4.4.1 Calculate in bases specified in 3.5.3

For binary and hexadecimal calculations, only addition is required.

JSR Notes. And you'll note they are not in the normal Times New Roman font, rather, Monaco since its characters are evenly spaced, and that makes for easier to read binary additions and so on.

Adding binary:

Binary Addition Rules # 1 & 2: 0 + 0 = 0 and 0 + 1 = 1

```
  0000 0000       1010 1010
+ 0000 1010     + 0101 0101
-----------     -----------
  0000 1010       1111 1111
```

Binary Addition Rule # 3: 1 + 1 = 10.

This makes sense, since the binary representation of 1 plus the binary representation of 1 equals the binary representation of 2, which is 10.

But, note that as you work your way across the places from right to left, you need to carry the 1, just like you do when adding decimal numbers.

```
       1          1 1  1 1
  0000 0011       0101 0101
+ 1100 0010     + 0101 0101
-----------     -----------
  1100 0101       1010 1010
```

Overflow Error

Because of the above binary addition rule, an overflow error can happen due to carry over past what the data type can accommodate.

If the carrying goes past the left most place, you'll have a problem. For example, following is an addition of two 8-bit binary numbers, where there is a carry over of the addition of the last two (left-most) digits. And so this causes overflow "outside of", or "over" the "container" (the "container" in this case being an 8-bit byte.)

```
 1     1
  1001 0101  (Decimal 133)
+ 1000 0100  (Decimal 132)
-----------
  0001 1001  (Decimal  25, which is not 133 + 132!)
```

The above answer is 0001 1001 because it cannot be a 9-digit answer. Most modern

compilers will pick up this error. But anyway, you should always pick data types that will comfortably accommodate the values that will likely be worked with. And, so in the example above, we should have know better than to try to add two values so large, when the "container" we are working with - an 8-bit byte - can only hold, max., 256 (2 ^ 8) unique values.

Binary Addition Rule # 4

1 + 1 + 1 = 11.
This makes sense, since the binary representation of 1 plus the binary representation of 1 plus the binary representation of 1 equals the binary representation of 3, which is 11.

The reason we'll need to keep this rule in mind is that often when we carry a 1, the two values to be added at that place will be 1 and 1. So we carry a 1 to the next place, but also leave a 1 at that place:

```
      1          111  11
  0000 0100       0111 0110
+ 0000 0100     + 0111 0110
-----------     -----------
  0000 1100       1110 1100
```

Adding Hexadecimal

The first thing you need is a visual. This will take approximately 5.6 seconds to scribble down on an IB exam: 5.6 seconds well spent.

Hexadecimal	Decimal Equivalent
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
A	10
B	11
C	12
D	13
E	14
F	15

The second thing you need is a reminder of what value each place is:
Well, that's pretty straight forward:

```
....16^4  16^3  16^2  16^1, or
...4096s  256s   16s    1s
```

And really, this is not fundamentally different than any numbering system:

Decimal (Base 10):
... 1000s 100s 10s 1s
(For example decimal 7345 is 7 groups of 1000s, 3 groups of 100s, 4 groups of 10s, and 5 1s.)

Binary (Base 2):
... 8s 4s 2s 1s
(For example, binary 1010 is 1 group of 8s, 0 groups of 4s, 1 group of 2s and 0 groups of 1s.)

Hexadecimal (Base 16):
... 4096s 256s 16s 1s
(For example, hex. 1234 is 1 group of 4096s, 2 groups of 256s, 3 groups of 16s, and 4 1s.)

Adding hexadecimal values, as with adding decimal values or adding binary values is only tricky when you need to carry values over to the next place. Just remember you carry when you reach a total of decimal 16, since no place can represent any more than 15 of that place - i.e., no more than 15 (F) groups of 1s, no more than 15 (F) groups of 16s, no more than 15 (F) groups of 256s etc.