Subnetting and Binary Script

Overview

This script is going to cover the Subnetting aspect of Block 2. We are going to include a basic review of binary, a core aspect of subnetting as well as going through some general knowledge and even examples to help explain things in a better light.

Subnetting Video

Outline

~~**Introduction-**what subnetting is, why we subnet, what this video will cover (reference to a binary video as supplemental?)

~~**Basics of Subnetting-** IP address classes, Subnet masks, how to find and apply subnet masks

~~**More Basics**- Gateway, Broadcast Addresses

~~**Sample Problems**- based on aforementioned information

~~ **CIDR Notation**- mention that this makes the overall address classless

~~**Host Requirements**

~~**Sample Problems**- based on aforementioned information

~~**Subnet allocations**- provide a certain number of subnets, provide subnet IDs, and yuh

~~**Sample Problems**- something more similar to the last problem in the Packet

Introduction

Person 1: Today we are going to discuss Subnetting! This is a very broad topic, but we are going to explore all the details under that umbrella that you need to know.

Person 2: If you haven't already please refer to our quick Binary video, as that background will be very important when talking about the topics in this video.

Person 1: To start off, we will go over the importance of subnetting.

cool title transition occurs with pretty background

Person 1: Subnetting is the term we use when we divide a network into smaller, more manageable networks, called subnets. The primary reasons we subnet are to improve network performance by reducing unnecessary traffic (like broadcasts), reduce overall congestion, segment different groups into isolated domains, control the growth of the network, and general ease of administration.

Person 2: To get started, let's begin by discussing more about what subnetting is and why we would want to use it.

Basics of Subnetting

Person 2: IP Addresses are grouped into ***classes*** based on the first octet of the address. An octet being the first 8 bits of the IP address. There are Classes A, B, C, D, and E and they are defined as described in this chart.

Address Class	**1st Octet Range**	**Subnet Mask**	**CIDR Notation**
A	1 - 126	255.0.0.0	/8
B	128 - 191	255.255.0.0	/16
C	192 - 223	255.255.255.0	/24
D	224 - 239	N/A	N/A
E	240 - 255	N/A	N/A

As you may have noticed, Classes are defined not only by their first octet, but also by their Subnet mask. Subnet masks are what tell a system what addresses belong to the subnet. This will be explained in more detail later, but for now know that Class A subnets have the mask of 255.0.0.0, Class B have a mask of 255.255.0.0, and Class C has 255.255.255.0

Person 1: The commonly used classes are A, B, and C. These are the classes you will see in general use and they are the ones that we will cover today. D and E are reserved for multicast and research purposes respectively.

More Basics

Person 1: When you allocate IP addresses you should keep in mind that some addresses in a subnet are reserved and cannot be given to hosts. The first and last addresses in a given subnet are called the subnet and broadcast addresses respectively, and cannot be allocated to hosts.

Person 2: This must be taken into account when trying to allocate a certain number of hosts. Always remember your subnet must contain the required number of host addresses + 2! In addition to this you may need to calculate the host or network IDs. These are pretty simple to figure out using the given subnet mask.

Person 1: For example, we can use a method called ‘masking’ to find our network ID. To do so, we look at the binary representations of the host address and subnet mask, and keep all bits that are ones in both addresses. These bits make up the network ID, while the remaining bits of the host address are the host ID.

draw an example showing masking

Person 2: For an easier case, if the subnet mask is 255.255.0.0 this means that the first two octets are the Network ID and the last two and the Host ID.

Person 1: Regarding classful addresses, if you have a Class A address. There are 8 bits for the Network portion and 24 for the host. Class B has 16 and 16. Class C has 24 for network and 8 for host.

Sample Problems 1

Person 1: It's time for an example. So keeping in mind everything let's do this example:

IP Address	Class	Subnet Mask	Network ID	Host ID
169.16.2.3				

As we can see there's a decent chunk of information here that we want to figure out. Since we are given just the IP address the simplest thing we can do is determine the class.

Person 2: We can look at the first octet and see that the value of the first octet in decimal is 169. This lies within the 128-191 range for class B, so the IP address is Class B. Furthermore, since we now know it's class B, we know the subnet mask is 255.255.0.0!

IP Address	Class	Subnet Mask	Network ID	Host ID
169.16.2.3	B	255.255.0.0		

Person 1: Now that we have this information we can easily obtain the Network and Host ID using the subnet mask like we just talked about. So the Subnet Mask is 255.255.0.0 which means that there are all 1's in the 1st and 2nd octet which means that anything in those octets are within the Network ID. Following that logic, the Network ID is 169.16.0.0 and the Host ID is 0.0.2.3.

IP Address	Class	Subnet Mask	Network ID	Host ID
169.16.2.3	B	255.255.0.0	169.16.0.0	0.0.2.3

Person 2: Not too bad huh! We found all of the information and most of it was derived from the subnet mask.

CIDR Addressing

Person 1: I know that we have mentioned how each class of IP address has a specific subnet mask, but the subnet masks are not confined to just the ones we've shown you thus far. We call subnets with masks that do not fit within these classes "classless" addresses.

Person 2: Depending on the requirements of a given network network, you may want to use one of these classless addresses, so how do we do that? To help with that, we will use what is called CIDR (classless inter-domain routing) notation. This is typically given by /X where X can be any number from 0 to 32, and can be added to the end of an IP address to indicate the subnet mask.

Person 1: For example, if we were to have a /27 subnet, that would mean that in regards to our subnet mask, there are 27 bits that are "on" or set to 1, starting from the leftmost bit. Remember, each decimal number of a subnet mask or IP address can be broken down into an octet.

Person 2: If we have a /27 subnet, the mask can be represent in binary as:

11111111.11111111.11111111.11100000

As you can see the first three octets are all 1's the same as a Class C address, but because we have more than the 24 bits for class C, this IP address is no longer in a class, making it classless.

Person 1: You can see here that there are 3 bits being "borrowed" or "taken" from an octet. These are used for the subnet mask, and consequently shrink the host portion of the IP address, decreasing the total number of supported hosts on the subnet. Going off of binary we know that 8 1's is 255 so 255.255.255 is the first part of our subnet mask. From here we can add up the values that we "borrowed."

Person 2: Remember, each binary bit represents a power of 2. The Left most bit in an octet, if it is 1, represents 2^8, or 128. The next 64, and so on. So with three bits borrowed, the last decimal

number for our subnet mask is 128 + 64 + 32 which is 224. Making our final subnet mask 255.255.255.224.

Person 1: We can denote the CIDR notation like this: A normal IP address, representing the first IP in the subnet, with the /X where X is the full number of bits in the subnet mask. For example: 192.168.0.1/27. Or one can simply use /27 to refer to the subnet mask itself.

Host Requirements

Person 2: Now that we have mastered all these other concepts, we can begin building networks with specific requirements. There will be many hosts that can be allocated using our vast array of IP addressing. In general though we want to allocate the smallest number of hosts or subnets we can to fulfill the requirement.

Person 1: Since IP addresses use binary, there is not always a way to get the exact number of hosts or subnets required. You will most likely allocate MORE than is actually needed, but you want to allocate the closest number to the requirement. And don’t forget to account for the subnet and broadcast address!

Sample Problems II

Person 2: Now that we know how CIDR works and that we can allocate specific numbers of hosts, let's allocate for 300 hosts! The Network ID for this example will be 60.0.0.0

Person 1: First let's see what we need to find. We want to find the Class* of the address, the Bits Taken, Subnet Mask, and the CIDR.

Network ID	Host Request	Class	Bits Taken	Subnet Mask	CIDR
60.0.0.0	300				

As we can see we need to find the Class first. According to the Network ID, this is a class A address, as it falls in the range of 1-126. This helps us begin our journey in finding the other information.

Person 2: We now know that the subnet mask consists of *at least* 255 in the first octet. Now what we want to do is look at the number of hosts required. We need 300. We need to allocate space for these hosts in the host portion of the IP address, which will be the latter half. So we're gonna have to use some binary to accomplish this.

Person 1: We need to set aside the number of bits required to hold at least 300 hosts. If we include the subnet and broadcast addresses, that makes 302 total addresses. Since the host part of the address is the last portion, let's start from the right end of the address. The largest value 8 bits can hold is 255, so we know that we need at least 8 bits. We still need one more bit to hold 302. So let's grab the 9th bit from the left. This breaks into the 3rd octet but THAT'S OKAY. We now have 9 bits being used for the hosts which in turn gives us 512 options. That definitely holds 302.

Person 2: Now that we have figured out how long the host portion is, 9 bits from the right, we can find the bits taken. We know that there is 9 bits from the right, and 8 bits from the left, so in

order to give the smallest amount possible for hosts we need to “take” every in between for the subnet mask. This means the 15 bits between the 8 from the Class and the 9 from the host requirement are the “taken” bits. Here is our updated table.

Network ID	Host Request	Class	Bits Taken	Subnet Mask	CIDR
60.0.0.0	300	A	15		

Person 1: Now the rest is easy. To figure out the subnet mask we simply ascribe the value of 1 to every bit according to the class and bits borrowed. So we will have 11111111.11111111.1111111 0.00000000 To get the subnet Mask we simply need to add up these 1’s in binary. We know 8 1’s is 255 so that’s easy. 255.255.?.0. Adding up the 1’s in the third octet gives us the value of 254. Thus, the subnet mask is 255.255.254.0. Finally we need the CIDR, which is simply just the number of 1’s in the subnet mask. Which for this example is 23. So finally our completed problem looks like this.

Network ID	Host Request	Class	Bits Taken	Subnet Mask	CIDR
60.0.0.0	300	A	15	255.255.254.0	/23

Subnet Allocations

Person 2: Well what if we have a network that we want to divide into a number of subnets instead of hosts? The process is similar to allocating hosts but slightly different, we'll show you.

Person 1: When allocating for a specified number of subnets what you're doing is taking away host bits from left to right. So if you need 1,000 subnets from the class B network:

Network ID	Subnet Req.	Class	Bits Taken	Subnet Mask	CIDR
130.200.0.0	1000	B	???	255.255.??.??	/??

Person 2: We will then need to subtract or take away the necessary number of bits from the host portion to meet the subnet allocation. Therefore, for this example, you will need to allocate 10 bits from the host portion starting from the left on the bit before the last network bit, leaving it with a subnet mask and CIDR of:

Network ID	Subnet Req.	Class	Bits Taken	Subnet Mask	CIDR
130.200.0.0	1000	B	10	255.255.255.192	/26

Sample Problems III

Person 2: Let's look at our last example. Here we are given a Network ID and a subnet request. We see that the request is for 8 subnets, and the value of the first octet is 216. This means that it is a Class C address.

Network ID	Subnets Request	Class	Bits Taken	Subnet Mask	CIDR	1st Subnet ID	2nd Subnet ID	Last Subnet ID
216.10.12.0	8	C						

Person 1: In order to find the bits taken, you're gonna need to use some math. This is very similar to how binary works. The subnet request we have is for 8 subnets. We need to figure out what power of 2 fulfills that request. Conveniently for us, 2^3 is 8. So we know we have to borrow 3 bits, notice how we don't have to add 2 when dealing with subnets. If the request was 74 we would have to use 2^7 because 128 is the closest number that is greater than 74 and thus can hold 74.

Person 2: The next easiest thing is going to be figuring out the CIDR notation of the address. We know that we are taking 3 bits in addition to the 24 native to Class C addresses. That means our total for CIDR is 27. So it is a /27.

Network ID	Subnets Request	Class	Bits Taken	Subnet Mask	CIDR	1st Subnet ID	2nd Subnet ID	Last Subnet ID
216.10.12.0	8	C	3		/27			

Person 1: Moving on towards the subnet mask, we know that this Network ID is a Class C and thus the first three Octets of the subnet mask are 255. This only leaves us to figure out the last octet. We know we borrowed 3 bits. Remember that in Subnet Masks, the bits associated with the network are all 1's. Thus those first 3 bits in the final octet of the mask are all 1. A byte with only the first three bits valued at 1 comes out to 128 + 64 + 32 which is totaled to 224. This is the value of the final octet which gives us the full subnet mask as 255.255.255.224.

Network ID	Subnets Request	Class	Bits Taken	Subnet Mask	CIDR	1st Subnet ID	2nd Subnet ID	Last Subnet ID
216.10.12.0	8	C	3	255.255.255.224	/27			

Person 2: Now we are finally on to the last part, the Subnet IDs. As for the first subnet ID, this will always be 0.0.0.0. In order to find the second ID, we can invert the subnet mask and add one. This effectively shows us the amount that we would increment for each subnet. The last subnet ID is fairly simple, here we just take the "taken bits" of the subnet mask. In this case that gives us 0.0.0.11100000, which is 0.0.0.224.

Network ID	Subnets Request	Class	Bits Taken	Subnet Mask	CIDR	1st Subnet ID	2nd Subnet ID	Last Subnet ID
216.10.12.0	8	C	3	255.255.255.224	/27	0.0.0.0	0.0.0.32	0.0.0.224

Person 2: Alright that's all we got! Hopefully these examples and explanations helped a lot, and if you have any other questions please ask your instructors. They are there to help you. Alright, Thanks for watching!

— End of Script —

Binary-Decimal Conversions

Welcome. In this video we will be learning how to convert from binary to decimal, and back. To start with this, we have to ask what is a binary number? Well, a binary number is a string of 1's and 0's used to represent a value.

__0 off VALUE=0 __1 on VALUE=1

The 1 represents a value is active, or ON, and the 0 represents a placeholder, or an OFF value. An important thing to note about binary numbers is that the value increases from right to left for a given bit. The rightmost value represents one if the bit is turned on. The next bit can only be on or off, however if it is ON it is double the value of the previous bit, in this case 2.

_00 off VALUE=0 _10 on VALUE=2

If both of these are turned on, their values are then added together.

_11 off VALUE=3

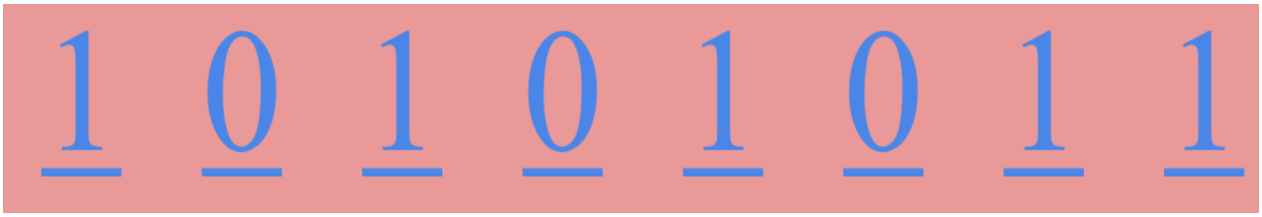
This pattern continues for as long as the string is.

CONVERTING FROM BINARY TO DECIMAL

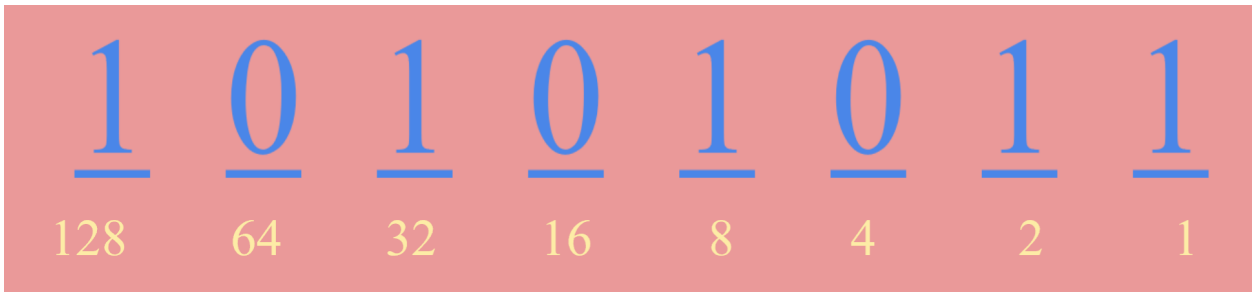
Let's look at an example to make sense of this in the context of IP addressing. IP address binary values come in groups of 8 bits. This can be daunting, so let's learn how to convert a longer string to decimal.

10101011 VALUE=???

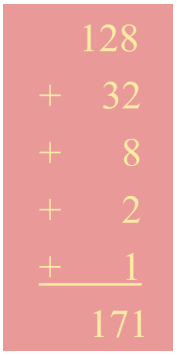
The first step to break this down would be to figure out how many bits are present. Counting them tells us there are 8 bits present, so let's draw that out on a piece of paper to make the coming calculations easier.



We know that the right-most bit has a value of 1 if it is turned on, so the next bit must have a value of 2. The bit next to that would be double 2, so 4. Let's notate what the value of each bit can be by adding that beneath it.



Finally, add together every value underneath an on bit. The number that is total is the decimal equivalent to this binary string.



By breaking the process down into steps like this, it becomes more manageable. Try it yourself with this string (answer will be provided at the end). 11010010. (210)

CONVERTING DECIMAL TO BINARY

Converting decimal to binary is very similar to the process of converting binary to decimal. The only difference is instead of adding numbers up to get a value, you subtract. Let's see how it works.

VALUE=167 Binary notation= ????????

To begin, again let's write out 8 bits and what values they can represent.

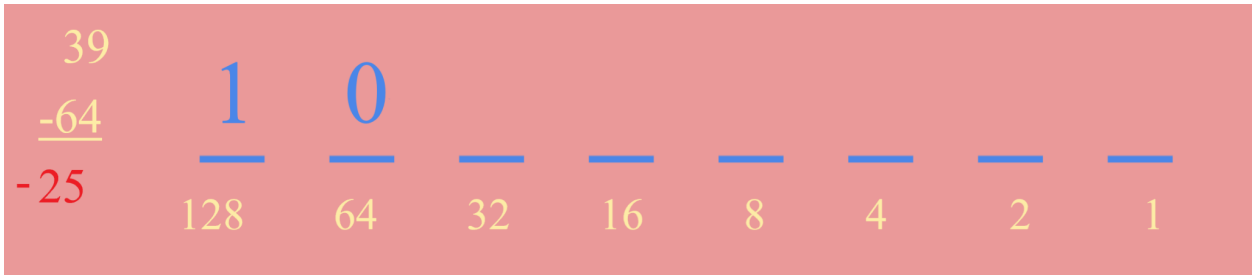
128	64	32	16	8	4	2	1

To convert decimal to binary, take the leftmost bit and subtract that from the given number. If that number can be taken and not leave the decimal negative, that bit will be on. In this case, 128 can be subtracted from 167 and leave 39, so that leftmost bit will be turned on.

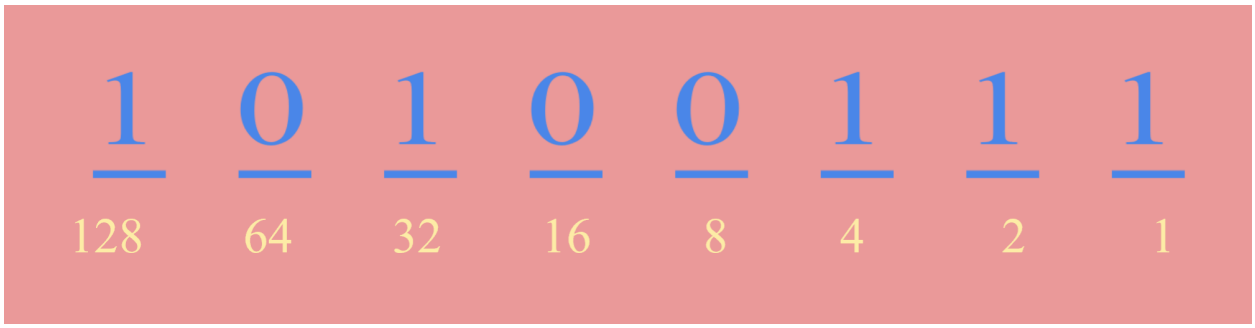
167
-128
39

1							
128	64	32	16	8	4	2	1

The next bit cannot be subtracted from the 39, so that will be an off bit.



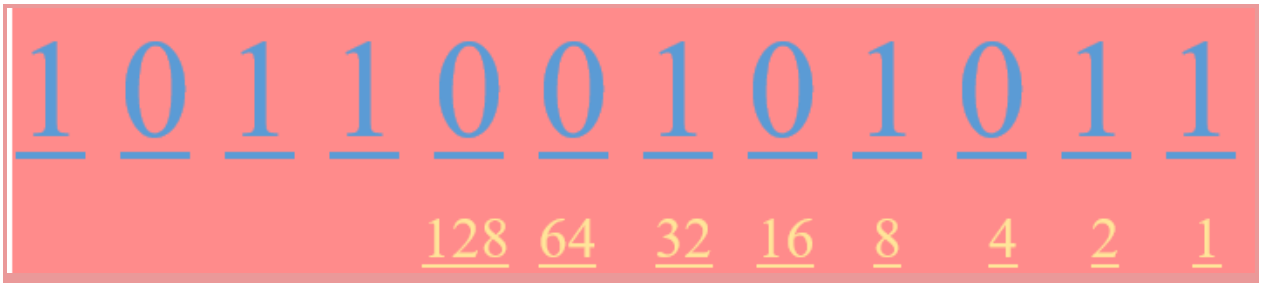
Continue this process to find the remaining values.



Try it yourself with the decimal of 224.

CONVERTING BINARY WITH MORE THAN 8 BITS TO DECIMAL ADDENDUM

Now that you have done some sample conversions with binary numbers of 8-bits, you can now tackle binary conversion with more than 8 bits. Take for instance the following binary number:



As you can see it is 12 bits long. This is bigger than the octets you will be dealing with with class-based subnetting, but will be common with variable-length subnetting (VLSM). Using what you have learned, we can see the 8 bits counting up from the right and their decimal equivalents. What we do not see is the decimal equivalents of the first 4 bits.

Using what you have learned (bits are double the value of the previous, right-most bit, starting from 1 when on), take a guess at what the decimal value of the next bit is going to be in the series.

If you guessed 256, then you are correct. The value in decimal of each bit is still double the value of the previous bit. Completing the series:

1	0	1	1	0	0	1	0	1	0	1	1
2048	1024	512	256	128	64	32	16	8	4	2	1

And with the chart complete, it is simple matter of adding the values of the places that are on:

```
2048
 512
 256
  32
   8
   2
+  1
----
2859
```

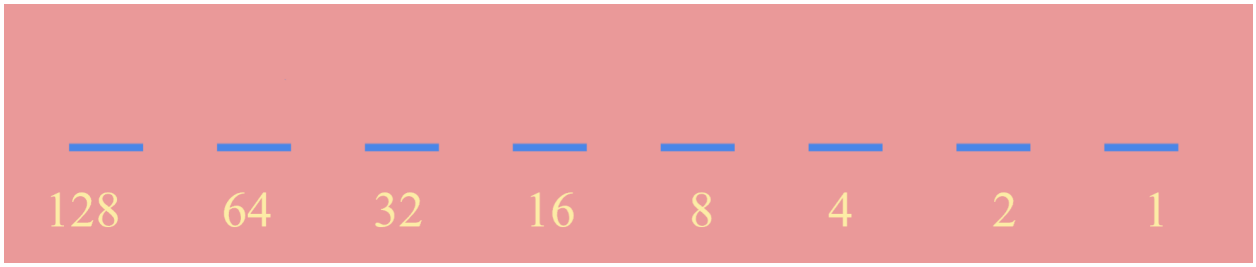
101100101011 = 2859

CONVERTING DECIMAL TO BINARY WITH MORE THAN 8 BITS ADDENDUM

Like converting binary to decimal with more than 8 bits, decimal to binary conversion with more than 8 bits is similar. Take for instance the number:

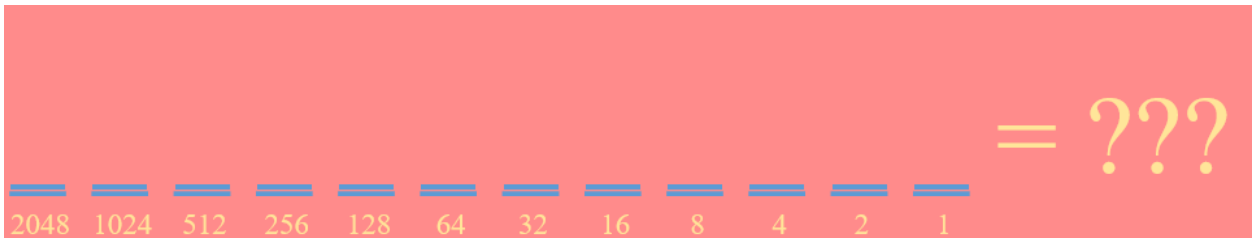
691 = ??? in binary

Just like before with our previous examples of decimal to binary conversion, we can start by writing out:

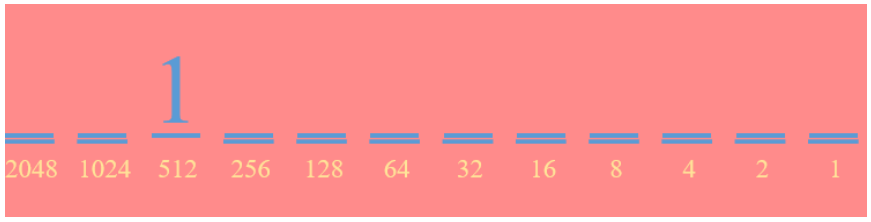


But you should find out that 691 is to big to convert into 8 bit binary because:

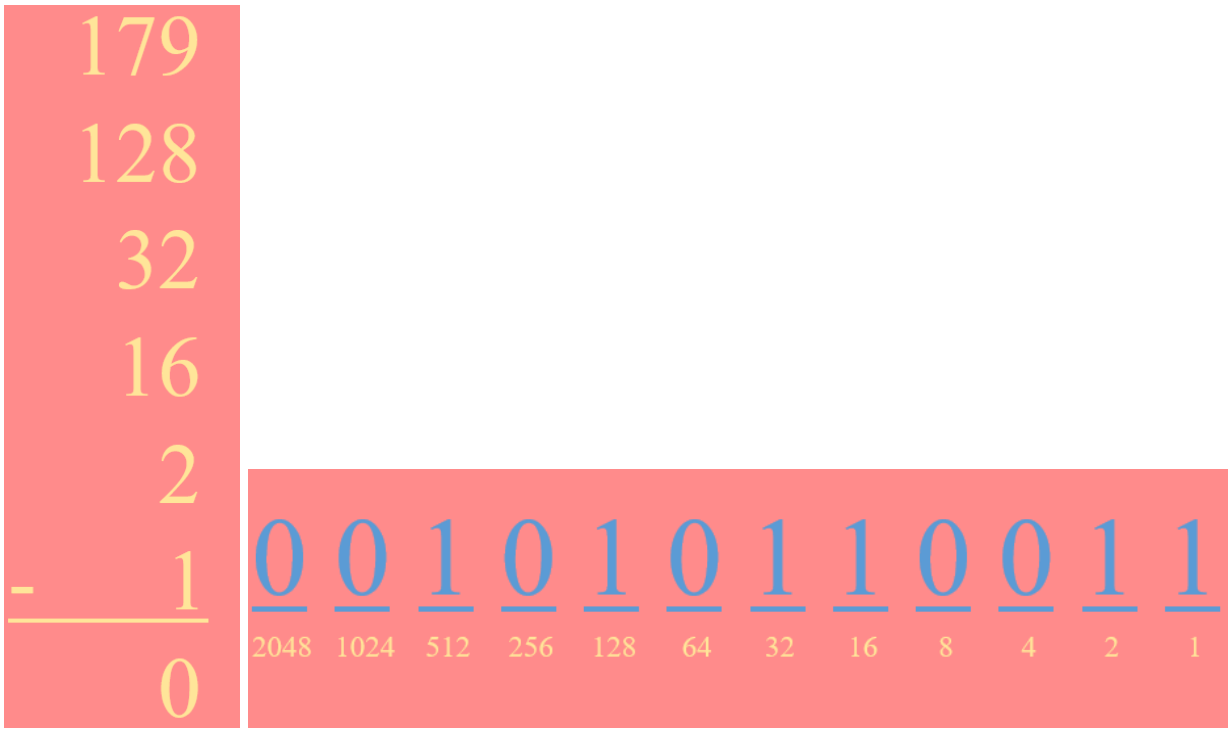
Therefore, you are going to include more bits, say 12 bits, to determine this decimal number's (691) equivalent in binary.



Now, using the chart, we can see the bit valued 512 the biggest number that can "fit" into 691.



Now, we can subtract for the remaining on bits:



And remove the two leading zeros because they are placeholders, not values to count towards the binary number.