Using BMP2LO to Create LGR (Lo-Res) and DLGR (Double Lo-Res) Files
By Bill Buckels – December 19, 2012 – Revised April 2013

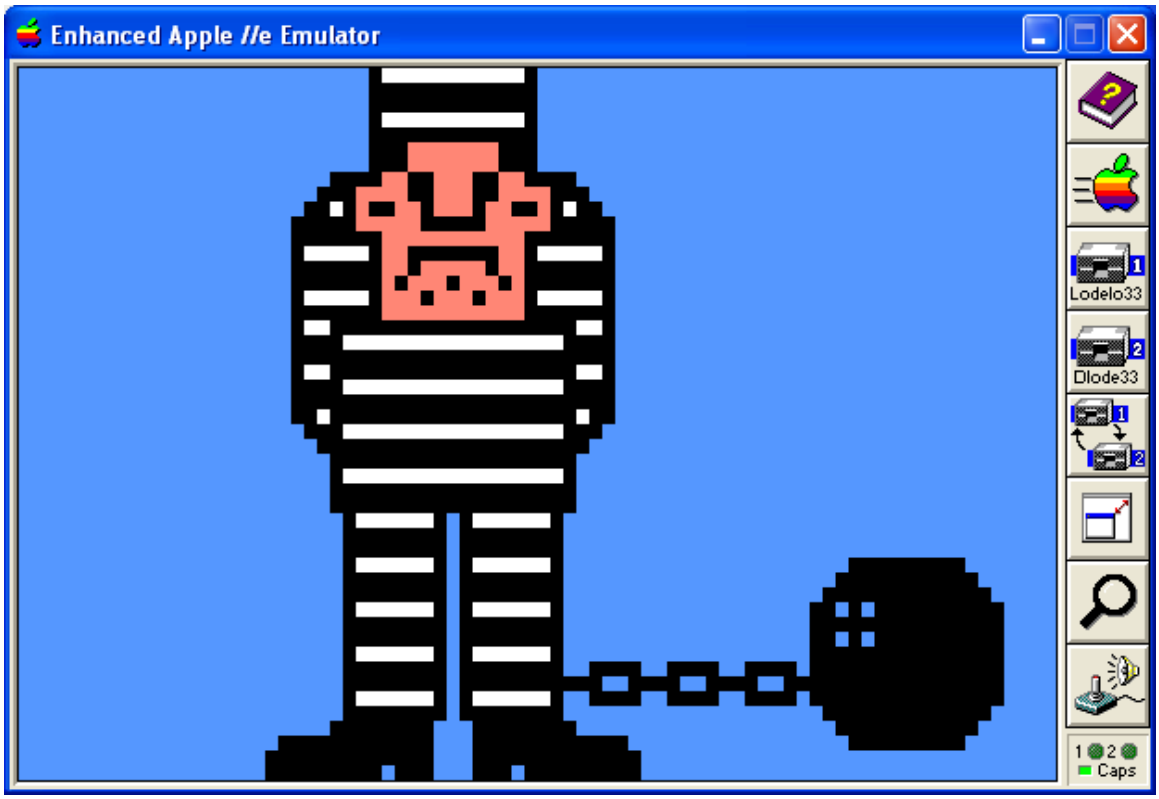


Table of Contents

Licence Agreement

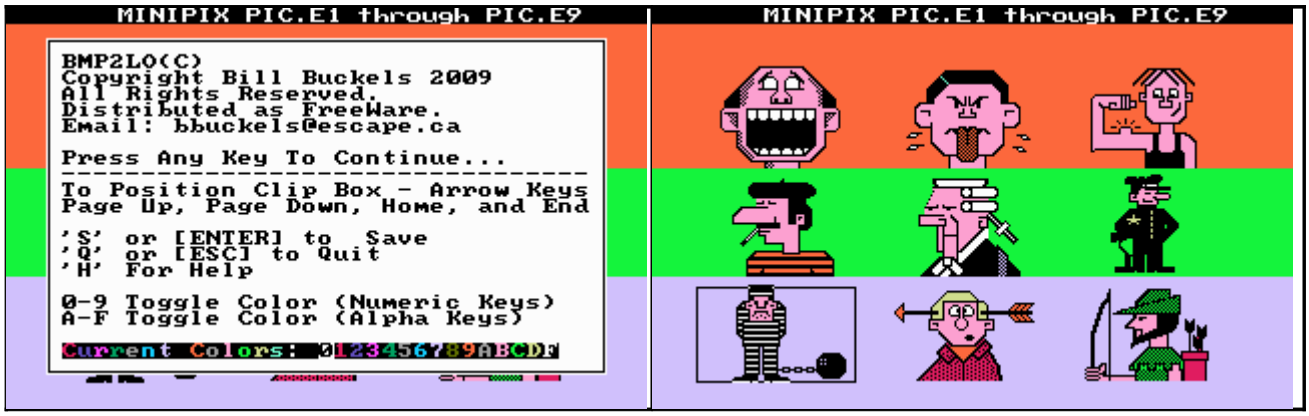
You have a royalty-free right to use, modify, reproduce and distribute BMP2LO and related source code and companion files in any way you find useful, provided that you agree that Bill Buckels has **no warranty obligations or liability** resulting from said distribution or use in any way whatsoever. If you don't agree, don't bother reading further and remove BMP2LO, related source code, companion files and every other artifact of said distribution from your computer now.

Please keep in mind that BMP2LO.exe is distributed without support of any kind.

Intended Audience

I am assuming that you already know what an Apple II file is and you have utilities to place these on Apple II disks or disk images. I assume too that you have some knowledge of IBM PC graphics files including Windows BMP files. I am also assuming that you know how to use the Windows Clipboard and how to use Windows Paint. I am hoping as well that you also know about color palettes and screen resolutions and such. If you don't have this preliminary knowledge this document may not be for you.

Introduction



Back in 2009, as part of a larger collection of programming for the Apple II, I wrote an MS-DOS utility called BMP2LO.EXE. In a nutshell, all BMP2LO does is remap the colors from a PC graphic image to an Apple II graphic image in either Lo-Res (LGR) or Double Lo-Res (DLGR) format and save the results. BMP2LO is distributed with reasonably well-commented source code. For detailed information about what it is actually doing and how it does what it does read the source code and run the program. But read this document first for whatever its worth.

BMP2LO outputs a BSaved Image File pair which can be easily BLoaded in an AppleSoft BASIC program, and also outputs a raster oriented image which is more elegant, and more suitable for a C language program outside BASIC's bog and mire.

BMP2LO.exe Version 3.0 has been much improved and expanded-on. Originally it was part of the Apple33 Aztec C65 distribution for DOS 3.3, but now that support for LGR and DLGR been added to the AppleX Aztec C65 distribution for ProDOS, Apple33 provides a only subset of BMP2LO, and you must download AppleX for the complete version including source code.

How to Get BMP2LO.EXE

BMP2LO.exe is part of the much larger Aztec C Museum project at **www.aztecmuseum.ca** which is targeted at C programming on legacy platforms like the Apple II. It is distributed with the **AppleX** Aztec C65 cross-compiler distribution for ProDOS. The **Apple33** Aztec C65 cross-compiler distribution for DOS 3.3 now provides only a subset of BMP2LO, so you must download AppleX for the complete version including source code.

AWINDLO.EXE – A Companion Utility for BMP2LO.exe

Round trip conversion and titling of LGR and DLGR files is supported by BMP2LO's companion utility, **AWINDLO.exe**. They are distributed together.

BMP2LO Modes of Operation

BMP2LO.exe has two "modes" of conversion to LGR and DLGR files:

- Clipping Editor – Interactive – Single and Double Lo-Res Output. MS-DOS 4-color BAS and PCX files and 16 – Color BMPs only. Note: the Clipping Editor can be (or could be) called from the command line and uses the same command line options as the command line only mode (see below), but launches into the clipping editor's MCGA graphics mode where you can clip from an input file of up to 9 individual graphics per input file and remap colors.

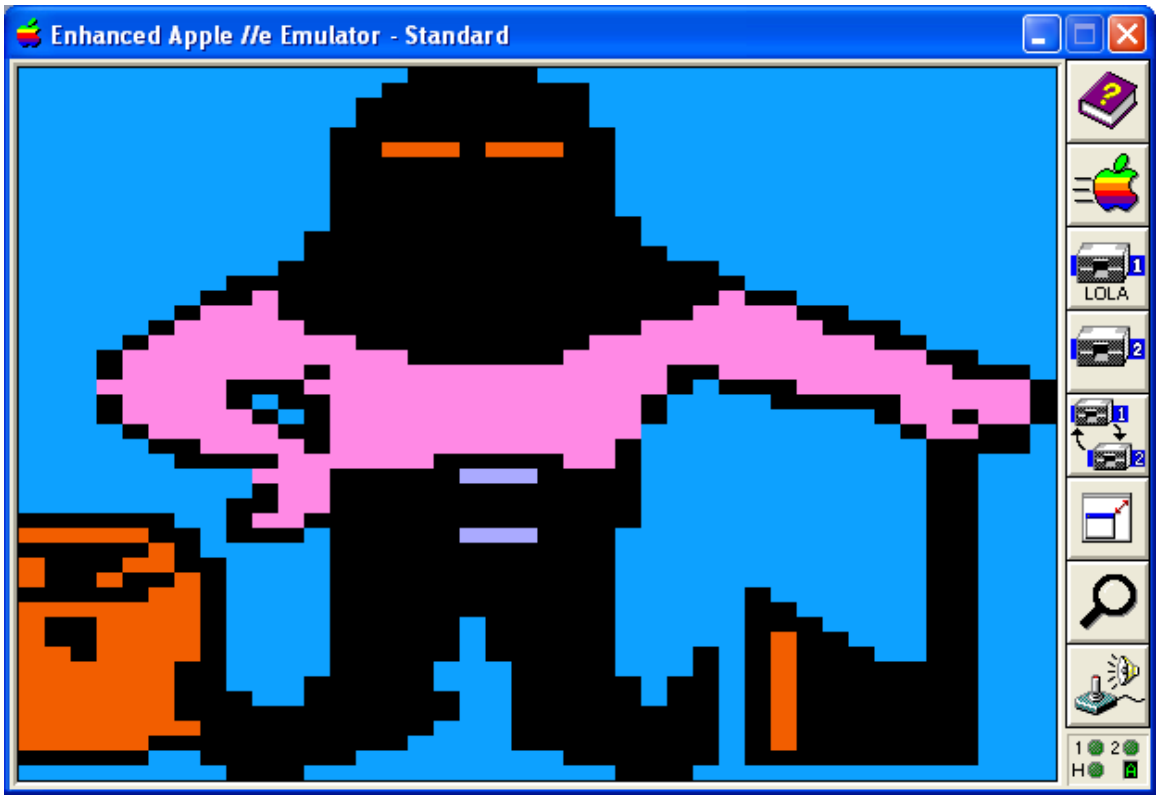
- Command Line Only - Non-Interactive – Single and Double Lo-Res Output. 24 – bit BMPs only. This mode is also "text mode" only. On systems that do not support MS-DOS graphics applications, this is your only option, besides running within an MS-DOS emulator like DOSBox that supports MCGA video mode.

Interactive conversion is designed for "size-on" conversion of sheets of up to 9 "MiniPix" (88 x 52 Old Printshop Graphics) created by clipboard pastes from my ClipShop utility into a Windows Paint 16 Color BMP file. Another utility called Mini2BMP.exe is also distributed with BMP2LO for converting native Apple II Minipix to 16 color bmp files suitable for use with BMP2LO's interactive conversion mode.

Command Line only conversion is designed for conversion of the 24 – bit BMP's produced by AWINDLO and other 24-bit BMP's in a fixed-size range. See below:

BMP Image Formats accepted by BMP2LO version 3.0

Non-Interactive Conversion	
24Bit.BMP	Sizes (resolution) below
AWINDLO or similar	160 x 96 - 160 x 128 - 80 x 48 -80 x 64
MINIPIX single file	88 x 52 or 176 x 104
Interactive Conversion	320 x 200 resolution
MINIPIX 9 clip maximum	CGA 4- Color GWBASIC BSaved Images
MINIPIX 9 clip maximum	CGA 4- Color PCX Images
Interactive Conversion	maximum 320 x 200 resolution
MINIPIX 9 Clip maximum MINIPIX single file	16 Color Windows Paint BMP Files of any size up to 320 x 200



BMP2LO LGR "Lo-Res" Output – Command Line Option L

BMP2LO produces LGR output as well as DLGR output. When converting a BMP to Lo-Res Apple II files you must enter L (for "lo-res") on the command line somewhere following the BMP input file name.

BMP2LO "Mixed Text and Graphics" - Command Line Option T

BMP2LO produces "Mixed Text and Graphics" LGR or DLGR Apple II raster files of 40 scan-lines in height rather than the default "Full Graphics" files of 48 scan-lines in height when the T (Top) option is used. The "lo-res" file has the extension STO ("single top") and the DLGR file has the extension DTO ("double top"). They use less disk space.

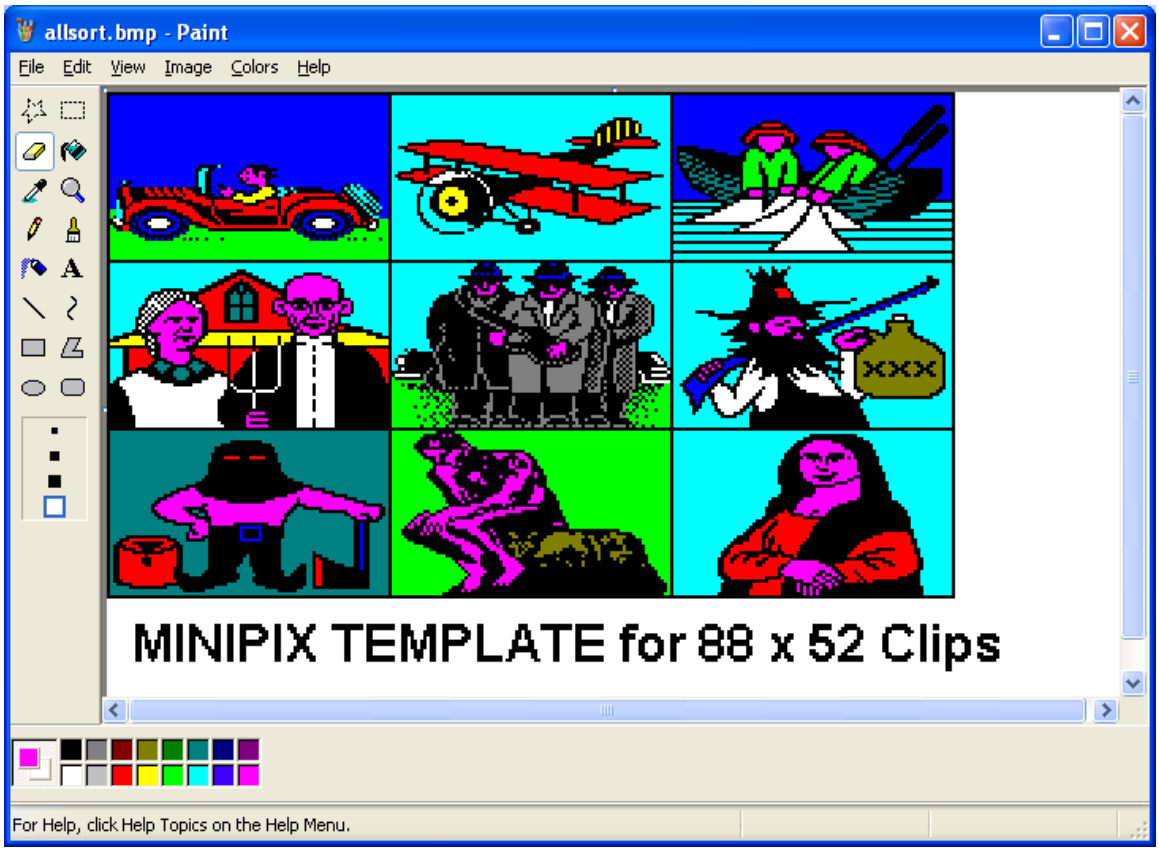
BSAVED Apple II output files are not supported for this feature because it would not make sense… the Apple II LGR/DLGR and text screen memory is a shared area, and is is interleaved and not contiguous, and the memory for the bottom 4 text lines is mixed between chunks of DLGR graphics, so you get a mess when you BLOAD. Raster oriented files are selective about where they are loaded and the mess does not occur.

BMP2LO Output File BaseName Option

Most of my utilities use the basename of the input file for the output file. This saves on typing and isn't generally a problem. But BMP2LO's interactive mode can have up to 9 – 80 x 48 graphics in a 320 x 200 input file… so BMP2LO supports using a different

basename to accommodate files with more than one graphic. By default, if no output file basename follows the input file name on the command line, BMP2LO will just use the basename of the input file for the output file names. Remember MS-DOS 8 character basenames only!

Windows Paint "Roll Your Own" - Creating and Acquiring PC Graphics



The image above was "hand built" by pasting-in IBM-PC Minipix from my **Clipshop** utility into the 16 color BMP template provided for you , and coloring them. See the tutorial in **Appendix A** for more detail. I then converted the 9 Minipix in the BMP to Apple II DLGR files using BMP2LO and built the DLGR slideshow which is on one of the Apple II disks that comes with AWINDLO and BMP2LO.

I could also have saved these as individual 16 color BMPs or as individual 24-bit BMPs and BMP2LO would have merrily converted them. If I saved them as 24 bit BMPs I would have needed to fuss with setting exact colors from the supported palette. BMP2LO is set-up to map "similar" standard Windows colors to their LGR or DLGR equivalent as well as supporting AppleWin colors.

But ClipShop doesn't know about AppleWin colors and I bet neither do a lot of others, so it is quickest and easiest to just paste these clips into a 16 color BMP in Windows Paint, and use BMP2LO's 16 color mapping from standard Windows colors.

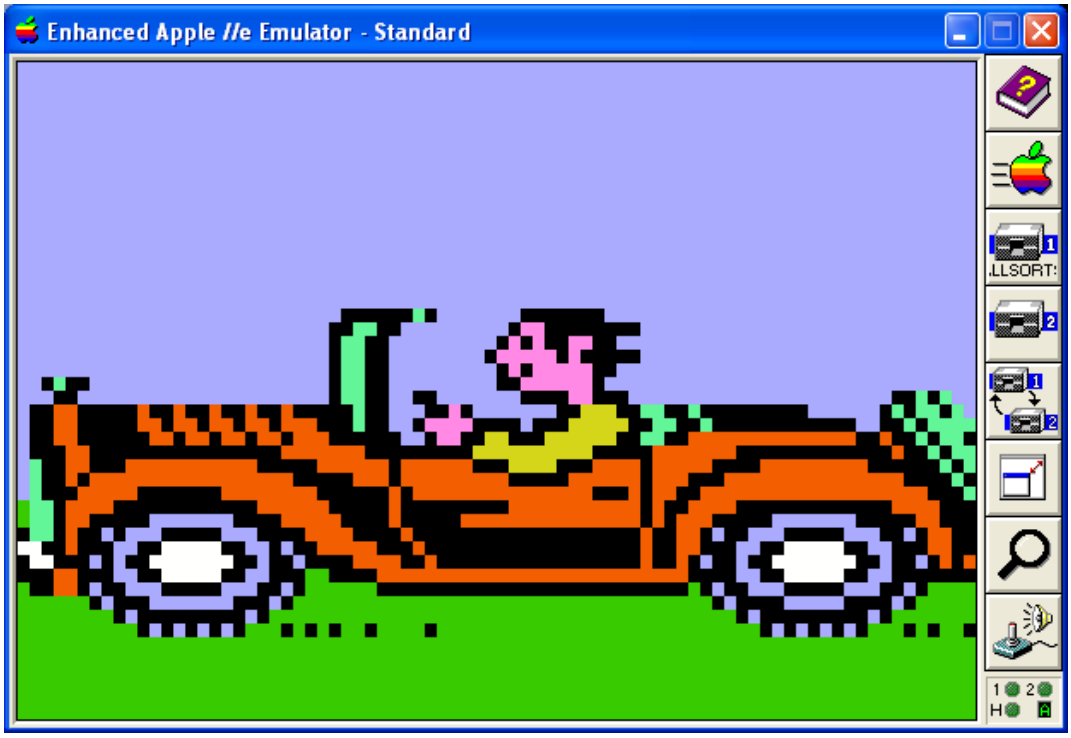
Minipix and Other "Big-Little Pictures" in 16 color Double Lo-res

Double Lo-res Graphics Mode for the Apple II is a pretty decent venue for displaying full-screen colored versions of Minipix (and other "little pictures"). Since Double Lo-res is 80 pixels x 48 rasters, and Minipix are 88 pixels x 52 rasters, this is close enough in size for the BMP2LO.EXE conversion utility to output these so they can be displayed on the Apple II in glorious full-screen detail and 16 discrete colors (after you color them). And since this utility works on Windows 16 color BMP files it is also easy enough to create original Double Lo-Res images of ones own design to be converted.

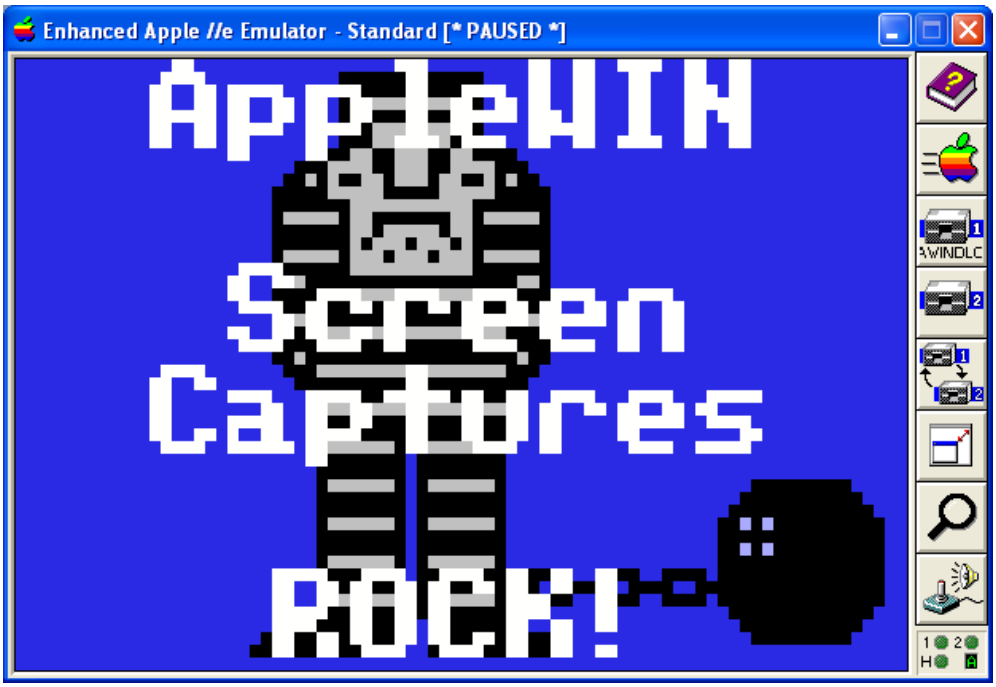
Graphics Files, Additional Resources, and Minipix in General

First off, the distribution for BMP2LO.exe includes several samples and examples to give you a starting point for its use whether you are coming from a programming perspective or just want to have fun with this or all of the above.

Over the years I have managed to accumulate a monstrous collection of Minipix which I distribute with my Clipshop program for Windows Users at **www.clipshop.ca** and which I hope you will take time to download and take full advantage of, if only for the Minipix alone. Clipshop comes with an extensive help file which is full of additional info about Minipix among other things.



DLGR and LGR Screen Captures In AppleWin



1. The **AppleWin Emulator** can be used to capture Color (standard) mode DLGR and LGR screens from a running Apple II program and save to a Windows BMP file. AppleWin has "hot keys" built-in to do this. The [Print Screen] "hot key" captures to a 560 x 384 x 256 Color BMP. This particular size is symmetrically "laid out" in 7 x 8 pixel (color) cells for DLGR, and 14 x 8 for LGR, which is perfect for conversion back to native Apple II DLGR Screen Files.

2. These color screen captures are enormous compared to the "coarse" resolution of the 80 x 48 x 16 Color DLGR screen or the 40 x 48 LGR screen. When AWINDLO converts an AppleWin screen capture to native Apple II DLGR or LGR files, it replaces the AppleWin screen capture file with a smaller 24 bit BMP suitable for editing in the same colors as AppleWin Version 1.22.

The top of the AWINDLO BMP is the 160 x 96 DHGR image and the bottom contains a "color bar palette" of the 16 available colors for editing. The "eyedropper" tool in Windows Paint can then be used to set the "draw color", without "mucking about" entering RGB values.

3. After editing, BMP2LO converts your finished work directly back to native Apple II DLGR or LGR screen files, which can either be titled in AWINDLO and then re-converted ad-infinitum, or used as-is.

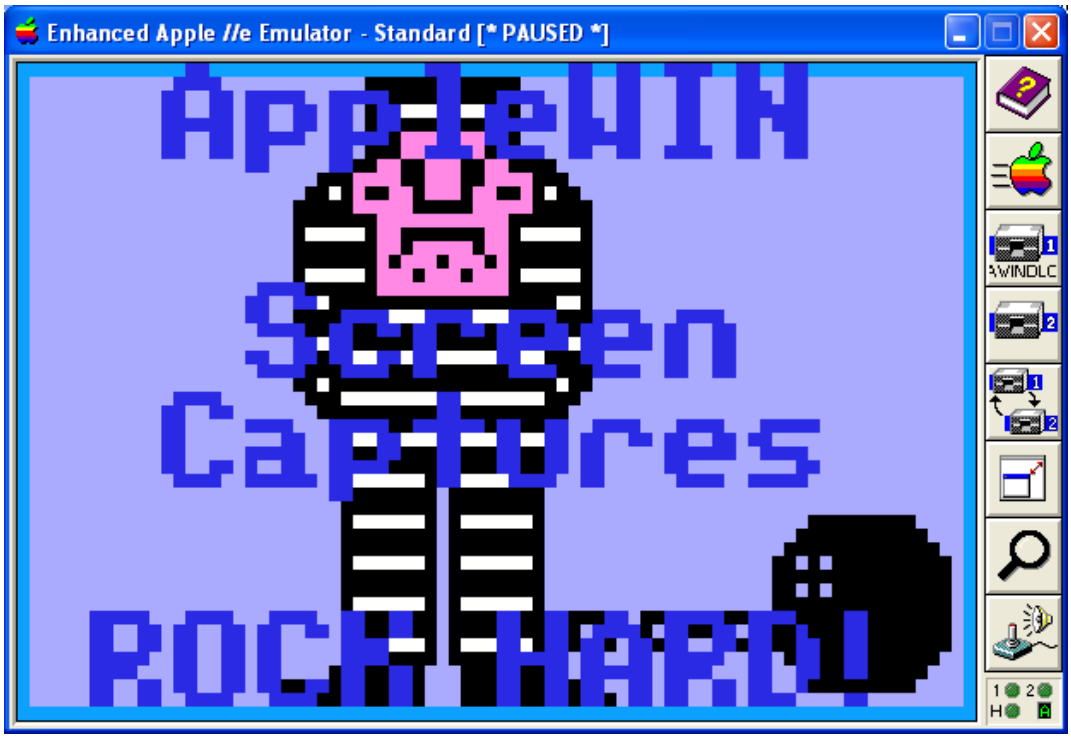
When converting back to Apple II DLGR or LGR files from the BMP with "color bar palette" using BMP2LO, the color bar palette area (at the bottom) is ignored and left alone as a color reference for future editing.

4. These "recycled" graphics can be placed back on an Apple II disk image again using utilities like **CiderPress**, and subsequently used in your own programs or slide-shows (or for whatever purpose you see fit).

Please consult the documentation for AppleWin, Windows Paint, CiderPress, and the documentation and source code for my own utilities for additional information.

Some Applications for Managing Apple II ProDOS Disk Images

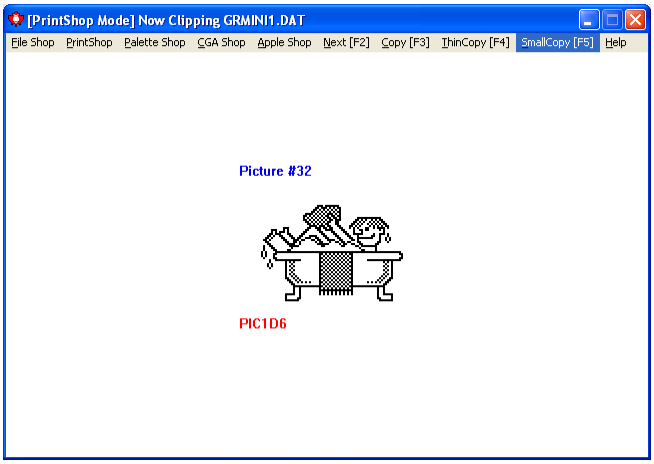
- **CiderPress** - No other Apple II Disk Manager has as many features or supports as many formats. Actively developed and supports CF (compact flash) storage etc.
- **Apple II Oasis Disk Manager** – Has "Batch Commands" and great support for DOS 3.3 disk images as well as ProDOS, but not actively developed.
- **AppleCommander** – Cross-Platform Support, written in Java and actively developed. Targeted towards Apple II Developers with support for the cc65 compiler, but can be used by casual Apple II enthusiasts as well.



Appendix A - Tutorial - Acquiring Minipix for BMP2LO

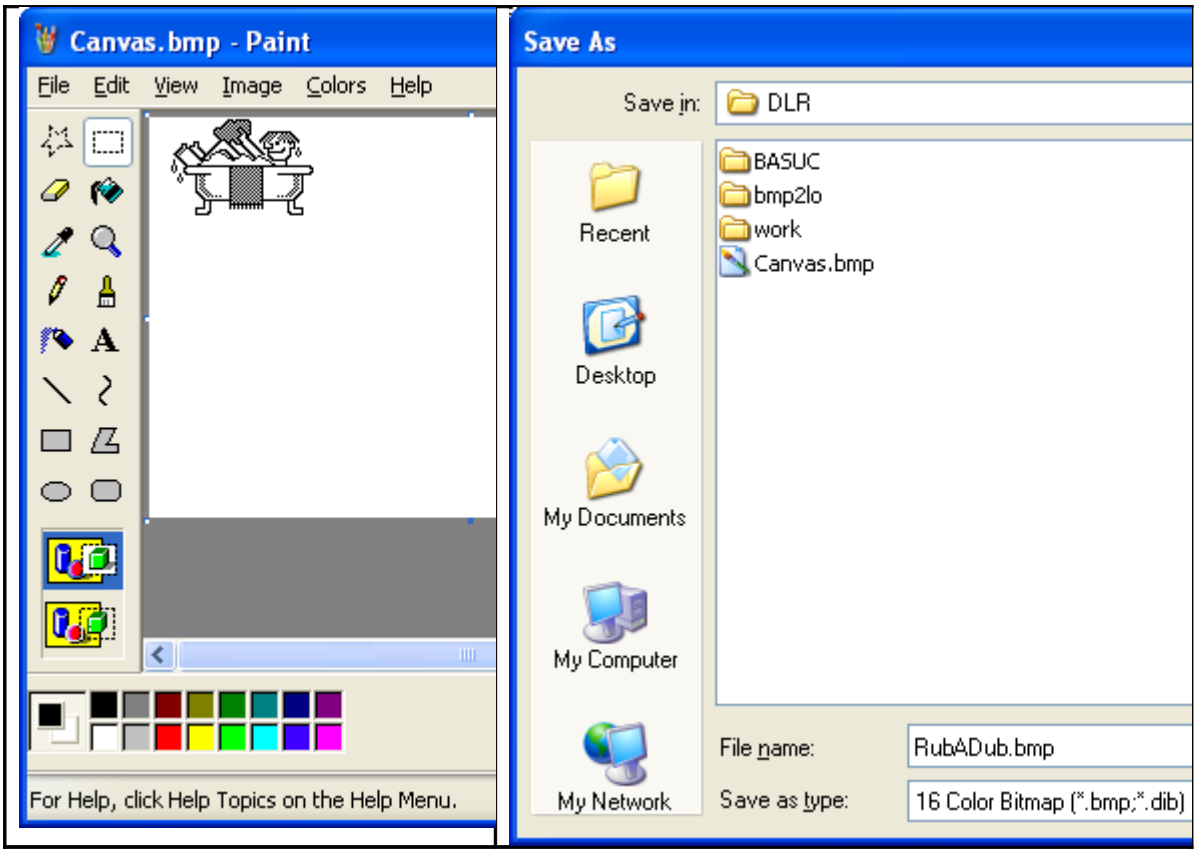
Step 1 – Copy the Graphic to the Windows Clipboard

Use **Clipshop** to load a Clipart Library, then **SmallCopy** the "Minipic":



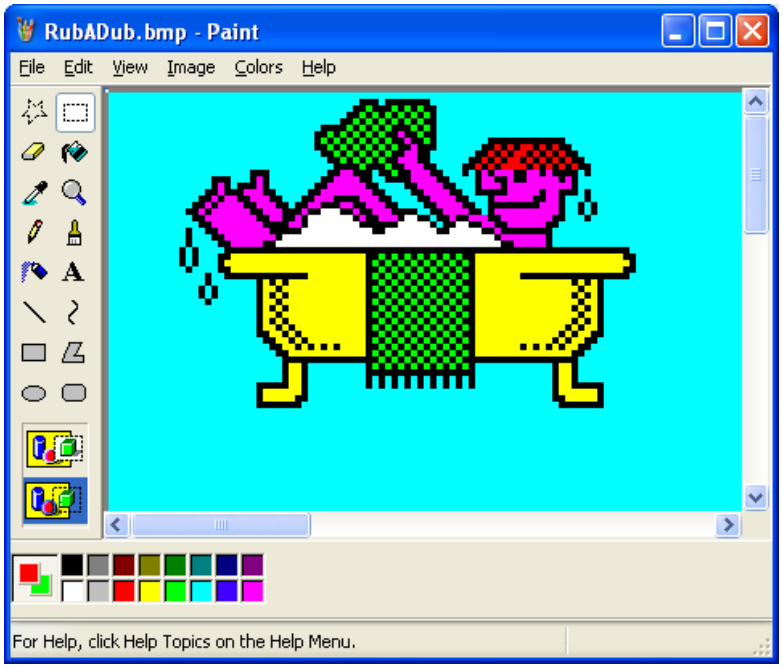
Step 2 – Paste the Graphic into Windows Paint

Open the Canvas.bmp that comes with BMP2LO.exe with Windows Paint, paste the Minipic, and save as a 16 color bmp with an 8 character filename:

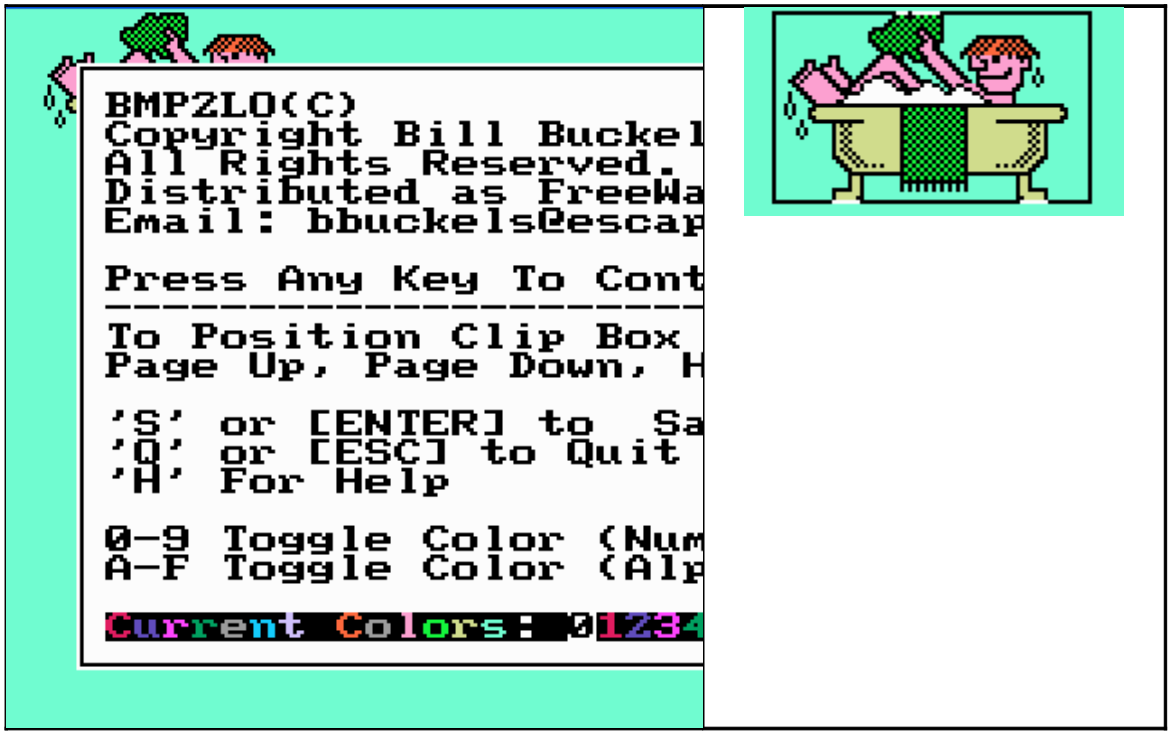


Step 3 – Color and Edit the Graphic in Windows Paint

Use “fat bits” (zoom large size) to color and edit the Minipic, then save when done:

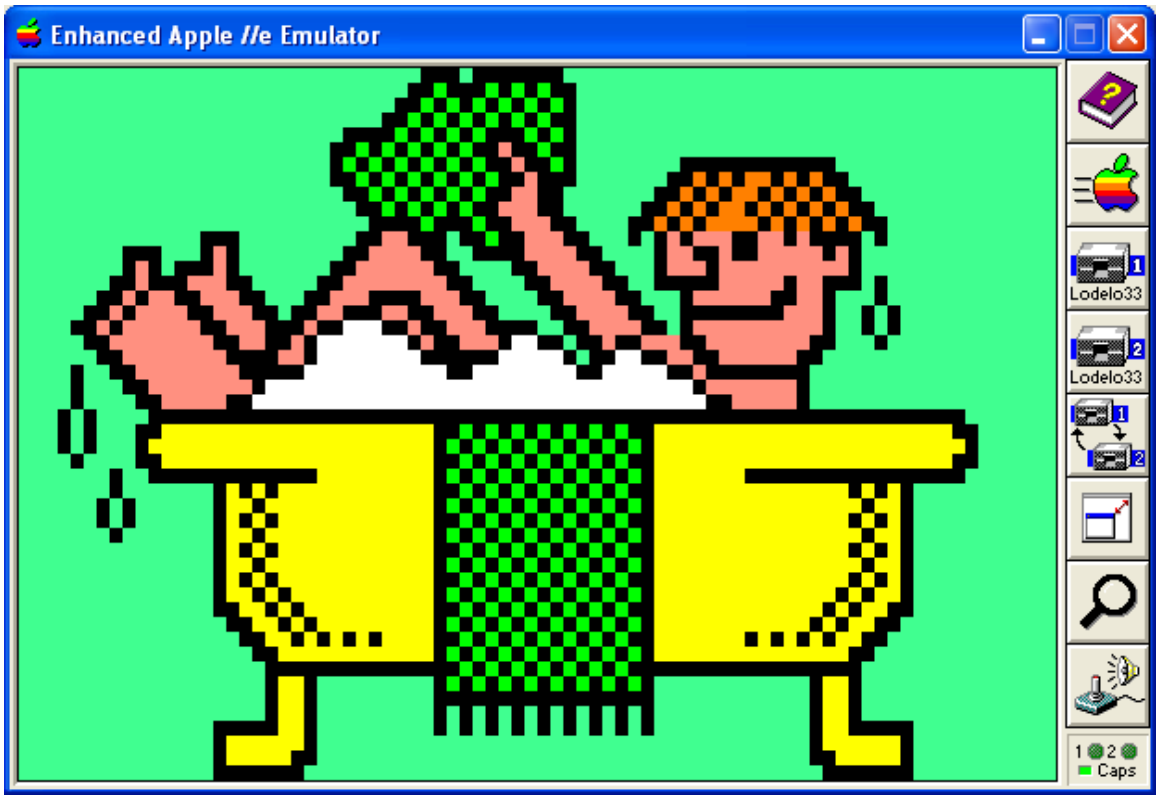


Step 4 – Use BMP2LO to Convert to Apple II format



Step 5 – Display the Graphic on Your Apple II

Transfer the Apple II DLOGR output file(s) created by BMP2LO.exe (either the DL1 and DL2 files or the DLO file) to an Apple II Disk or Disk Image and Display on your Apple II or in your Emulator in Double Lo-Res Mode:



Appendix B - DOS 3.3 BASIC Example

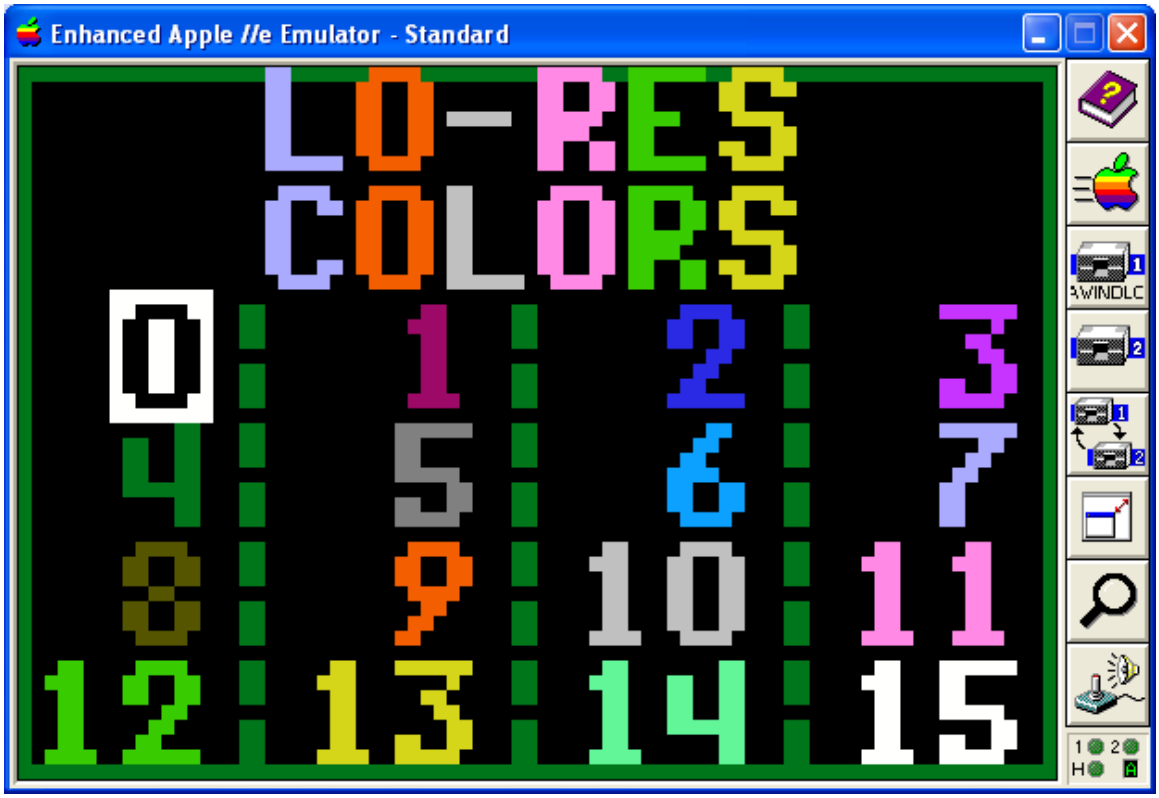
```
1   REM Simple Double Lo-Res Image Loader Example
2   REM By Bill Buckels, December 2012
3   D$ = CHR$(4)
4   PRINT D$;"PR#3" : PRINT : REM ENABLE 80 COL
5   TEXT
6   HOME
10  PRINT "*********************************"
11  PRINT "*                               *"
12  PRINT "*       DOUBLE LO-RES MENU      *"
13  PRINT "*                               *"
14  PRINT "* BASIC Demo                    *"
16  PRINT "* 1 -   BLOAD EXAMPLE           *"
17  PRINT "* ESC - EXIT TO DOS 3.3         *"
18  PRINT "*                               *"
19  PRINT "*********************************"
150   GET A$
160   IF A$ =  CHR$ (27) THEN  GOTO 800
170   IF A$ = "1" THEN  GOTO 400
200 GOTO 5
400   POKE 49246,0 : REM DOUBLE LO-RES ON
410   POKE 49238,0 : REM LO-RES ON
430   POKE 49232,0 : REM GRAPHICS
440   POKE 49234,0 : REM FULL GRAPHICS
455   PRINT D$;"PR#3" : PRINT : REM ENABLE 80 COL
460   CALL -1998 : REM CLEAR PAGE ONE
500   POKE 49237,0 : CALL -1998 : REM CLEAR PAGE TWO
510   PRINT D$;"BLOAD CROOK.DL1,A$400" : PRINT
520   POKE 49236,0 : REM PAGE ONE
530   PRINT D$;"BLOAD CROOK.DL2,A$400" : PRINT
700 GET A$
705   CALL -1998 : REM CLEAR PAGE ONE
710   POKE 49247,0 : REM DOUBLE LO-RES OFF
720   POKE 49233,0 : REM TEXT MODE
730   POKE 49236,0 : REM PAGE ONE
740   PRINT D$;"PR#3" : PRINT : REM ENABLE 80 COL
750 GOTO 5
800 TEXT
810 HOME
820 CALL  - 1184: CALL 42350
830 REM  THIS IS THE END
840 NEW
```

Please note that 2 files are loaded in the BASIC example above:

1. DL1 – The “Double-Lo 1” file is the Auxiliary Memory File
2. DL2 – The “Double-Lo 2” file is the Main Memory File

These 2 files are modeled after the AUX and BIN files that were commonly used in Apple DHGR BASIC programs and are each 1016 bytes in length and are simply 2 chunks of BSaved binary data.

Appendix C – Understanding DLGR Color Implementation



16 Colors – 4 bits per pixel

There are 16 – Colors on the Apple II DLGR screen. Each one has its own 4-bit value which is spread alternately across Auxiliary and Main Memory (Banks 1 and 0) in increments of 1 pixel, even pixels in bank 1 and odd pixels in bank 0.

2 Vertical Pixels per Byte

Two lines of color plot on the DLGR display at the same time when we put a byte into screen memory. This because 2 colors (4-bit pixels) fit into one byte and you can't "split" a byte across 2 different addresses like bank0 and bank1, both because a byte is the lowest common denominator of memory address storage, and due to "bank-switching" on the Apple II, each shares the same address. Also the DLGR memory address is the same as the text screen which of course also inherently works in bytes, and must work co-operatively with DLGR and vice-versa.

Screen Resolution 80 x 48 (full graphics) or 80 x 40 (mixed mode)

The amount of "space" for colors on DLGR is the storage size of an 80 column text screen… each byte is 2 colors (pixels) high and each line is 80 colors (columns, pixels) across.

This provides DLGR with a vertical resolution of 48 “pixel lines” in height in “full graphics” mode and 40 “pixel lines” in “mixed text and graphics” mode. Unlike in the HGR display memory which can hold a “full screen” graphic while we merrily switch “mixed” and “full” on and off, because the text and graphics in DLGR (and LGR) share exactly the same memory, graphics images used in LO-RES “mixed” modes (DLGR and LGR) must either be only 40 “pixel lines” (20 bytes) in height or the bottom 8 “pixel lines” (4 bytes in height, 4 lines of text) of a DLGR image will be “trashed” in mixed mode, and will trash your text if any.

Pixel Packing in DLGR – Even and Odd Scan-lines

As previously noted above, DLGR colors are stored 2 vertical pixels per byte. When you write a byte to DLGR (or LGR) screen memory, two pixels appear. When you write 80 bytes to DLGR two scan-lines (rows) of 80 pixels appear. The even rows are the 4-bit color values (pixels) in the “low nibble” of the bytes, and the odd rows are the 4-bit values in the “high nibble” of the bytes.

Also as previously noted, even pixels are in auxiliary memory (bank 1) and odd pixels are in main memory (bank 2). . Each horizontal scan-line is 80 bytes, with 40 bytes in Bank 0 alternating with 40 bytes in Bank 1. This is a “neater” pixel arrangement than in DHGR mode which splits 7 pixels over 4 bytes straddling bank 1 and bank 0. However, low level “bit twiddling” is still needed to make DLGR work (as it is in all Apple II graphics modes).

The pixel-packing scheme that DLGR uses and the colors it uses are the same as normal Lo-Res (LGR) except with a variation on color indices (color numbers) when it comes to Bank 1 (which we will discuss next).

Color Indices for Auxiliary Memory - Rotating the Bits of the Nibbles

The “packed pixel” colors for DLGR main memory (bank 0) use the same color indices (color numbers) as normal Lo-Res. In a manner of speaking, auxiliary memory uses the same numbers too, but they are stored in memory differently; when writing a Lo-Res color index number to bank1 (auxiliary memory), a right **rotate no carry** (right circular shift) of the 4-bit color number is needed, and when reading a Lo-Res color number from bank 1, a left **rotate no carry** (left circular shift) is needed.

In English, a **circular shift** is the operation of rearranging the entries in a “tuple”, (in the case of moving a pixel between main and auxiliary memory, the four-tuple of bits in the Lo-Res color or color nibble), either by moving the final entry to the first position, while shifting all other entries to the next position, or by performing the inverse operation.

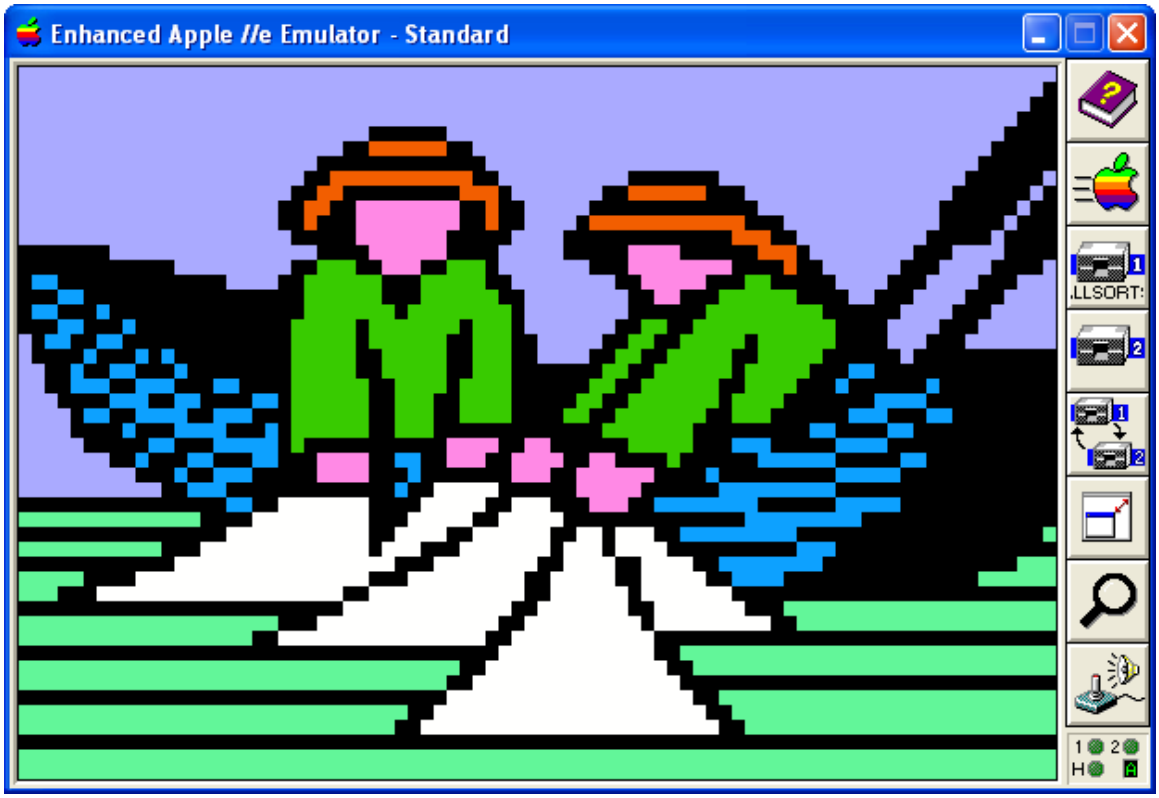
The Lo-Res plotting routines in the Apple II ROM can be used in DLGR as well as in Lo-Res mode. However when using those routines we need to rotate the Lo-Res color number as we do with our nibbles when addressing bank 1 memory.

Tables to Convert DLGR Colors between Main and Auxiliary Memory

LUT's (look-up tables) simplify production code and improve its readability. The memory a small table takes is more than compensated for by avoiding repetitive codified instructions, which can use more memory, diskspace, and be slower to process. This might not be important on today's monsters, but the Apple II is a different story.

```
/* the following is used to remap double lo res 4 bit colors
from bank 0 to bank 1 */
unsigned char dloauxcolor[16] = {
0,8,1,9,2,10,3,11,4,12,5,13,6,14,7,15};

/* the following is used to remap double lo res 4 bit colors
from bank 1 back to bank 0 */
unsigned char dlomaincolor[16] = {
0,2,4,6,8,10,12,14,1,3,5,7,9,11,13,15};
```



Program Listing – DLGR Conversion Table Generator

```
int main()
{
    int color;
    unsigned char nib,big,bit;

    /* produce conversion tables for DLGR color indices */
    puts("/* the following is used to remap\n"
         "double lo res 4 bit colors\n"
         "from bank 0 to bank 1 */\n"
         "unsigned char dloauxcolor[16] = {");

    for (color=0; color< 16; color++) {
        /* right circular shift */
         nib = (unsigned char )color;
        big = (nib >> 1);
        bit = (nib &1) << 3;
        nib = (big | bit);

        if (color) putchar(',');
         printf("%d",(int)nib);
    }
    puts("};");

    puts("/* the following is used to remap\n"
         "double lo res 4 bit colors\n"
         "from bank 1 back to bank 0 */\n"
         "unsigned char dlomaincolor[16] = {");

    for (color=0; color< 16; color++) {
        /* left circular shift */
         nib = (unsigned char )color;
         big = (nib << 1) &0xF;
         bit = (nib >> 3) &1;
         nib = (big | bit);

        if (color) putchar(',');
         printf("%d",(int)nib);

     }
     puts("};");

    return 0;
}
```

The working code above produced the DLGR color conversion tables listed earlier. The programming example that follows demonstrates how the conversion table is used on the Apple II. These tables are also used for color conversion in AWINDLO and BMP2LO.

Programming Example – Plotting a DLGR Pixel using Apple II Rom Routines

```
/* bank 0 color remapping to bank 1 color */
unsigned char dloauxcolor[16] = {
     0,8,1,9,2,10,3,11,4,12,5,13,6,14,7,15};
#define XREG  0
#define YREG  1
#define COLOREG 0
#asm
instxt <zpage.h>
XVAL equ  REGS
YVAL equ  REGS+1
COLOR equ REGS
#endasm
unsigned char *dlobyteregptr = (unsigned char *)0x80;
dloplot(x, y, color)
{
  int w, z = x;
/* Double Lo-Res works the same way 80-column text does:
   columns 0, 2, 4, ...78 are stored in AuxRAM,
   and columns 1, 3, 5, ...79 are stored in MainRAM. */
   x = z / 2;
   w = x * 2;
   if (z==w)  dlobyteregptr[COLOREG] = dloauxcolor[color];
   else dlobyteregptr[COLOREG] = (unsigned char)color;
#asm
  LDA COLOR; Sets the plotting color to N, 0 <= N <= 15
  JSR $F864
#endasm
   if (z!=w)
  {
#asm
     sta  $c054
#endasm
  }
  else {
#asm
     sta  $c055
#endasm
  }
  /* load parameters into user regs */
  dlobyteregptr[XREG] = x;
  dlobyteregptr[YREG] = y;
  /* make ml call */
#asm
  LDY XVAL ; Lo-Res Plot X (Horizontal) Coordinate (0-39)
  LDA YVAL ; Lo-Res Plot Y (Vertical) Coordinate (0-39)
  JSR $F800
#endasm
}
```

The above routine is written in Aztec C65 for Apple IIe ProDOS. It is a working code excerpt from the AppleX MS-DOS/Windows cross-compiler distribution.