Data Representation and Arithmetic

Gabriel Hugh Elkaim
Data Representation

In books...

On audio and video disks...

In paintings or drawings...

On tape...

In the human memory...

How to build + deliver H-bombs

In diagrams, etc!
Data Representation

**Goal:** Store information (numbers, characters, ...)

- in binary
### Data Representation

**Integers**

Integers, or whole numbers, if they are not too large, are encoded in straight binary. For instance, 185 would become 10111001.

**Binary Coded Decimal**

Binary coded decimal represents a number in decimal, but with each digit encoded in binary. For instance, 967 would become 1001 0101 0111 or 9 6 7.

**Floating Point Representation**

Floating point representation is for large or fractional numbers. For example, 19,700.0302 would be encoded as the binary equivalent of 10111100 00000000 01111011 00000000, meaning 197 x 10^5. Floating point representation often involves rounding off.
BIT is an abbreviation of "BINARY DIGIT." It refers to a single 0 or 1.

It's very common to group bits eight at a time, and any string of eight bits is called a

BYTE.

There are $2^8$, or 256, possible bytes, from 00000000 to 11111111.
Storing Information (n-bits)

\[
\begin{array}{c|c}
T & 1 \\
F & 0 \\
\end{array}
\quad \begin{array}{c|c}
\text{High} & 1 \\
\text{Low} & 0 \\
\end{array}
\quad \text{More Bits} \quad 1024
\]

25 bits

\[
\begin{array}{c}
11 \\
10 \\
01 \\
00 \\
\end{array}
\}
\text{4 items}
\]

3 bits

\[
\begin{array}{c}
111 \\
110 \\
101 \\
000 \\
\end{array}
\}
\text{8 items}
\]

N-bits \rightarrow 2^n

8 bits = 256
16 bits = 65536
32 bits = 1294967296
64 bits = 1.84467 \times 10^{19}

Gabriel Hugh Elkaim
CMPE-012/L
Integer Representation (1.2)

- Assume I have a fixed number of bits (32)
- Which 4 billion integers do I want?

→ UNSIGNED → 0 → 4 Billion consecutive unsigned integers

→ SIGNED ↓ 2's complement
   ↓ 1's complement
   ↓ Biasing / SHL
Integer Representation (2.2)

Unsigned

2's Complement

Most Negative

Most Positive Number

-128 0 127
1 megabyte — 16 bit — 65K — ~1 minute
32 bit — 48 — ~50 days
Unsigned Integers (1.2)

\[ \begin{align*}
 0000 & \rightarrow 0 \\
 0001 & \rightarrow 1 \\
 \vdots & \\
 1111 & \rightarrow 15
\end{align*} \]

Encodes only positive integers

Range: \( 0 \rightarrow 2^n - 1 \)
Unsigned Integers (2.2)

4 bit unsigned int.

\[ n = 4 \Rightarrow 2^n - 1 = 16 - 1 = 15 \]

\[ 7 \rightarrow 0111 \]

17 \[ \begin{array}{c}
10001
\end{array} \]

-3 \(?\)

32 bits \[ \rightarrow 0 - 2^{32} - 1 = [0 : 4,294,967,295] \]
One’s Complement

Flip all the bits

1011
0100

1011 0111
0100 10100

0000 0111
7 1000 1

-
One’s Complement Representation
Two’s Complement (1.2)

- Top bit is set if number is < 0

- Additive inverse of a number

- Take the number (in binary)
  1. Flip all the bits
  2. Add 1.
Two’s Complement (2.2)

4 bits $\rightarrow (-5)$

\[
\begin{align*}
1 & \quad 1 & \quad 1 & \quad 1 \\
0 & \quad 1 & \quad 0 & \quad 1 \\
+ & \quad 1 & \quad 0 & \quad 1 & \quad 1 \\
\hline
1 & \quad 0 & \quad 0 & \quad 0 & \quad 0
\end{align*}
\]

\[
\begin{align*}
0101 & \quad (5) \\
1010 & +1 \\
\hline
1011 & \quad (-5)
\end{align*}
\]

\[
\begin{align*}
1011 & \quad (5) \\
01\infty & +1 \\
\hline
0101 & - (5)
\end{align*}
\]
Two’s Complement Conversion

-12 in 8-bit 2's complement.

12
1100 - "c"

0000 1100 (12)

1111 0011 +1

0000 10100 (-12)

0000 1100

CARRY OVER
Two’s Complement Addition (1.2)

1000 0000
FF
-128

0

0

3F

255
Two's Complement Addition (2.2)

\[-20 + 15:\]

\[\begin{array}{c}
\begin{array}{c}
0000 \\
1110 \\
1100 \\
\hline
1111 \\
1011 \\
\hline
0000 0100
\end{array}
\end{array}\]

\[15\]

\[-20\]

\[\begin{array}{c}
\begin{array}{c}
0000 1000 \\
1110 1011 \\
\hline
1110 1100
\end{array}
\end{array}\]

\[(5)\]
Subtraction (1.2)

\[
\begin{array}{c}
127 \\
-6 \\
\hline
18
\end{array}
\quad
\begin{array}{c}
01010 \\
-0101 \\
\hline
1010
\end{array}
\quad (10) \\
\quad (5)
Subtraction (2.2)

1 - 1 = 0
0 - 0 = 0
1 - 0 = 1
10 - 1 = 1
0 - 1 → need to borrow
Two’s Complement Subtraction (1.2)

Don’t subtract!! Add instead

6 Bits 3 - 4

\[ \begin{align*}
\text{000011 (3)} + &\text{11100 (-4)} \\
\underline{\text{111111 (-1)}} &\text{101000 (-2)} \\
\underline{\text{010000 (16)}} &\text{110000 (-8)}
\end{align*} \]
Two’s Complement Subtraction (2.2)

True Additive Inverse
Add Numbers
Ignore Carryout Bit

\[-10 - 3\]

\[
\begin{array}{c}
00011 \\
\text{-10 (10)} \\
\text{10110 (-10)} \\
\hline \\
\text{11100}
\end{array}
\]

\[
\begin{array}{c}
\times 10011 \\
\hline \\
\end{array}
\]
Sign Extension (1.2)
Sign Extension (2.2)

(5) 0101

(10) 1010

(-2) 1010

0 0 0 1010

0 1 0 0 1010

0 1 1 0

10 unsigned

UNSCONED
Sign Extension — Unsigned

Copy bits down
Pad upper bits w/ 0's.

0 0 1 0 1 1 0
Sign Extension — 1’s and 2’s Complement

If \( \text{it is } > 0 \rightarrow \text{same as unsigned} \)

If \( \text{it is } < 0 \rightarrow \)

Look at MSB

Replicate it out to left

Gabriel Hugh Elkaim
Digital Logic and Gates

Gabriel Hugh Elkaim
Statements are True or False

\[ P = \text{"The pig has spots."} \]

\[ Q = \text{"The pig is glad."} \]

\[
\begin{array}{c}
\text{T} & \text{F} \\
\end{array}
\]

\[
\begin{array}{c}
\text{T} & \text{F} \\
\end{array}
\]
# Truth Tables

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P AND Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P OR Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>NOT P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Truth Table

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Basic representation of a logic function.

$2^n$
**NOT**

**NOT** = The pig is NOT spotted.

This operator simply turns a statement into its opposite.

<table>
<thead>
<tr>
<th>P</th>
<th>NOT·P</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Truth Table: Inverter

<table>
<thead>
<tr>
<th>A</th>
<th>( Y = \bar{A} \lor A' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
This kind of switch is called an **inverter**, and it has a symbol, too:

\[ A \rightarrow \overline{A} \]

0 "short hand"
Truth Table: AND

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y = A \cdot B (A \land B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth Table: AND

That's why this arrangement of switches is called an AND-GATE and it has its very own symbol.
# Truth Table: NAND

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y = \overline{(A \cdot B)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Y = \overline{(A \cdot B)} = \text{NOT } (A \text{ AND } B)

Diagram:

- AND gate
- Invertor

NAND gate
Truth Table: NAND

NAND Gate, which is merely an abbreviation of "not-and."

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>NAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
# Truth Table: OR

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( Y = (A + B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth Table: OR

This is the OR-GATE and its symbol is:
Truth Table: NOR

\[ y = \overline{(A + B)} = \overline{A} \cdot \overline{B} \]

\[ y = \text{NOT}(A \text{ OR } B) \]
Truth Table: NOR

A NOR-gate is a shorthand way of writing "not or:" i.e.

\[
\begin{array}{c|c|c}
A & B & A \text{ NOR } B \\
\hline
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0 \\
\end{array}
\]
# Truth Table: XOR

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( Y = A \oplus B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\( \wedge \) if and only if two inputs are different.

**Exclusive OR**
## Multiple Input AND/OR Gates

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **ALL 0's**
- **ALL 1's**
Truth Table to Gates (1.2)
Truth Table to Gates (2.2)

1. Read each row of the truth table independently.
2. For each row where output = 1, draw a gate that matches those inputs.
3. On the output of step 2 together.
4. This always works, not optimal.
Example: XOR
Arbitrary Gate Synthesis
Binary Addition Review (1.2)

Binary calculation is simple. There are only five rules to remember:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

And the handy fifth rule:
1 + 1 + 1 = 11

As opposed to 100 sums in decimal: 9+6, 7+5, 9+3, 8+4, 4+6, etc etc etc!!!
Binary Addition Review (2.2)
Truth Table: Full Adder (1.3)
Truth Table: Full Adder (2.3)
Truth Table: Full Adder (3.3)
Boolean Algebra (1.2)
Boolean Algebra (2.2)
Boolean Algebra Properties
Boolean Algebra: Single-Variable
De Morgan’s Laws (1.4)
De Morgan’s Laws (2.4)
De Morgan’s Laws (3.4)
De Morgan’s Laws (4.4)
NAND Gates (1.4)

The Amazing NAND:
NAND Gates (2.4)

The Amazing NAND:
NAND Gates (3.4)

The Amazing NAND:
NAND Gates (Examples)
NAND Gates (Examples)
NAND Gates (Examples)
NAND Gates (4.4)

A

IS THE SAME AS


A
B

IS THE SAME AS


A
B

IS THE SAME AS


IS THE SAME AS