In this lab, we explore a process for allocating subnets when using variable length subnet masks (VLSMs). The process is designed to help reduce the number of unused addresses on the network.

LAB OBJECTIVES

The objectives of this lab are:

- to learn how to efficiently allocate subnets using variable length subnet masks on a network with no subnets.
- to learn how to efficiently allocate subnets using variable length subnet masks on a network with existing subnets.

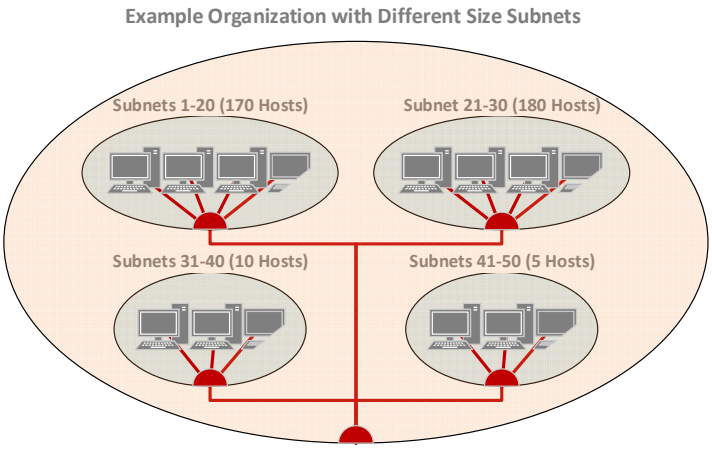
LAB SUBMISSION

Use the submission template provided in the assignment inbox to perform the steps requested by this lab. Return to the assignment inbox to submit your lab.

Section One – Variable Length Subnet Masks

OVERVIEW

To review the issues that can arise with use of a fixed length subnet mask, let us take a look at an example of an organization's network in the following figure.



This example organization uses 50 subnets which are of varying sizes. Now imagine that this example organization is assigned an 18-bit network identifier. Using a fixed length subnet mask, the organization would minimally need to use 6 bits for subnet identifier, which would allow for 64 subnets. However, in doing so, all subnets then allow for 254 host addresses. This allowance is reasonable for the subnets that contain 170 or 180 hosts, and in fact is the smallest size that would allow for that number of hosts, but what about the subnets that contain only 5 or 10 hosts? Hundreds of addresses are wasted on each of those subnets!

Wasted addresses are not the only concern. If the organization's network substantially grows, the addresses wasted on the smaller subnets could cause the organization to run out of room, that is, to no longer be able to assign addresses to the new hosts. And, while beyond the scope of this lab, the presence of subnets that have many wasted addresses actually reduces routing performance, because the routers must accommodate for the addresses that will never be used.

Recall that the concept of VLSMs is subdividing subnets into additional subnets, and that the use of VLSMs can help significantly reduce the number of unused addresses on a network.

The process of allocating subnets when using VLSMs differs from the process of allocating subnet when using fixed length subnet masks. When using fixed length subnet masks, we identify the subnet(s) that requires the most hosts, choose the length of subnet identifier that supports that number of hosts, and then apply that length to all

subnet identifiers regardless of their size. The largest subnet is the lowest common denominator that determines the number of available hosts for all subnets, regardless the number of hosts each subnet need support. When using VLSMs, we in effect apply the fixed length subnet mask process iteratively, so that each different size subnet uses only the length subnet identifier that it needs. A description of the process to apply to create sub-subnets is given below.

Process for Creating VLSMs on a Network

Beginning with the subnet(s) that requires the most number of hosts and ending with the subnet(s) that require the least number of hosts, repeatedly choose the next available subnet identifier that is as long as possible, but still supports the needed number of hosts for each subnet. With each choice, do not re-use any subnet identifier that has already been used.

Now this process may not be as clear as it can be to you at the moment, so let us explore the process by continuing with the aforementioned example organization that needs 50 subnets of varying sizes.

Assume that the 18 bit network address assigned to the organization is 76.34.192.0/18, which means the network identifier is 01001100 00100010 11 in binary. We know that organization needs 6 bits for the subnet identifier to support the 50 subnets because 6 bits supports 64 subnets, but 5 bits would only support 32 subnets. We also know that the first 30 subnets have 170 or 180 hosts, and so those 30 subnets will need the remainder of the bits, 8 bits, for the host identifier. This is because 8 bits accommodates 254 hosts, but 7 bits only accommodates 126 hosts.

So for the first 30 subnets, we would begin assigning them network identifiers as follows:

First subnet: 01001100 00100010 11000000
Second subnet: 01001100 00100010 11000001
Third subnet: 01001100 00100010 11000010
Fourth subnet: 01001100 00100010 11000011
…
Thirtieth subnet: 01001100 00100010 11011101

Up until now, we have followed the same methodology we learned previously, which is that of using a fixed length subnet mask. The next subnet is where we do something different. Subnet 31 only has 10 hosts, and so if we simply create its network identifier as:

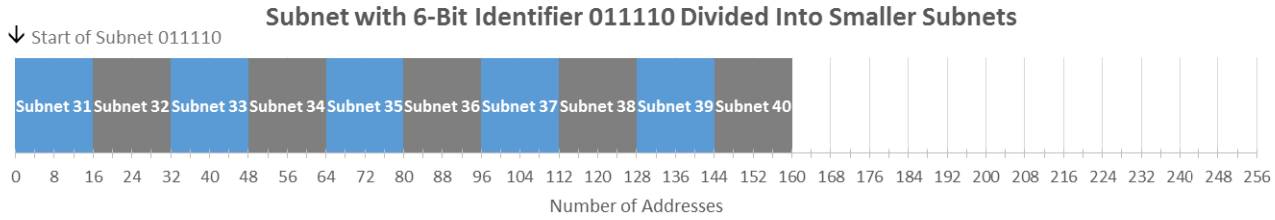
01001100 00100010 11011110

then 244 of those addresses would be unused on that subnet. What is the solution? Treat the would-be 31st subnet as another network and create additional subnets from that subnetwork. We need 4 bits for the host identifier, because 4 bits supports 14

hosts. So we start with what would have been the 31st network identifier if we were using only 6 bits for the subnet identifier, then start creating subnets from that, like so:

```
Thirty-first Subnet:  01001100 00100010 11011110 0000
Thirty-second Subnet: 01001100 00100010 11011110 0001
Thirty-third Subnet:  01001100 00100010 11011110 0010
…
Fortieth Subnet:      01001100 00100010 11011110 1001
```

So subnets 31 through 40 use 10 bits for their subnet identifier instead of 6, thereby subdividing the 256 addresses available for would-be 31st subnet (with 6-bit identifier 011110) into blocks of 16. This is illustrated in the following figure.



In the preceding figure, the subnet's available addresses appear consecutively, and it is clear which addresses have been allocated to which sub-subnets. Notice that subnet 31 uses the first 16 addresses from the would-be larger 31st subnet, subnet 32 uses the next 16 addresses, and so forth, effectively subdividing the larger subnet into smaller subnets.

In this particular example, subnet 011110 contains an 8-bit host identifier, which allows for 256 possible addresses. Of course, not all subnets contain an 8-bit host identifier, and therefore other subnets may support a different number of addresses.

Let us summarize the steps we went through in this example thus far. The first 30 subnets need all 8 bits for the host identifier, but subnets 31 through 40 only need 4 bits. Rather than using the full 8 bits and wasting 244 addresses per subnet, we create the would-be 31st subnet, treat it as a network, then subdivide that network 10 times to create subnets 31 through 40. In doing so, subnets 31 through 40 use 10 bits for the subnet identifier instead of 6. Only 4 addresses per subnet are unused, which is not a problem, because the extra 4 gives room for growth on each subnet.

And what about subnets 41 through 50 which only have 5 hosts? Technically we only need 3 bits in the host identifier to support these subnets, because 3 bits allows for 6 hosts. So now we continue with this same methodology to create additional subnets once again. We initially start with the next available subnet with a 10-bit subnet identifier, treat it as a network, and create more subnets.

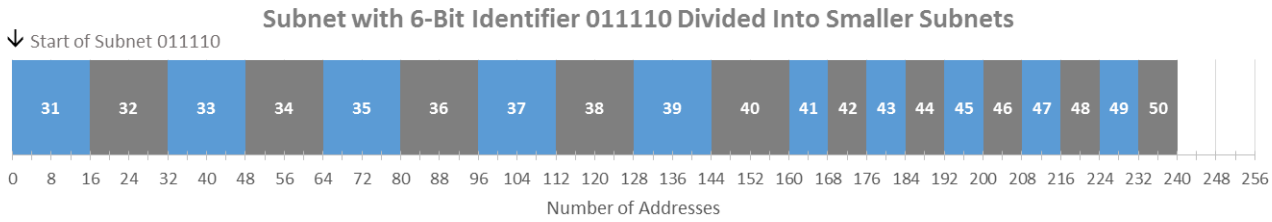
```
Forty-First Subnet:  01001100 00100010 11011110 10100
Forty-Second Subnet: 01001100 00100010 11011110 10101
```

We run into the issue that the one additional bit allows for two subnets but not more, so we must increment the 10-bit subnet identifier by one, then repeat the process.

```
Forty-Third Subnet:  01001100 00100010 11011110 10110
Forty-Fourth Subnet: 01001100 00100010 11011110 10111
…
Fiftieth Subnet:     01001100 00100010 11011110 11101
```

In this case, we could not limit ourselves to subdividing a single subnet with the 10-bit identifier, but we needed to subdivide 5 of those subnets to create our smaller subnets as needed. This is fine, and is a part of the process. The key is that we cannot subdivide larger subnets that have already been used in full, but we are free to subdivide any larger subnets that have not yet been used to create our smaller subnets.

If we now take a look at what happened with the original, would-be 31st subnet with the 6-bit subnet identifier, it has been further subdivided as illustrated in the following figure.



Notice that subnets 41 through 50 continue subdividing the larger subnet, and that each of these smaller subnets uses 8 addresses.

Now this may all still seem confusing, so try reading through the following summary of how we applied the process to this example, to gain a better understanding.

Step	***Example***
i) Beginning with the subnet(s) that requires the most number of hosts and ending with the subnet(s) that require the least number of hosts, repeatedly choose the next available subnet identifier that is as long as possible, but still supports the needed number of hosts for each subnet. With each choice, do not re-use any subnet identifier that has already been used.	*In the example above, we found that subnets 1 through 30 had the highest number of hosts, which was 170 and 180 hosts. Using 6 bits for the subnet identifier was the largest identifier we could use and still accommodate 170 or 180 hosts. We used subnet identifiers 000000 through 011101 for these first 30 subnets. We would have done the same if we had used a fixed length subnet mask here, so we have not yet diverged from that process.*
ii) Beginning with the subnet(s) that	*We found that subnets 31 through 40 had*

requires the most number of hosts and ending with the subnet(s) that require the least number of hosts, repeatedly choose the next available subnet identifier that is as long as possible, but still supports the needed number of hosts for each subnet. With each choice, do not re-use any subnet identifier that has already been used.	*the next highest number of hosts, which was 10 hosts. Using 10 bits for the subnet identifier left 4 bits for the host identifier, which was enough to accommodate 10 hosts per subnet. So, we started with the next subnet identifier from step i, 011110, to avoid reusing any subnet identifier already used. We then appended the 4 additional bits, and started creating more subnets with those additional bits. That gave us subnet identifiers 0111100000 through 0111101001 for subnets 31 through 40. In this step, we did something different using VLSMs than we would have done by using a fixed length subnet mask.*
iii) Beginning with the subnet(s) that requires the most number of hosts and ending with the subnet(s) that require the least number of hosts, repeatedly choose the next available subnet identifier that is as long as possible, but still supports the needed number of hosts for each subnet. With each choice, do not re-use any subnet identifier that has already been used.	*We found that subnets 41 through 50 had the next highest number of hosts, which was 5 hosts. Using 11 bits for the subnet identifier left 3 bits for the host identifier, which was enough to accommodate 5 hosts per subnet. So we started with the next subnet identifier from step ii, 01111010100, to avoid reusing any subnet identifiers. We then appended an additional bit, and created more subnets. This gave us subnet identifiers 01111010100 through 01111011101.*

STEPS

1. So now you have seen a process to create VLSMs, and have seen some examples as well. Let us look take a look at another small example, and then you will be given the opportunity to try one yourself.

   Imagine that an organization is assigned the address block 62.44.0.0/16, and the organization needs one subnet with 100 hosts, one subnet with 25 hosts, and one subnet with 4 hosts. Let us apply the process established in the overview section to create these subnets.

   First, we determine that the subnet with 100 hosts is the one with the most hosts. The closest number over 100 that is also a power of 2 is 128 which requires 7 bits. Remember that we are not representing 128 itself with bits, which would be

10000000 and require 8 bits, but we are representing numbers 0 through 127, which gives us 128 possibilities, so we only need 7 bits for 128 possibilities. An easy way to think about this is that 2^7 = 128, so 7 is the number of bits that we need.

So we decide that the first subnet needs 7 bits for the host identifier. To determine the length of the subnet identifier, we simply subtract the length of the network prefix, 16 bits, and the length of the host identifier, 7 bits, from 32. We find that 32 – 16 – 7 is 9, so we will use 9 bits for the subnet identifier. A 9-bit subnet identifier is the longest subnet-identifier we can use and still support 100 hosts on that subnet.

Putting this together with the network prefix, the first subnet's network identifier would thus be:

First Subnet: 00111110 00101100 00000000 0

Now we repeat this same process for the next largest subnet, which we determine to be the subnet with 25 hosts. The closest number over 25 that is also a power of 2 is 32, which requires 5 bits for representation, so we will need 5 bits for the host identifier. 32 – 16 – 5 is 11 bits, so we will use 11 bits for the subnet identifier for this subnet.

Because we have already used the 9-bit subnet identifier 000000000, we start with the next 9-bit subnet identifier, 000000001, then include the two additional bits. In doing so, our second subnet's network identifier is:

Second Subnet: 00111110 00101100 00000000 100

We repeat the process again for the next largest subnet, which is the last subnet which requires 4 hosts. The closest number over 4 which is a power of 2 is 8, which requires 3 bits. You may have been tempted to try and use 2 bits, since 2 bits gives us 4 possible values, but it is important to remember that the network address and the broadcast address cannot be assigned to a host within the subnet. So if we had used 2 bits, we would only have 2 available host addresses and not 4, and 4 host addresses is required. So we use 3 bits for the host identifier, which leaves us with 13 bits for the subnet identifier (32 – 16 – 3 = 13).

Because we have already used the 11-bit subnet identifier, 00000000100, we start with the next 11-bit subnet identifier, 00000000101, then append the additional 2 bits.

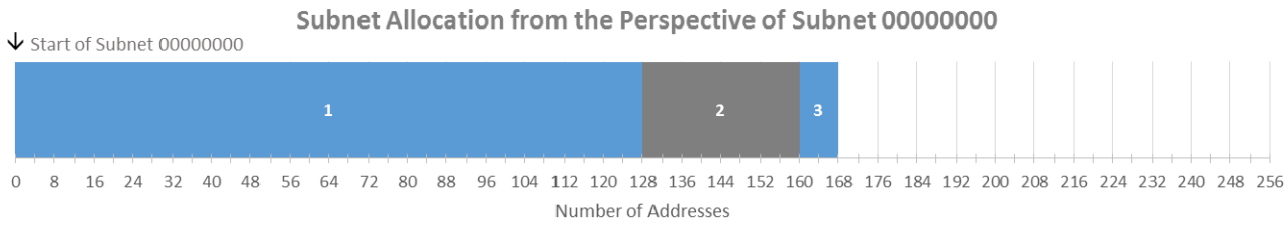
Third Subnet: 00111110 00101100 00000000 10100

So in summary the subnets' network identifiers are as follows:

```
First Subnet:  00111110 00101100 00000000 0
Second Subnet: 00111110 00101100 00000000 100
Third Subnet:  00111110 00101100 00000000 10100
```

For each subnet, we repeatedly choose the longest subnet identifier possible that supports the needed number of hosts. With each choice, we make sure not to re-use any subnet identifier that has already been used.

Take a look at the graphical representation of how would-be subnet with identifier 00000000 was subdivided into smaller subnets.



Notice that subnet 1 uses the first 128 addresses, subnet 2 uses the next 32 addresses, and subnet 3 uses the next 8 addresses.

2. Imagine that an organization is assigned the address block 169.222.0.0/18, and the organization needs one subnet with 220 hosts, one subnet with 128 hosts, and one subnet with 19 hosts.

   Create the appropriate network identifiers for each subnet, using the process established for VLSMs in the overview section. Refer to step 1 for a similar example. Make sure to show your work.

3. Explain in your own words how the process of allocating subnets using VLSMs differs from the process of allocating subnets using a fixed length subnet mask.

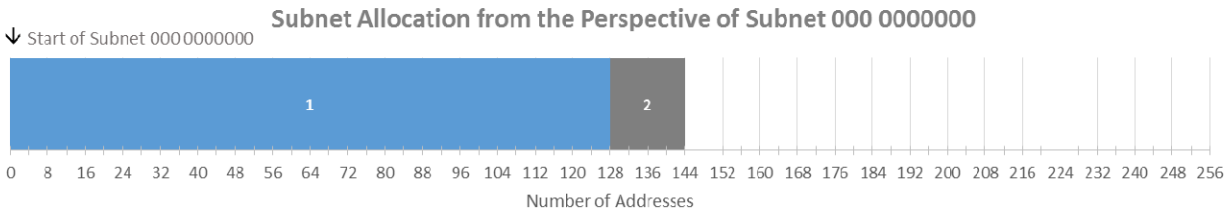
4. Great! You worked through the VLSM process in step 2. Let us now explore what can happen when a new subnet needs to be allocated on an existing network. The VLSM process described in the overview section starts with the largest subnets and works its way down to the smallest subnets. This ensures efficient subnet allocation and avoids gaps of unallocated addresses. If the new subnet is larger than some existing subnets, there will be a gap of unallocated addresses. On the other hand, if the new subnet is the same size or smaller than the smallest existing subnet, then there will not be a gap of unallocated addresses.

   Let us take a look at an example. Imagine that an organization is assigned the block of addresses defined by 99.216.0.0/13. Further imagine that two subnets are already allocated. The first subnet needs 100 hosts and the second needs 13 hosts, and were

allocated with network identifiers as follows:

First Subnet: 01100011 11011000 00000000 0
Second Subnet: 01100011 11011000 00000000 1000

Would-be subnet with identifier 000 0000000 is thus subdivided as follows.

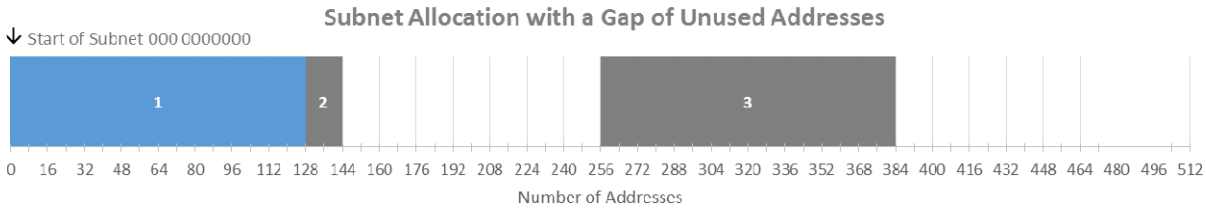


Notice that in this example, subnet with identifier 000 0000000 supports 9 bits for the host identifier, which allows for 512 total addresses. However, in the preceding figure, only 256 of those 512 addresses are displayed for illustrative purposes.

Now imagine that the organization needs a new subnet the same size as the first subnet, 100 hosts. We know from the first subnet calculation that the subnet identifier would be 12 bits in length. So what do we do now? Applying the process we are already familiar with, we choose the next available 12-bit subnet identifier that has not been used previously to create a network identifier of:

Third Subnet: 01100011 11011000 00000001 0

We now have a gap of unallocated addresses, illustrated in the following figure.



Notice that subnet 3 is allocated beginning at address 256, and that addresses 144 through 255 are not allocated. Because of the properties of binary numbers, we must start a 128-bit subnet identifier on a value is divisible by 128. That is, we cannot start the 128-bit subnet identifier at the next available address, 144.

In such a situation, it is tempting to get rid of all existing subnet identifiers and to allocate the subnets from scratch. While doing so would result in an efficient allocation scheme, in reality this is rarely practicable, because it would mean a change in IP address for existing client computers, routers, and devices throughout the organization, which would be significantly disruptive. So the network is

necessarily left with some unallocated addresses.

In summary, when a new subnet must be added to an existing network, and that subnet is larger than at least one existing subnet, applying the process of creating VLSMs covered in the overview section will result in an unavoidable gap of unallocated addresses.

5. Imagine that an organization is assigned the address block 45.0.0.0/8, and the organization has an existing subnet with 190 hosts, and another existing subnet with 53 hosts. Further imagine that after the network has been in used for some time, the organization must add another subnet that requires 105 hosts.

   First, use the process established in the overview section to create the network identifiers for the two existing subnets. Second, use the same process to create the network identifier for the new subnet (illustrated in step 4). Last, identify the number of addresses that are left unallocated in the gap between the second and third subnet. Make sure to show your work.

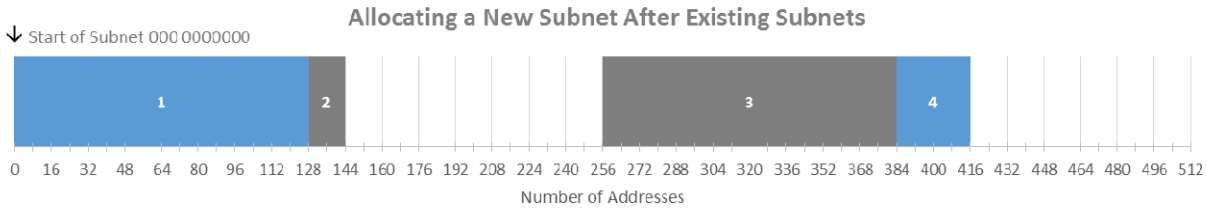
6. Explain in your own words why allocating a new subnet on an existing network results in an unavoidable gap of unallocated addresses.

7. So now you have learned and applied the process of creating subnets using VLSMs on an empty network, and the process of adding a new subnet to an existing network. Good work!

   In step 4 we found that adding a subnet to an existing network sometimes unavoidably results in a gap of unallocated addresses. However, it is important to note that these addresses need not necessarily remain unused forever. If the organization needs to add a new subnet, and the size of that subnet fits into the block of unallocated addresses, we can allocate the subnet in the gap.

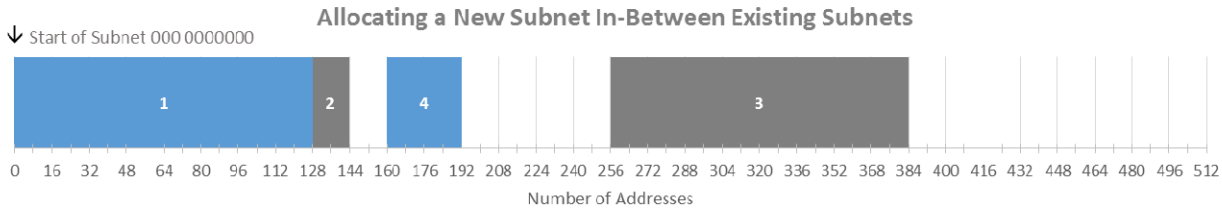
   Continuing with the example in step 4, where the organization originally had 2 subnets allocated then added a third, imagine that the organization needs yet another subnet, which has 29 hosts. This new subnet requires 5 bits for the host identifier to support this number of hosts, and therefore a 14-bit subnet identifier (32 – 13 – 5). We could start with the third subnet's network identifier (01100011 11011000 00000001 0) and increment it by 1, then add the additional bits to arrive at the new network identifier of 01100011 11011000 00000001 100.

   If we use the network identifier just described, the new subnet would be allocated as follows.



While this is a valid choice of a network identifier for the new subnet, this leaves the unallocated gap of addresses untouched and is therefore suboptimal.

To ensure that we are laying out our network in the most efficient manner, we can instead allocate the subnet in the block of unallocated addresses. Since the new subnet spans 32 addresses, we just need to be sure it starts on a number divisible by 32. In this case, address 160 is a number divisible by 32 and so we choose to use that address. Now in terms of bits, we are literally choosing the next available but unused 14-bit subnet identifier, which is *not* the subnet identifier *after* subnet 3, but is the located in-between subnet 2 and subnet 3. Our goal is to allocate subnet 4 as shown in the following figure.



Notice that with this allocation, subnet 4 has been allocated in-between subnets 2 and 3. This is different than what we have seen thus far, but perfectly appropriate when assigning new subnets to existing network infrastructures.

So how do we go about achieving this goal for the new subnet allocation? Let us take a look again at both existing network identifiers:

```
Second Subnet: 01100011 11011000 00000000 1000 (15 bits)
Third Subnet:  01100011 11011000 00000001 0    (12 bits)
```

Did you spot where we should start? We start not by incrementing the third subnet's identifier, but by increment the 14th bit of the second subnet's identifier to the next available identifier, like so:

```
Fourth Subnet: 01100011 11011000 00000000 101
```

Let us compare these two carefully so that it is very clear what just happened.

```
Second Subnet: 01100011 11011000 00000000 1000
Fourth Subnet: 01100011 11011000 00000000 101
```

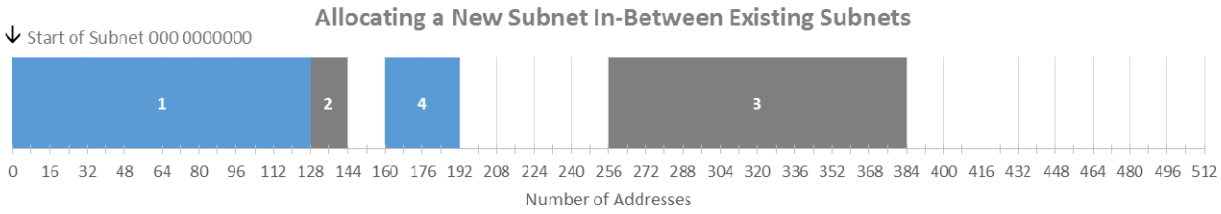
Notice that the second and fourth subnets' network identifiers are identical up to the 3rd bit in the last octet. The second subnet effectively subdivides the 14-bit subnet 000 00000000 100. And, for the fourth, new subnet, we just need to use the next available 14-bit subnet, which is 000 00000000 101.

This would make the entire organization's allocation look as follows:

```
First Subnet:  01100011 11011000 00000000 0
Second Subnet: 01100011 11011000 00000000 1000
Third Subnet:  01100011 11011000 00000001 0
Fourth Subnet: 01100011 11011000 00000000 101
```

Causally we can say that the first subnet uses the *first* 12-bit subnet identifier in full (effectively subdividing the first 11-bit subnet identifier). The second subnet subdivides the *second* 12-bit subnet identifier with 3 additional bits. The third subnet subdivides the *second* 11-bit subnet identifier. The fourth subnet further subdivides the *second* 12-bit subnet identifier with addresses unused by the second subnet. From this casual description, you can probably discern that allocating subnets using VLSMs is all about perspective. We can treat each subnet as its own network and further subdivide it when necessary.

Graphically, the four subnets now match our goal.



And if the organization were to need another subnet that would fit between subnets 2 and 4, or between subnets 4 and 3, we could allocate the subnet in the gap of unallocated addresses.

Now to be sure, even with this type of allocation, we have not violated the process given in the overview section, which states:

> …repeatedly choose the next available subnet identifier that is as long as possible, but still supports the needed number of hosts for each subnet. With each choice, do not re-use any subnet identifier that has already been used.

For the fourth subnet, we simply chose the next available 14-bit subnet identifier that was unused. The result of this choice was a little different than what we had

seen in previous steps, because the new subnet was allocated in-between two existing subnets. Regardless, the next available, unused 14-bit subnet identifier was chosen.

8. Now it is your turn to allocate a new subnet in-between existing subnets on a network. In the three parts below, keep in mind that there will be a gap of unallocated addresses in your subnet allocation, similar to the example given in step 7.

   a) Imagine that an organization is assigned the address block 222.221.128.0/17, and the organization has an existing subnet with 107 hosts, and another existing subnet with 51 hosts. Using the process given in the overview section, allocate both subnets and identify the network identifier for both subnets. Make sure to show your work.

   b) Now the organization decides to add an additional subnet that supports 113 hosts. Allocate this subnet and identify its network identifier, making sure to show your work.

   c) Last, the organization decides to add one additional subnet that supports 9 hosts. Allocate this subnet and identify its network identifier, making sure to show your work.

9. Explain in your own words how the process of allocating a new subnet in-between existing subnets works.

The advanced material you learned in this lab is a significant milestone in learning to work with modern networks on a day-to-day basis. VLSMs are commonly used on networks today. Learning to do this type of work is difficult; however, with this knowledge and these skills, you are now able to intelligently dialog with other networking professionals about many facets of an organization's computer network. Further, you are able to make intelligent decisions on how to allocate, or reallocate if necessary, subnets and IP addresses across an organization. Congratulations!