4B 4F 53 57 5B 5F
9480- 00 00 00 00 00 00 00 00
9488- 80 80 80 80 80 80 80 80
9490- 00 00 00 00 00 00 00 00
9498- 80 80 80 80 80 80 80 80
94A0- 00 00 00 00 00 00 00 00
94A8- 80 80 80 80 80 80 80 80
94B0- 00 00 00 00 00 00 00 00
94B8- 80 80 80 80 80 80 80 80
94C0- 28 28 28 28 28 28 28 28
94C8- A8 A8 A8 A8 A8 A8 A8 A8
94D0- 28 28 28 28 28 28 28 28
94D8- A8 A8 A8 A8 A8 A8 A8 A8
94E0- 28 28 28 28 28 28 28 28
94E8- A8 A8 A8 A8 A8 A8 A8 A8
94F0- 28 28 28 28 28 28 28 28
94F8- A8 A8 A8 A8 A8 A8 A8 A8
9500- 50 50 50 50 50 50 50 50
9508- D0 D0 D0 D0 D0 D0 D0 D0
9510- 50 50 50 50 50 50 50 50
9518- D0 D0 D0 D0 D0 D0 D0 D0
9520- 50 50 50 50 50 50 50 50
9528- D0 D0 D0 D0 D0 D0 D0 D0
9530- 50 50 50 50 50 50 50 50
9538- D0 D0 D0 D0 D0 D0 D0 D0
9540- 20 24 28 2C 30 34 38 3C
9548- 20 24 28 2C 30 34 38 3C
9550- 21 25 29 2D 31 35 39 3D
9558- 21 25 29 2D 31 35 39 3D
9560- 22 26 2A 2E 32 36 3A 3E
9568- 22 26 2A 2E 32 36 3A 3E
9570- 23 27 2B 2F 33 37 3B 3F
9578- 23 27 2B 2F 33 37 3B 3F
9580- 20 24 28 2C 30 34 38 3C
9588- 20 24 28 2C 30 34 38 3C
9590- 21 25 29 2D 31 35 39 3D
9598- 21 25 29 2D 31 35 39 3D
95A0- 22 26 2A 2E 32 36 3A 3E
95A8- 22 26 2A 2E 32 36 3A 3E
95B0- 23 27 2B 2F 33 37 3B 3F
95B8- 23 27 2B 2F 33 37 3B 3F
95C0- 20 24 28 2C 30 34 38 3C
95C8- 20 24 28 2C 30 34 38 3C
95D0- 21 25 29 2D 31 35 39 3D
95D8- 21 25 29 2D 31 35 39 3D
95E0- 22 26 2A 2E 32 36 3A 3E
95E8- 22 26 2A 2E 32 36 3A 3E
95F0- 23 27 2B 2F 33 37 3B 3F
95F8- 23 27 2B 2F 33 37 3B 20
```

END OF LISTING 3

```
        KEY PERFECT 4.0
             RUN ON
        BLOCK.ROUTINES
==============================
  CODE        ADDR# - ADDR#
 ------       -------------
  264E        9076 - 90C5
  2931        90C6 - 9115
  286D        9116 - 9165
  2A6E        9166 - 91B5
  28F9        91B6 - 9205
  222F        9206 - 9255
  238D        9256 - 92A5
  2592        92A6 - 92F5
  24D1        92F6 - 9345
  2EC8        9346 - 9395
  2B27        9396 - 93E5
  3226        93E6 - 9435
  2748        9436 - 9485
  2418        9486 - 94D5
  2DFF        94D6 - 9525
  2DE8        9526 - 9575
  2937        9576 - 95C5
  1C7C        95C6 - 95FF
PROGRAM CHECK IS : 058A
```