USC Viterbi School of Engineering

Ming Hsieh Department of Electrical Engineering

EE 459Lx - Embedded Systems Design Laboratory

Converting Binary Images to Program Data

by Allan G. Weber

Fall 2006

This document describes a method for converting a binary (1-bit/pixel) image into a set of C and assembly language data initialization statements. Once the binary image is stored in the program as a data array, the program can work with the image data to do things like download it into a graphics device. The procedures below are only for 1-bit/pixel black and white images and do not apply to grayscale or color images with more than one bit per pixel.

Create Image File

The first step is to use a image editing program such as Windows "Paint" or Adobe Photoshop to create an image of whatever is needed for the project. In the image editor, create a new blank image. If the program allows you to set the image depth (bits/pixel) when creating the blank image set it to one bit per pixel. This mode may be designated as "Monochrome" or "Bitmap". For example, with Adobe Photoshop in the new image dialog, you can use the Color Mode menu to set it to "Bitmap". Set the width and height of the image large enough to accomodate whatever it is you are going to draw. These can be reduced later if necessary. In this example we will be working with an image 40 pixels wide by 16 pixels high.

Use the various drawing or text tools to fill the image with whatever contents you wish to have.

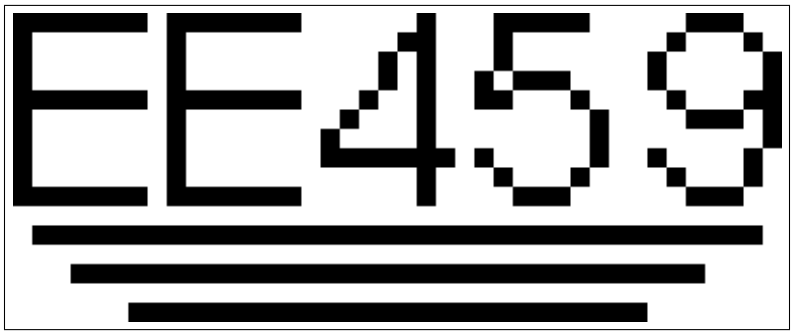

Figure 1: 40 by 16 pixel bitmap image

The image can be drawn in either black-on-white or white-on-black since the conversion program can invert the image data when converting it.

Saving In Compatible Format

Once the image has been created, the next step is to save it in a format that is compatible with the program that translate the image into C or assembler program data. The conversion program used in EE459L works with input files in Windows BMP format. This is the native file format for the Window "Paint" program, and other programs like Photoshop can save images in this format.

The Paint program has to be told when saving the image to save it as a 1-bit/pixel image. In the "Save as" dialog box, use the "Save as type" menu to set the format to "Monochrome bitmap". Regardless of what

program is used, it is important that the image be saved as a 1-bit/pixel BMP image since this is the only format our conversion program currently understands.

Conversion Program

The BMP files are converted to C or assembler data using a C program available on the class web site. The program (`bmp2hex.c`) reads a 1-bit/pixel BMP file as input and outputs a set of hexadecimal values. This program should be usable on any system with a C compiler. Simply compile and link the program. It needs to be run from a command line interface.

The hexadecimal numbers the program outputs contain the pixel values with eight pixels in each byte. The bytes are output in the order top line left to right, second line left to right, ..., bottom line left to right. Each line of the image produces one line of hexadecimal values on the output. Within each 2-digit hexadecimal number the pixels are stored with the left most pixel in the most significant bit and the right most pixel in the least significant bit. Each line of the image starts with a new byte so if the number of bits in a line is not a multiple of eight, there will be extra bits in the last byte of each line that are not part of the image. In summary, the MSB of the first byte is the upper left corner, the LSB of the last byte is the lower right corner (assuming all eight bits of the byte are used.)

The following shows the `bmp2hex` program being run on the BMP image file in Fig.1.

```
% bmp2hex < Bitmap_Image.bmp
0xfe, 0xfe, 0x04, 0x7c, 0x1c,
0x80, 0x80, 0x0c, 0x40, 0x22,
0x80, 0x80, 0x14, 0x40, 0x41,
0x80, 0x80, 0x14, 0xb8, 0x41,
0xfe, 0xfe, 0x24, 0xc4, 0x23,
0x80, 0x80, 0x44, 0x02, 0x1d,
0x80, 0x80, 0x84, 0x02, 0x01,
0x80, 0x80, 0xfe, 0x82, 0x42,
0x80, 0x80, 0x04, 0x44, 0x22,
0xfe, 0xfe, 0x04, 0x38, 0x1c,
0x00, 0x00, 0x00, 0x00, 0x00,
0x7f, 0xff, 0xff, 0xff, 0xfe,
0x00, 0x00, 0x00, 0x00, 0x00,
0x1f, 0xff, 0xff, 0xff, 0xf0,
0x00, 0x00, 0x00, 0x00, 0x00,
0x03, 0xff, 0xff, 0xff, 0x80,
%
```

Since each line of the image contains 40 pixels, the program outputs five 8-bit hexadecimal numbers for each of the 16 lines.

It is possible that the image data will come out of the program inverted from what you expected. The example above has white pixels as zeros and black pixels as ones. If you want it the other way, run the program with the "`-i`" option to invert all the output data.

```
% bmp2hex -i < Bitmap_Image.bmp
0x01, 0x01, 0xfb, 0x83, 0xe3,
0x7f, 0x7f, 0xf3, 0xbf, 0xdd,
.
.
.
```

Now the white pixels are ones and the black pixels are zeros.

C and Assembler Initialization Code

The output of the `bmp2hex` program can be copied and pasted into a C or assembler program with only slight modifications in order to initialize a data array.

```
const unsigned char myimage[80] = {
        0xfe, 0xfe, 0x04, 0x7c, 0x1c,
        0x80, 0x80, 0x0c, 0x40, 0x22,
        0x80, 0x80, 0x14, 0x40, 0x41,
        0x80, 0x80, 0x14, 0xb8, 0x41,
        0xfe, 0xfe, 0x24, 0xc4, 0x23,
        0x80, 0x80, 0x44, 0x02, 0x1d,
        0x80, 0x80, 0x84, 0x02, 0x01,
        0x80, 0x80, 0xfe, 0x82, 0x42,
        0x80, 0x80, 0x04, 0x44, 0x22,
        0xfe, 0xfe, 0x04, 0x38, 0x1c,
        0x00, 0x00, 0x00, 0x00, 0x00,
        0x7f, 0xff, 0xff, 0xff, 0xfe,
        0x00, 0x00, 0x00, 0x00, 0x00,
        0x1f, 0xff, 0xff, 0xff, 0xf0,
        0x00, 0x00, 0x00, 0x00, 0x00,
        0x03, 0xff, 0xff, 0xff, 0x80
        };
```

For assembler the following should work.

```
myimage: db     0xfe, 0xfe, 0x04, 0x7c, 0x1c
         db     0x80, 0x80, 0x0c, 0x40, 0x22
         db     0x80, 0x80, 0x14, 0x40, 0x41
         db     0x80, 0x80, 0x14, 0xb8, 0x41
         db     0xfe, 0xfe, 0x24, 0xc4, 0x23
         db     0x80, 0x80, 0x44, 0x02, 0x1d
         db     0x80, 0x80, 0x84, 0x02, 0x01
         db     0x80, 0x80, 0xfe, 0x82, 0x42
         db     0x80, 0x80, 0x04, 0x44, 0x22
         db     0xfe, 0xfe, 0x04, 0x38, 0x1c
         db     0x00, 0x00, 0x00, 0x00, 0x00
         db     0x7f, 0xff, 0xff, 0xff, 0xfe
         db     0x00, 0x00, 0x00, 0x00, 0x00
         db     0x1f, 0xff, 0xff, 0xff, 0xf0
         db     0x00, 0x00, 0x00, 0x00, 0x00
         db     0x03, 0xff, 0xff, 0xff, 0x80
```