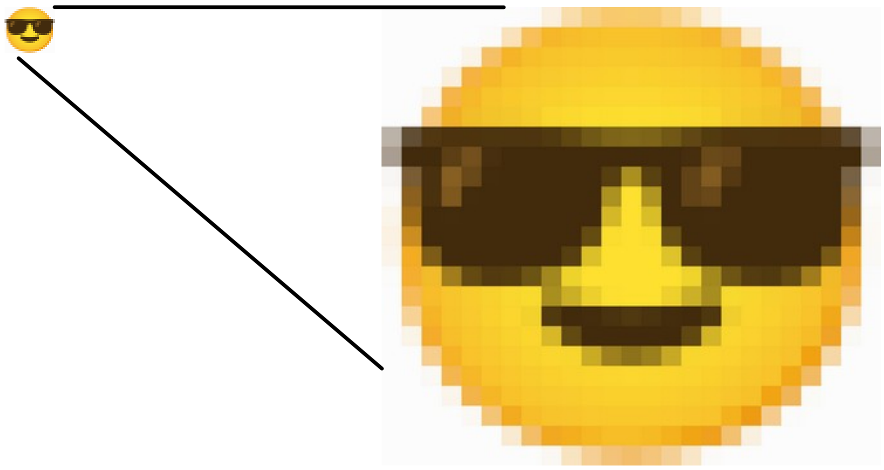
 English name: ____________________

Document: Reading a Bitmap File

©2025 Chris Nielsen – *www.nielsenedu.com*

In computer graphics and digital photography, a ***raster graphic*** represents a two-dimensional picture as a rectangular matrix or grid of pixels. Raster images have a fixed width and height, so lose quality when scaled. For example, when we view the small “Smiling Face with Sunglasses” icon at normal size, it loos clear, but if we enlarge the image, it becomes blocky.



Common raster image file formats include files that use the file extensions `jpg`, `png`, `gif`, and `bmp`.

This document describes the *Microsoft’s BMP (Bitmap) format* that was introduced with Windows 2.0 operating system in 1987. Despite its drawbacks, and after some evolution, the format is still commonly used today.

Below is a hexadecimal dump of the bits that make up a `bmp` format file that we will examine and manually interpret to reveal the image it describes.

Table 1: Example `bmp` *File Hexadecimal Dump*

	Least Significant Nibble of Address															
Address of Leftmost Byte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000000	42	4d	ea	00	00	00	00	00	00	00	92	00	00	00	7c	00
0000010	00	00	31	00	00	00	0b	00	00	00	01	00	01	00	00	00
0000020	00	00	58	00	00	00	13	0b	00	00	13	0b	00	00	02	00
0000030	00	00	02	00	00	00	00	f8	00	00	e0	07	00	00	1f	00
0000040	00	00	00	00	00	00	42	47	52	73	00	00	00	00	00	00
0000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000070	00	00	00	00	00	00	00	00	00	00	02	00	00	00	00	00
0000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	ff	ff
0000090	ff	00	ff	ff	ff	ff	ff	ff	80	00	ff	ff	ff	ff	ff	ff
00000A0	80	00	dd	ae	30	e3	76	3d	80	00	dd	55	ff	5d	55	ff
00000B0	80	00	c1	54	18	dd	54	1d	80	00	dd	75	d7	dd	55	dd
00000C0	80	00	dd	76	38	e3	2e	3d	80	00	eb	ff	ff	ff	ff	fd
00000D0	80	00	f7	ff	ff	ff	ff	fd	80	00	ff	ff	ff	ff	ff	ff
00000E0	80	00	ff	ff	ff	ff	ff	ff	80	00						

Document: Reading a Bitmap File

©2025 Chris Nielsen – *www.nielsenedu.com*

A bmp file format is made up of the following parts:

- Bitmap file header
- Device-Independent Bitmap (DIB) header
- Color table (optional)
- Image data pixel array (a two-dimensional grid)
- International Color Consortium (ICC) color profile

We will go through each of these to decode our image.

Bitmap File Header

The *bitmap file header* contains a signature, a file length, and an offset to where the image data pixel array starts. There are also two fields that were originally reserved for future use, but should generally be set to zero.

The diagram below shows the fields of the bitmap file header. The first row numbers the byte offset into the bitmap file. The second row shows what the byes contain. The third row shows the first 14 bytes of data copied from the example hexadecimal dump of our example bitmap file.

byte	0	1	2	3	4	5	6	7	8	9	A	B	C	D
use	signature		file size				reserved1		reserved1		file offset to pixel array			
data	42	4d	ea	00	00	00	00	00	00	00	92	00	00	00

Signature

The first two bytes make up the ***signature*** field. In the example data, these bytes contain the hexadecimal values `0x42` and `0x4D`. As can be seen from the table of ASCII characters, below, these values correspond to the letter `B` and the letter `M`. All valid `bmp` (bitmap) files will begin with these ASCII character codes – `BM`, for bitmap.

ASCII Character Set (0x20-0x7F)

hex	char	hex	char	hex	char	hex	char	hex	char	hex	char
20	*space*	30	0	40	@	50	P	60	`	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	“	32	2	**42**	**B**	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	'	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	-	3D	=	**4D**	**M**	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	*delete*

File Size

The next field is the 4-byte ***file size***. Since 4 bytes represents any value from 0 to 4,294,967,295. This means the theortical maximum size of a `bmp` file is `4` GiB. This would be a massive image, so bitmaps generally don't even come close to that size.

Or example bitmap file hexadecimal dump is not long, but when we look at the number in the *file size* field, we see it looks like a large number: `ea 00 00 00`.

This is because the field is saved in ***little-endian*** format. For *little-endian*, the values are written from ***least significant byte*** to ***most significant byte***. Thus, if we write the number as a 32-bit value, the `ea` goes on the right.

For our example `bmp` file, the file size is: `00 00 00 ea`. Converting `0xEA` to decimal, we have the denary value 234. There are 234 bytes in this bitmap file. Rather than count the bytes, if we examine the hexadecimal dump given in Table 1 on page 1, we see that the addresses of the bytes run from `0` to `0xE9`, suggesting our interpretation of the *file size* field is correct, and thus far the `bmp` file looks valid.

File Offset to Pixel Array

The field named ***file offset to pixel array*** is also a 4-byte value, and our example bitmap file has following values in these bytes: `92 00 00 00`. Writing this *little-endian* byte stream into a 4-byte value (reversing the bytes), we get an offset of `00 00 00 92`. When we determine the values of the pixels of our image we will use this offset to find the start of the image pixel values.

Device-Independent Bitmap (DIB) Header

As the `bmp` format evolved, different versions of the ***device-independent bitmap* (*DIB*)** header have been developed. Which header a `bmp` files uses is determined by its length, which is given in the first 4 bytes of the DIB header. The bitmap file header finished at address `0x0D`, so the DIB header size runs from byte address `0x0E` to `0x11`, inclusive.

Verify for yourself that the bytes values from the hexadecimal dump of our example `bmp` file is: `7c 00 00 00`, and converting from little-endian gives us a 32-bit value of `00 00 00 7c`. This value in decimal is 124 bytes. This is the length of the most recent version of the header – version 5 of the bitmap header. The first fields of this header is given in the table below, with the values from our example bitmap given on the right hand side.

	Device-Independent Bitmap (DIB) Header		Example Bitmap	
0E	DIB header size		0000007c	
12	image width		00000031	
16	image height		0000000b	
1A	planes	bits per pixel	0001	0001
1E	compression		00000000	
22	image size		00000058	
26	x pixels per meter		00000b13	
2A	y pixels per meter		00000b13	
2E	colors in color table		00000002	
32	important color count		00000002	
36	red channel bitmask		0000f800	
3A	green channel bitmask		000007e0	
3E	blue channel bitmask		0000001f	
42	alpha channel bitmask		00000000	
46 ⋮ 86	other fields			

Image Width and Image Height

The image width is `0x31`, or 49 pixels, and the image height is `0xB`, or 11 pixels high. The image dimensions (width x height) is: 49x11 pixels.

Bits Per Pixel

The value of *bits per pixel* is `1`. This means each pixel can only be one of two possible colors. One color will be represented by a bit value of `0`, and the other color will be represented by the bit value of `1`. Which two colors are present in the image will be given in the ***color table***, which comes after the DIB header.

Image Size

This is the size of the image data pixel array in bytes. The pixel array in our example image is `0x58` (denary 88). From our pixel array offset of `0x92`, we can calculate the end of the pixel array to be at: `0x92 + 0x58 = 0xEA`. This is exactly the end of the file.

 ©2025 Chris Nielsen – *www.nielsenedu.com*

Recommended Resolution (Pixels Per Meter)

Although it is often ignored when deciding the size to display an image on a screen, it may be used for such things as ensuring the correct size for printing, so we will show how to determine the recommended size of the image given the recommended resolution in pixels per meter in the DIB header.

The fields *x pixels per meter* and *y pixels per meter* allow us to calculate what physical size the image is recommended to be displayed as. In the case of our example, there is expected to be `0xb13` pixels per meter. As a denary number, this is 2835 pixels. We can use this number to calculate the size of the image:

width: $49\, pixels \div 2835 \frac{pixels}{meter} \approx 1.73\, cm$

height: $11\, pixels \div 2835 \frac{pixels}{meter} \approx 0.4\, cm$

This image is intended to be displayed in a very small space. Considering how few pixels the image has, this is not surprising. If it is displayed much larger, the image will appear blocky (which we will see when we recreate the image).

Color Table

The DIB header gives us a value for the number of *colors in color table*, which for our example is `2`. A color table associated with version 5 of the DIB header will use `4` bytes to describe each color in the table.

The final 4 bytes of the DIB header started at offset `0x86` into the `bmp` file. The color table will begin after these 4 bytes, so at offset `0x8A`. Reading our example file, we find the following values for the colors in the color table (remember we need to convert from little-endian).

File offset	Index	Color Value	Color
`0x8A`	`0`	`0x0000_0000`	`black`
`0x8E`	`1`	`0x00FF_FFFF`	`white`

The color values are stored in this table as `ARGB`. We will discuss what this means very soon in the course, but for now, just know that for our example `bmp` file, color `0` is black and color `1` is white.

We will talk about how colors are represented in a later lecture, but for our assignment, you will be asked to name the color. Please use the HTML/CSS standard color names for the colors in your bitmap file.

RGB value	Color name
`00 00 00`	black
`00 00 80`	navy
`00 00 FF`	blue
`00 80 00`	green
`00 80 80`	teal
`00 FF 00`	lime
`00 FF FF`	cyan

RGB value	Color name
`80 00 00`	maroon
`80 00 80`	purple
`80 80 00`	olive
`80 80 80`	gray
`A5 2A 2A`	brown
`C0 C0 C0`	silver

RGB value	Color name
`FF 00 00`	red
`FF 00 FF`	magenta
`FF A5 00`	orange
`FF C0 CB`	pink
`FF D7 00`	gold
`FF FF 00`	yellow
`FF FF FF`	white

©2025 Chris Nielsen – *www.nielsenedu.com*

Image Data Pixel Array

The start of the pixel data is given in the *bitmap file header*, and for our example, the value is `0x92`. We know that our example image uses `1` bit per pixel, and the image is `49` pixels in width. The `bmp` file format requires that each row of pixels in the pixel array must be a multiple of 4 bytes (32 bits). 4 bytes is not enough to store 49 bits, but 8 bytes (64 bits) is enough. Thus, for our example bitmap file, each row in the pixel array is 8 bytes in length.

The table below shows the original `bmp` file with the bitmap file header, the device-independent bitmap header, the color table, and the image data pixel array highlighted. White space is left between each field of the headers, between each color of the color map, and between each row of the pixel array. It can be seen that there are 11 rows in the pixel array. This is as we expect given the image height from the DIB header is 11 pixels high. Again, the file looks like a valid `bmp` file.

Table 2: Example `bmp` *File Hexadecimal Dump With Color Coding of Sections*

	Least Significant Nibble of Address															
Address of Leftmost Byte	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000000	42	4d	ea	00	00	00	00	00	00	00	92	00	00	00	7c	00
0000010	00	00	31	00	00	00	0b	00	00	00	01	00	01	00	00	00
0000020	00	00	58	00	00	00	13	0b	00	00	13	0b	00	00	02	00
0000030	00	00	02	00	00	00	00	f8	00	00	e0	07	00	00	1f	00
0000040	00	00	00	00	00	00	42	47	52	73	00	00	00	00	00	00
0000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000070	00	00	00	00	00	00	00	00	00	00	02	00	00	00	00	00
0000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	ff	ff
0000090	ff	00	ff	ff	ff	ff	ff	ff	80	00	ff	ff	ff	ff	ff	ff
00000A0	80	00	dd	ae	30	e3	76	3d	80	00	dd	55	ff	5d	55	ff
00000B0	80	00	c1	54	18	dd	54	1d	80	00	dd	75	d7	dd	55	dd
00000C0	80	00	dd	76	38	e3	2e	3d	80	00	eb	ff	ff	ff	ff	fd
00000D0	80	00	f7	ff	ff	ff	ff	fd	80	00	ff	ff	ff	ff	ff	ff
00000E0	80	00	ff	ff	ff	ff	ff	ff	80	00						

Color key: Bitmap file header | DIB Header | Color Table | Image Data Pixel Array

Document: Reading a Bitmap File

©2025 Chris Nielsen – *www.nielsenedu.com*

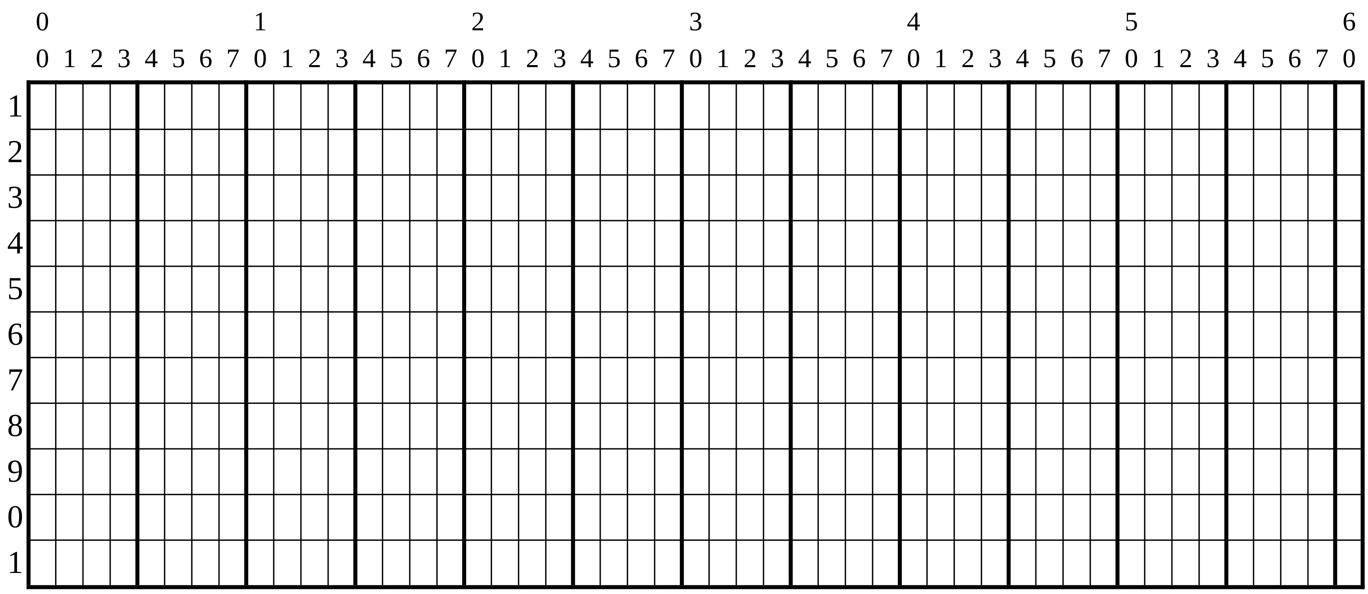
The table below has copied to each pixel array row onto it's own row. It has also placed the first row from the `bmp` file to the bottom row of the pixel array. This is because when we convert the pixel array to an image, the first row corresponds to the bottom of the image.

Table 3: Example `bmp` *File Pixel Array Hexadecimal Values*

Row Number \ Byte Number	0	1	2	3	4	5	6	7
11	ff	ff	ff	ff	ff	ff	80	00
10	ff	ff	ff	ff	ff	ff	80	00
9	f7	ff	ff	ff	ff	fd	80	00
8	eb	ff	ff	ff	ff	fd	80	00
7	dd	76	38	e3	2e	3d	80	00
6	dd	75	d7	dd	55	dd	80	00
5	c1	54	18	dd	54	1d	80	00
4	dd	55	ff	5d	55	ff	80	00
3	dd	ae	30	e3	76	3d	80	00
2	ff	ff	ff	ff	ff	ff	80	00
1	ff	ff	ff	ff	ff	ff	80	00

We can now convert each hexadecimal value to its binary equivalent. In parsing the DIB header and color table, we found that each pixel is represented by a single bit, and a bit value of `0` represents the color black while a bit value of `1` represents the color white.

For the diagram below, leave each pixel with a bit value of `1` (white) empty and fill in each pixel with a bit value of `0`, and the final image will be revealed! You'll see that it's Awesome!



Document: Reading a Bitmap File

©2025 Chris Nielsen – *www.nielsenedu.com*

The diagram below has left a pixel bit value of `1` (white) empty and filled in a pixel bit value of `0` with a hash symbol (#). The final image is revealed! Isn't that Awesome!?

	0								1								2								3								4								5								6
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
1																																																	
0																																																	
9					#																																									#		
8				#		#																																								#		
7			#				#		#				#			#	#	#				#	#	#				#	#	#			#	#		#				#	#	#					#		
6			#				#		#				#		#				#		#						#				#		#		#		#		#				#				#		
5			#	#	#	#	#		#		#		#		#	#	#	#	#				#				#				#		#		#		#		#	#	#	#	#				#		
4			#				#		#		#		#		#										#		#				#		#		#		#		#									
3			#				#			#		#				#	#	#			#	#	#	#				#	#	#			#				#			#	#	#					#		
2																																																	
1																																																	