Learn2Subnet

Section One

Subnetting

by PrepLogic®

First the Basics
Binary Arithmetic

If you understand how binary numbers work, you can skip this section and go to the next. But, if you don't, you need to spend a bit of time here to get a basic understanding of binary numbering.

All numbering systems work the same way. The one we are most familiar with, Base 10 (decimal), works the same way as Base 2 (Binary) or Base 16 (Hexadecimal) for that matter. Let's take a random number, such as 1,234. We know immediately what that number is. It is so obvious to us that it seems trite to say that the number is the sum of one thousand plus two hundred plus thirty plus four. However, we can express this sum in another, more interesting way:

(1 * 10^3) + (2 * 10^2) + (3 * 10^1) + (4 * 10^0) = 1,234 (The "^" symbol indicates "*raised to the power of*", eg "10^3" means 10 raised to the power of 3, or 10 * 10 * 10)

It should be pretty obvious why we can refer to this number system as Base 10. We have 10 digits to work with (0 – 9).

Binary (Base 2) numbering is like decimal (Base 10) numbering except that we have only 2 digits to work with: 0 and 1. Let's take a typical binary number, such as 1101. Like Base 10 numbers, we can express a binary number as a sum of other numbers. In the case of the example (1101), we can express the number thus:

(1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2 ^ 0) = 8 + 4 + 0 + 1 = 13. (By the way, any number raised to the power of "0" is "1".)

As with decimal numbers, we can easily internalize a lot of the work we have to do by looking at the positions of the digits in a particular number.

Take a longer binary number, such as the octet, `11111101`. If you are familiar with counting in binary you will know immediately that this number is 253. If you are not familiar with counting in binary, don't despair: it is not that hard. Consider the table below.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	1	1	1	1	0	1

```
128+ 64 + 32 +  16 +  8 +  4 +   0 +  1  = 253
```

But there is an even easier way to convert this number to decimal. We know that the binary number 11111111 is 255 in decimal (you just have to memorize this). When we look at a number like 11111101, all we have to do is subtract 2 (in this case) from 255 to arrive at our decimal conversion, 253.

Although a TCP/IP address is 32 digits long, the address is broken up into 4 equal groups of 8 digits (4 groups of single bytes) to make it easier to work with. So, here is a tip for working with binary numbers in TCP/IP addresses: draw out the first 8 positions as in the table above. If you use that table, you will be able to convert any decimal TCP/IP address to binary and vice versa. One final tip: practice converting binary to decimal and vice versa; check your results in the Windows calculator in scientific mode.

Calculating Subnet Masks

What are subnet masks and why do we need them? To answer these two questions, we have to spend a little time discussing some basics about TCP/IP. A TCP/IP address, such as `172.16.8.1`, is composed of at least two parts: a section that denotes the network portion of the address and a section that denotes the host portion of the address. As an analogy, you can think of a postal address: one part of the address refers to a street; another part refers to a particular house. In order for the address to work properly, both parts have to be unique. How does your computer "*know*" which portion or the TCP/IP address represents the network and what portion represents the hosts? It looks at the subnet mask for the address. The subnet mask distinguishes the network from the host portion of the address (e.g., `172.16.8.1/16`). Any bits in the address that are "*masked*" by the subnet mask will represent the network portion (`172.16`.) of the address; the remaining bits represent the host portion (`.8.1`) of the address.

When people look at a TCP/IP address, they often think about the class of address it represents. In our example above, the IP address `172.16.8.1` is a Class B address. For a Class B address, the default subnet mask is `255.255.0.0`. This simply means that the first two octets, `172.16`, represent the network portion of the address, and the last two octets represent the host portion of the address. To determine if an address is a Class A, Class B, or Class C address and the default subnet masks for these addresses, you can consult the following table:

First Octet (dec. & binary)		Class	Subnet Mask
1 – 126	(0xxxxxxx)	A	255.0.0.0
128 – 191	(10xxxxxx)	B	255.255.0.0
192 – 223	(110xxxxx)	C	255.255.255.0

For a Class A address, the first octet represents the network; for a Class B the first two octets represent the network; and for a Class C the first 3 octets represent the network. If you keep in mind our postal address analogy, it should be clear that there a few "*Class A*" streets, but a huge number of houses on each street. There are more "*Class B*" streets and fewer houses on each street, and so on.

Let's go back to our sample TCP/IP address, `172.16.8.1/16`. The default network portion of the address is represented by the first two octets, which can be expressed as `172.16.0.0/16`. The "/16" represents the number of bits used for the subnet mask—in this case, it is 16 bits (`11111111.11111111.00000000.00000000 = 255.255.0.0`). In fact, it is now standard practice to refer to all IP addresses this way. We are left with 16 bits to represent the host portion of the address. With 2^16 or 65,536 (give or take one or two) possible address to represent the hosts on the `172.16.0.0/16` network, we obviously have lots of room for growth.

But, hold on, that is way too many computers to place on a single cable. Furthermore, if we put the maximum number of hosts that we could physically accommodate on an Ethernet cable, we would waste a lot of addresses. That's where a custom subnet mask comes into play. We can borrow bits from the host portion of the address and use them to represent the network portion of the address. In effect, what we can do is subdivide our "*street*" into a number of smaller "*streets*", or subdivide our network into smaller subnets.

Let's say we decide to use the entire 3rd octet of the address to extend the number of subnets on our network. I would use a subnet mask of `255.255.255.0`. That would give us approximately 254 possible networks, each with approximately 254 hosts. (I say approximately because the actual and precise number of hosts and networks depends on your hardware and software, but this is a technicality best left to people who support routers or are studying for their Cisco certifications.)

Given this subnet, a host with an IP address of `172.16.8.1/24` is on a different network from a host with an IP address of `172.16.9.1/24`. This means that both hosts need to be separated by and reachable through a router. If the subnet mask were `255.255.0.0`, both hosts would be on the same network.

AND IP Addresses

Okay, we've seen the need to extend the subnet mask, but how does TCP/IP "*know*" whether hosts are on the same or different networks? Whenever a computer is instructed to communicate with another TCP/IP host, it "*ANDs*" its address and the destination address with the subnet mask and compares the result. If the result is the same, the TCP/IP stack will do an ARP (Address Resolution Protocol) broadcast to determine the MAC (Media Access Control) address of the network adaptor of the destination host. Once it has the MAC address of the adaptor, the computer will start communicating with the host. If, however, the result of the "*ANDing*" is different, the source host will do an ARP for the MAC address of router on the network, which is usually the configured default gateway. (Of course, your computer maintains an ARP cache in memory, and it will not do an ARP if it first finds the information in the ARP cache).

What is "*ANDing*"? "*ANDing*" is similar to multiplication, except it is used for logical operations. There are 4 logical operations we can perform with binary numbers: AND, OR, XOR (exclusive-or) and NOT. Here is why "*ANDing*" is like multiplication. A "0" ANDed with a "1" results in a "0". A "1" ANDed with a "1" results in a "1".

Let's go back to our sample extended network. A computer with an IP address of `172.16.8.1/24` is trying to establish an HTTP session with a computer that has an IP address of `172.16.9.1/24`. The question we need to answer is: Are these computers on the same or a different network?

Let's look at the subnet mask of the source computer. It is using 24 bits as the subnet mask, which can be expressed as `255.255.255.0` or as `11111111.1111111.11111111.0`. To determine if the two computers are on the same or a different network, your TCP/IP stack will AND `172.16.8.1` and `172.16.9.1` with `255.255.255.0`. We need only consider the third octet. We don't have to consider the first two octets because they are identical.

```
8
255
00001000 AND 11111111 = 00001000
```

```
9
255
00001001 AND 11111111 = 00001001
```

Because the result of the AND is different for the source and host address, the source computer will ARP for the router interface. Once it has the MAC address for the router, it will start communicating with it, and the router will send the packets on to another router or the final destination.

What if our subnet mask were something different, like `255.255.252.0`? Given this subnet mask, are these two machines on the same or a different network? If we AND the addresses, here is what we get:

```
8
252
00001000AND 11111100 = 00001000

9
252
00001001 AND 11111100 = 00001000
```

Because the result of the AND is the same for both addresses, your TCP/IP stack will assume that the two hosts are on the same network and do an ARP for the MAC address of the final destination.

As it turns out, ANDing is something a computer is very good at (it needs ANDing to route packets properly, for example) Fortunately, though, we don't have to AND addresses with subnet masks to determine if hosts are local (on the same network) or remote (on different networks). There are easier ways to do this, so easy in fact that you should not have to use a calculator or software to calculate subnet masks. Calculating subnet masks needn't require anything more than your brain and pencil and paper. In section two we will take a look at a simpler method for calculating subnet masks.

Learn to Subnet Section Two

Let's take a look at an example of two hosts on a Class B network trying to communicate with one another. Let's assume the IP address of the source host is `172.16.32.1/16` and the destination host address is `172.16.64.1/16`. Because we are using the default subnet mask of `/16` or `255.255.0.0`, `172.16.32.1/16` and `172.16.64.1.16` are on the same network. As long as there is no change in the network portion of the address, the two hosts are on the same network. The network portion of the address is determined by the subnet mask of `255.255.0.0`, which tells the TCP/IP stack that the first two octets, `172.16.0.0`, represent the network portion of the address. However, what if we need more than one network, perhaps because we have remote locations or we have more hosts than we can place on a single cable? It makes sense to sub-divide the network into smaller networks. What if we needed to create 6 subnets from our larger, single network? In a case like this, we would need to extend the default subnet mask by borrowing some of the bits from the host portion of the IP address. To create at least six subnets from the `172.16.0.0/16` network, we need to borrow at least 3 bits. So, our subnet mask would be `255.255.224.0`. In binary, the subnet mask would look like this:

```
11111111.11111111.11100000.00000000
```

As we learned in section one, the TCP/IP stack will AND the source and the destination IP address with the subnet mask and compare the results. If the results of the ANDing are the same, the two hosts are on the same network. If the results of the ANDing are different, the two hosts are on different networks.

Let's do the ANDing for `172.16.32.1/19` and `172.16.64.1/19`. We can ignore the first two octets, since they are identical for both addresses. The octet of interest is the 3rd octet.

```
32                224        32
00100000   AND   11100000 = 00100000

64                224         64
01000000   AND   11100000 = 01000000
```

Given a subnet mask of `255.255.224.0`, the two hosts are on different networks. One way you can think about subnet masks is this: any time there is a change in the bits used to represent the network portion of the address, you have a separate network. In the case of a custom subnet mask of 224, we create at least 6 networks. (I say at least because the actual number depends on the hardware or the software. This is a rather technical issue that you needn't concern yourself with right now. For the time being, we are going to assume that neither the network nor the host portion of the address can be all 0's or all 1's, as per the original 1985 standard for subnetting, RFC 950.)

Here are the possible network IDs in the 3rd octet for our `255.255.224` subnet mask:

```
00000000    0    (Normally not allowed according to RFC950)
00100000    32    172.16.32.0/19
01000000    64    172.16.64.0/19
01100000    96    172.16.96.0/19
10000000    128   172.16.128.0/19
10100000    160   172.16.160.0/19
11000000    192   172.16.192.0/19
11100000    224  (Normally not allowed according to RFC950)
```

Remember, we are dealing only with the 1st 3 left-most bit positions. A subnet mask of 224 "*masks*" off these bits to represent the network. For any two IP addresses, if the values in these bit positions change, the IP addresses are on different networks. So, for example `172.16.64.1/19` is on a different network from `172.16.100.1/19`. However `172.16.32.1/19` is on the same network as `132.16.63.254/19`. Just write out the 3rd octet in binary for both numbers. Do the bits representing the network portion of the change? If the answer is "*yes*", then the addresses are on different networks. If not, the hosts are on the same network.

Let's go back to our example subnet of `255.255.224.0`. Consider the table representing the possible network IDs above. Notice any patterns? One pattern that stands out is the difference between one Network ID and another: the difference is 32. That is, starting with the first network ID of `172.16.32.0/19`, we increment by 32 in the 3rd octet to arrive at the next network ID of `172.16.64.0/19` and so on up to `172.16.192.0/19`.

Here's another relationship to consider. Have a look at the subnet mask expressed in binary.

```
128 64 32 16 8 4 2 1
1   1  1  0  0 0 0 0 = 128 + 64 + 32 = 224
```

The value of the lowest order bit for the subnet mask is 32. This is not a coincidence, if you consider the permutations in the table above.

So, here is a general rule for calculating subnet masks. Use the low order bit for the subnet mask to determine the starting network ID for your subnets and the value used to increment from one network ID to the next. In our example, the value is 32.

Here is another general rule for calculating subnet masks. The number of bits used to represent the custom subnet can be used to calculate the number of possible networks. The formula is this:2^(no. of bits used for subnet) – 2 = number of possible subnets. So for our example, we used 3 bits to give us (2^3)-2 = 6 networks. We subtract 2 because the all "0" and the all "1" subnet may or may not be allowed, depending on your hardware or software. (If your hardware or software supports the all "0" network, great. Instead of writing me, you can adjust the formula to suit your circumstances.)

These two rules may be applied to any subnet mask. Consider the two IP addresses `172.16.32.1/20` and `172.16.40.1/20`. Are these hosts on the same or different network? To determine this, draw out the bits for the subnet mask.

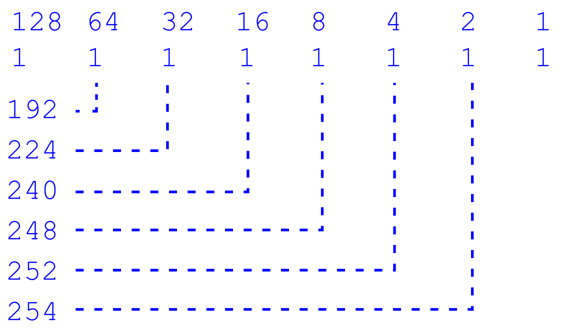
```
255      255      240      0
11111111.11111111.11110000.00000000
```

The value of the low order bit for the subnet mask is 16. 128 64 32 16 8 4 2 1 1 1 1 1 0 0 0 0 = 128 + 64 + 32 + 16 = 240. Here are the valid network IDs for the 172.16.0.0/20 subnets (excluding the all "0" and all "1" subnets)

```
172.16.16.0/16
172.16.32.0/16
172.16.48.0/16
...172.16.224.0/16
```

Any two hosts between `172.16.32.1/20` and `172.16.47.254/20` are on the same network. So, our two hosts (`172.16.32.1/20` and `172.16.40.1/20`) are on the same Any two hosts between `172.16.32.1/20` and `172.16.47.254/20` are on the same network. So, our two hosts (`172.16.32.1/20` and `172.16.40.1/20`) are on the same network. However, what if our subnet mask is `/21` (or `255.255.248.0`)? Our two hosts are on different networks. How can we tell this easily? Look at the low order value for the subnet mask. For a subnet mask of `248`, the value is "8". So, starting at "8" the valid network IDs are `172.16.8.0/21`, `172.16.16.0/21`, `172.16.24.0/21` and so on up to `172.16.240.0/21`. How many subnets does our extended mask give us? We borrowed 5 bits from the host portion of the address; therefore, the possible number of subnets we create is 2^5 – 2 = 30.

Anytime you are in a situation, such as a Microsoft exam, where you might have to calculate subnets, draw out this table.



Here is how you use the table. Let's say in an exam situation, you need to determine if two particular hosts are on the same or different networks. Let's take two IP addresses at random, such as `172.16.36.1/20` and `172.16.43.254/20`. The subnet mask is 20 bits long: `255.255.240.0` (`11111111.11111111.11110000.00000000`).Take a look at the low order bit used for the subnet mask in the 3rd octet. A value of 240 means that the low order bit is 16 (just follow the lines on the chart). If the low order bit is 16, that means the first network ID is `172.16.16.0/20`, the next is `172.16.32.0/20`, followed by `172.16.48.0/20`, and so on. In other words the network ID starts at x.y.16.0 and increments by 16 for each network. The two hosts are on the same network.

Let's take this a little farther. Given a Class B address of `172.16.0.0/16`, how many subnets are created by extending the subnet mask by 4 bits into the 3rd octet to `255.255.240.0`? The answer is 2^4 – 2 = 14. (This number assumes that our hardware and software do not support the all "0" or the all "1" subnets.)

How many hosts can we have on each subnet? With a Class B address and a subnet mask of `/20`, we have 12 bits left over to represent the host addresses on each network. Therefore, the number of addresses on each network is 2^12 – 2 = 4094. Like our network addresses, we are going to be consistent and say that the host portion of the address can't be all 0's—that represents the network ID. Nor can the host portion of the address be all 1's—that number represents the broadcast address on the subnet.

What is a valid range of host address on our `172.16.32.0/20` network? Keeping in mind that the host portion of the address can't be all 1's or all 0's, the valid range of host addresses is `172.16.32.1` – `172.16.47.254`. Why do we end at x.y.47.254? Because the next network ID is x.y.48.0, and we can't have all 1's in the host address, which rules out x.y.47.255.

Let's look at one last example before we move on. Assume we have been given a Class C address of `192.168.100.0/24` and need to create at least two subnets from this address. How many bits borrowed from the last octet will give us two subnets? 2 bits (2^2-2=2). Consulting the table above, we see that borrowing two bits gives us a subnet mask of `255.255.255.192`.

Assuming that we can't use the all 0 or the all 1 subnet, what is our first network ID? Consulting the table above, we see that the low order bit for the 192 subnet is 64. Therefore, the first network ID is `192.168.100.64/26`. What is the next network ID for this subnet mask? Look at the value of the low order bit for the subnet mask and add that to the first network ID to determine the next network ID. The next network ID is `192.168.100.128/26`.

Now for the tricky part. What are the valid ranges of host addresses for the subnets of `192.168.100.0/26`? The valid host ranges are `192.168.100.65/26` – `192.168.100.126/26` and `192.168.100.129` – `192.168.100.190/26`. If the numbers in the last octet look a little strange, write them out in binary. Keep in mind that for host addresses we can't have all 0s or all 1s in the host portion of the address. So, for example, `01000000` (or 64) is an invalid host address because the last 6 positions are 0's. Likewise `01111111`(127) is an invalid address because the last 6 positions (the host portion of the address) are all 1's.

If you practice calculating subnet masks, you will find after a while it becomes fairly easy, especially if you use the table. You will also find the table pretty handy for exams where you have to know subnetting. When you get into the exam booth, write out the binary table with the subnets. If you want, copy out these simple rules. The lowest order bit in the subnet mask determines the starting network ID and the value that you can add to the network ID to determine the next network ID. To calculate the number of hosts orthe number of networks, use this formula: 2^(no. of bits) – 2 = number of hosts/networks. You shouldn't have all 0s or all 1s in the network or host portion of the address. Keep in mind that some hardware/software may create exceptions to this last rule.

That's it. I think it's pretty easy, and it sure beats memorizing tables or performing complicated arithmetic. As a final test of your ability to subnet, here is one final teaser. What are the first 10 possible subnets for this subnetted network: `172.16.0.0/25`? Hint, the first network ID is `172.16.0.128/25` and there are 2^9 – 2 possible subnets. What are the ranges of valid hostaddresses for the first few subnets?

CIDR Block Rules or Supernetting Made Easy

You thought we were finished, didn't you? If your brain hurts from the previous section, put this aside and come back to it again. But, do come back. Knowing about CIDR blocks and supernetting, not to mention Variable Length Subnet Masks (VLSM), may be important to you someday. Furthermore, none of these topics is much more difficult than subnetting, if you spend some time with them.

If you know how to subnet, you know how to supernet, even though you might not think you can. A supernetted address is one in which bits are borrowed from the network portion of the address, rather than the host portion of the address. But, aside from this difference, the arithmetic works the same for supernetted addresses as for subnetted addresses.

Where do we use supernetted addresses? Normally, we would not use supernetted addresses on client computers. Supernetted addresses are used on routers to aggregate multiple entries in their routing tables into single entries. Here is how it works. Let's say we have a large block of class C networks that is reachable through a single router, Router A. Router A will advertise to its neighbor routers that it has routes to a bunch of class C networks. Let's say Router A advertises routes for each network in the range `192.168.8.0/24` – `192.168.15.0/24`. In other words, Router A has 8 individual entries in its routing table.

This is precisely the situation where CIDR blocks can be used. CIDR stands for Classless InterDomain Routing. Think about that term for a bit. In the previous section on subnetting, we dealt with classfull addresses, that is addresses that could be characterized as Class A, B, or C. With those addresses we cared (to some extent) what the default subnet mask for a particular address is. For Classless Interdomain Routing, we really don't care about the default subnet mask. All we are interested in is determining the mask that will allow us to describe a group of contiguous networks as a single network. In other words we are going to use the same techniques for supernetting as we used for subnetting, except we are going to ignore the class of the addresses we are dealing with and their default subnet mask.

Let's go back to our example router that contains entries for 8 routes to networks between `192.168.8.0/24` – `192.168.15.0/24`. We want to replace these 8 entries with a single entry. From the table we used for subnetting, we know that if we were dealing with a Class B address (instead of a Class C), we could describe this range of addresses as a single network with the subnet mask `255.255.248.0`. Look at the difference between 15 and 8 to determine the low order bit for the mask. Since 8 is the closest value, we know that we need 5 bits in the 3rd octet for the mask (`11111000` = 248). It doesn't matter that we are dealing with Class C addresses since this is classless addressing. With CIDR blocks I can now advertise a route to these 8 networks as a single entry: `192.168.8.0/21`.

Not all routers can support CIDR blocks. For a router to support CIDR blocks, it must also support a routing protocol such as RIP v 2 or OSPF.Both of these protocols provide information about the subnet mask when they communicate with other routers. That not only makes CIDR blocks possible, but it also makes the all "0" networks possible, along with Variable Length Subnet Masks.