# 50.002 Computational Structures

Information Systems Technology and Design

# Basics of Information

Natalie Agus (Fall 2018)

## 1 What is this course about?

To know how computer works from the bottom up: from

1. Binary data, to
2. MOSFET,
3. Gates,
4. Cells,
5. Modules,
6. Integrated circuits,
7. Circuit boards, and
8. Finally the personal computer.

## 2 Information and uncertainty

The amount of information (required/needed/given/revealed/acquired) is **inversely proportional to** the probability $p$ of that event happening,

$$\text{Information} \propto \text{Uncertainty} \propto \frac{1}{p}. \tag{1}$$

Equivalently, it is **proportional to** the uncertainty of that event happening. *In laymen terms: If an event is bound to happen, then the fact that the event happens does not give any kind of information..*

For discrete events $(x_1, x_2, ..., x_N)$ with probability of occurence of $(p_1, p_2, ..., p_N)$, the basic measure of information for all of these events is the ***bit***.

How many bits is needed to reveal that a random variable is $x_i$?

$$I(X) = \log_2 \frac{1}{p_i} \text{ bits,} \tag{2}$$

where:
$I(X)$ is the amount of information received in bits learning that the choice was $x_i$.

With $N$ equally probable choices, if it is narrowed down to $M$ choices $(N > M)$ then we are given,

$$I_{N \to M}(X) = \log_2 \left(\frac{N}{M}\right) \text{ bits,} \tag{3}$$

of information.

Example:
Let $N = 4$, and all 4 events are equally probable. The number of bits needed to encode all 4 events are:

$$I(X) = \log_2 \frac{1}{1/4} = 2 \text{ bits.}$$

We need to receive 2 bit of informations to learn that one event $x_i$ out of the 4 events happens. In this case, it can be $(0, 0), (0, 1), (1, 0)$, or $(1, 1)$.

# 3 Fixed Length Encoding (FLE)

The FLE is used in practice when all choices $x_i$ are **equally probable**.

**Decimal** : 4-bit to represent 1 to 10
**Characters**: 7-bit ASCII for 86 english characters
**16-bit unicode**: for other language alphabets that are fixed, e.g: Russian, Korean

How to encode numbers in binary? Suppose we have binary number:

```
0 0 1 1 0 1
```

The above means $0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$.

What if encoding in binary is too long on paper? We can encode in octal (groups of 3 binaries) or hex (groups 4 binaries). See the table on the last page on the conversion between binary, octal, hex, and decimal.

| Hex | Binary | Octal | Decimal |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| A | 1010 | 12 | 10 |
| B | 1011 | 13 | 11 |
| C | 1100 | 14 | 12 |
| D | 1101 | 15 | 13 |
| E | 1110 | 16 | 14 |
| F | 1111 | 17 | 15 |
| 10 | 1 0000 | 20 | 16 |
| 11 | 1 0001 | 21 | 17 |
| 24 | 10 0100 | 44 | 36 |
| 5E | 101 1110 | 136 | 94 |
| 100 | 1 0000 0000 | 400 | 256 |
| 3E8 | 11 1110 1000 | 1750 | 1000 |
| 1000 | 1 0000 0000 0000 | 10000 | 4096 |
| FACE | 1111 1010 1100 1110 | 175316 | 64206 |

**Figure 1**

Example 1:

1. Binary : 101 101 101 111
2. Octal : $5557_8$
3. Decimal : 2927
4. Hex: $0xB6F$

Example 2:

1. Binary : 111 110 101
2. Octal : $765_8$

3. Decimal : 501

4. Hex (pad the MSB of the binary so that it's the closest divisible by 4, in this case, we expand from 9 bits to 12 bits) $\rightarrow$ 0001 1111 0101, hence $0x1F5$

# 4 2's Complement

The purpose: so we can subtract using addition. How to make 2's complement:
**Step 1**: inverse all $0$s into $1s$ and vice versa on the original binary number
**Step 2:** add 1 **to the number in step 1**

Example:
0 0 1 1 = 3
**Step 1**: 1 1 0 0 (inversed)
**Step 2**: 1 1 0 0 + 0 0 0 1 = 1 1 0 1 (add 1)
The value in step 2 above is : $-2^3 + 2^2 + 2^0 = -3$.

**Signed bit** : the most significant bit, if 1 then negative number, if 0 then positive number.

How to encode decimal in binary? Suppose we have **signed** binary number:

```
1 0 0 1 . 0 0 1 1
```

The above means $-1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-3} + 1 \times 2^{-4}$.

# 5 Binary Number Range

Given $X$ bits,

1. We can encode $2^X$ *choices, or random variables*

2. If it is unsigned bit, we can represent the number ranged from 0 to $2^X - 1$

3. If it is signed bit, we can represent the number ranged from $-2^{X-1}$ to $2^{X-1} - 1$