Interpreting bit Combinations

We recognise that we have only a fixed number of bits to represent information in our Computer. This number may be 32-bits, 64-bits, or something else. Regardless of what the number is, however, it will be fixed at design time.

Given a particular Interpretation of the bits we are manipulating, each bit-Combination refers to a specific Piece of Information within that Intrepretation.

Therefore, if we are Interested in representing natural numbers we see each Combination of bits as a binary number from $0 \rightarrow 2^n - 1$, where n is the number of bits in the Combination.

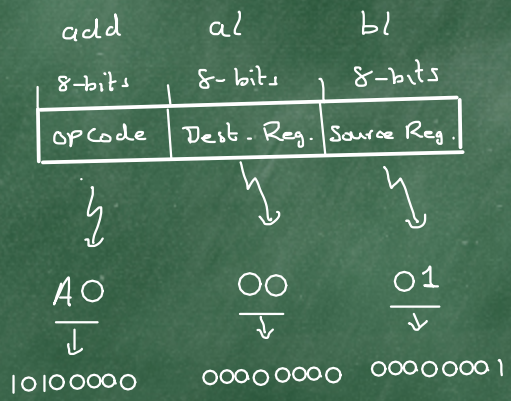
If we have 4-bits, for example we can represent numbers from $0 \rightarrow 2^4 - 1$ i.e., from $0 \rightarrow 15$

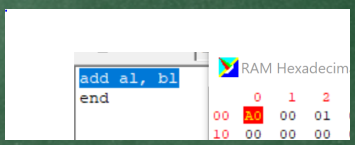
Likewise, if we are interested in representing instructions, we see each combination as composed of different fields whose corresponding binary value represents an operation code and register numbers for example.

Thus, as we have seen, the Samphire instruction:

```
add al, bl
```

Is 24-bits long and consists of 3 fields, each 8-bits long:





Representing Negative Numbers

Since, for a given number of bits, n, we are limited to 2^n combinations, if we wish to represent both Positive and negative numbers, we must share the Combination among them.

2^n Is alway even. Therefore, we could Consider having 2^{n-1} Positive numbers and 2^{n-1} negative numbers.

What about 0?

if we Consider 0 as Positive, and we decide that our Positive numbers are at the start of the start of the binary Counting Sequence, our Positive range would be from $0 \rightarrow 2^{n-1}-1$. (eg. from 0–7 if n=4)

Similarly, our negative range would extend from 2^{n-1}.

Let's take $n=4$ for some Insight:

2^3 Positive numbers Including 0.

0 0 0 0	1 0 0 0
0 0 0 1	1 0 0 1
0 0 1 0	1 0 1 0
0 0 1 1	1 0 1 1
0 1 0 0	1 1 0 0
0 1 0 1	1 1 0 1
0 1 1 0	1 1 1 0
0 1 1 1	1 1 1 1

2^3 negative, non-zero numbers

We see that if we are to have only one value for 0, that the number of non-zero Positive numbers (2^3-1) and the number of non-zero negative numbers (2^3) differ by 1.

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	?
1 0 0 1	?
1 0 1 0	?
1 0 1 1	?
1 1 0 0	?
1 1 0 1	?
1 1 1 0	?
1 1 1 1	?

How should we allocate the negative numbers? →

Things to note:

The MSB for the +ve numbers = 0

The MSB for the -ve numbers = 1

→ Easy way to distinguish +ve from -ve

Well, it would be nice if

$$x + (-x) = 0$$

Let's try the following:

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	-8
1 0 0 1	-7
1 0 1 0	-6
1 0 1 1	-5
1 1 0 0	-4
1 1 0 1	-3
1 1 1 0	-2
1 1 1 1	-1

Things to note:

There is no positive number corresponding to -8 (i.e., corresponding to -2^{n-1})

Check for $x + (-x) = 0$

Try: $5 + (-5)$

$= 0101$

$+ 1011$

$1\ 0000$

Carry out ↗ 4-bit answer = 0

Same for every $x, -x$ pair

Try $5 + (-3)$

$= 0101$

$+ 1101$

$1\ 0010$

4-bit answer = 2

Try $3 + (-5)$

$= 0011$

$+ 1011$

$0\ 1110$

Carry out ↗ 4-bit answer = -2

$7 + 1 = -8$!

$-8 - 1 = 7$! Boundary Problems overflow, underflow

This, unbalanced, Convention for representing negative numbers is called two's Complement notation.

Binary Sequence	Unsigned Interpretation	Signed Interpretation	Hex
0 0 0 0	0	0	0
0 0 0 1	1	1	1
0 0 1 0	2	2	2
0 0 1 1	3	3	3
0 1 0 0	4	4	4
0 1 0 1	5	5	5
0 1 1 0	6	6	6
0 1 1 1	7	7	7
1 0 0 0	8	-8	8
1 0 0 1	9	-7	9
1 0 1 0	10	-6	A
1 0 1 1	11	-5	B
1 1 0 0	12	-4	C
1 1 0 1	13	-3	D
1 1 1 0	14	-2	E
1 1 1 1	15	-1	F
2^n Combinations	$0 \to 2^n - 1$	$0 \to 2^{n-1} - 1$; $-2^{n-1} \to -1$	A = 10 if unsigned, A = -6 if signed etc

For 32-bits:

000...0000	0	0	0
000...0001	1	1	1
⋮		⋮	⋮
0111...1111	2,147,483,647	2,147,483,647	7FFFFFFF
100...0000	2,147,483,648	$-2,147,483,648$	80000000
100...0001	2,147,483,649	$-2,147,483,648$	80000001
⋮	⋮	⋮	⋮
1111...1110	4,294,967,295	-2	FFFFFFFE
1111...1111	4,294,967,296	-1	FFFFFFFF
2^{32} Combinations	$0 \to 2^{32}-1$	$0 \to 2^{31}-1$ $-2^{31} \to -1$	

From the foregoing, we note that we could implement subtraction using addition by negating the second operand and then adding it to the first.

But....

Question: Is there a simple algorithm that will negate a binary number?

In other words, how would we form the two's complement of a binary number?