Bits, bytes, representation

How data is represented in a computer

Binary

- A bit represents one of two states - e.g. 0|1
- A byte is eight bits, can represent 2^8 states/values
- For more values, more bytes are needed (byte is a common unit)
- Bits can represent numbers, or anything
- Even text can be represented as numbers (ASCII)
- If you try to store a larger number than can be represented in a given number of bits, you will get a value you didn’t expect

Demo

- Jshell - interactive Java shell
- We’ll show you what can go wrong with a byte (8 bits) variable

Decimal numbers

- Base 10
- Sum of powers of 10
- Digits 0-9

$132 = 1*10^2 + 3*10^1 + 2*10^0$

Or:

```
132
||  `-> 2*10^0 =    2
|  `--> 3*10^1 =   30
 `---> 1*10^2 = +100
                  132
```

Binary numbers

- Base 2
- Sum of powers of 2
- Digits 0-1

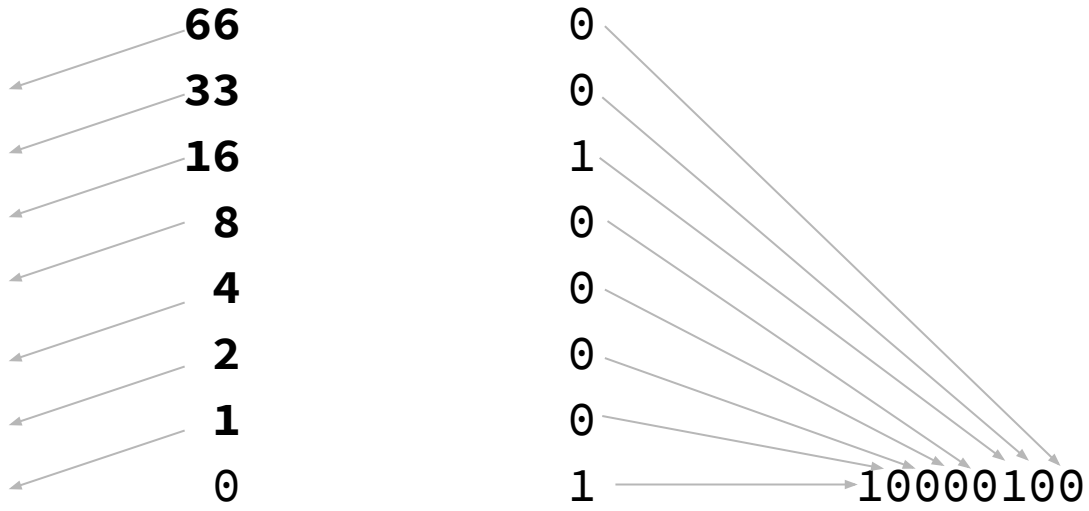
$10000100_2 = 1\star2^7 + 0\star2^6 + 0\star2^5 + 0\star2^4 + 0\star2^3 + 1\star2^2 + 0\star2^1 + 0\star2^0$

```
|||||||  `->  0*2^0 =    0
||||||  `-->  0*2^1 =    0
|||||  `--->  1*2^2 =    4
||||  `---->  0*2^3 =    0
|||  `----->  0*2^4 =    0
||  `------>  0*2^5 =    0
|  `------->  0*2^6 =    0
 `-------->  1*2^7 = +128
                      132
```

Decimal to binary

132_{10} -> divide with two, and note the remainder:

number	quotient	remainder
132	**66**	0
66	**33**	0
33	**16**	1
16	**8**	0
8	**4**	0
4	**2**	0
2	**1**	0
1	0	1



Counting on your fingers

□🖐 -> 1111111111_2 = 1023_{10}

🖐 -> 0000011111_2 = 31_{10}

✌ -> 0000000011_2 = 3_{10}

Task:

Make 132 with your hands....

Octal

- Base 8
- Sums of powers of 8
- Digits 0-7
- Relation to binary: group binary number in threes

$0001\ 1111_2 = ?_8 = 31_{10}$

```
000 011 111
 0   3   7
```

$037_8 = 31_{10}$

$3*8^1 + 7*8^0$

Hexadecimal

- Base 16
- Sums of powers of 16
- Digits 0-9, A-F → A: 10, B: 11, C: 12, D: 13, E: 14, F: 15
- Relation to binary: groups by four

$0001\ 1111_2 = ?_{16} = 31_{10}$

0001 1111
1 15

$1F_{16} = 31_{10}$

$1*16^1 + 15*16^0$

Colors in hex - exercise

- Colors in a computer usually is represented by RGB (3*8 bit colors) How many colors in total?

```
Red:   1111 1111 0000 0000 0000 0000 (FF0000)
            R          G          B
Green: 0000 0000 1111 1111 0000 0000 (00FF00)
            R          G          B
Blue:  0000 0000 0000 0000 1111 1111 (0000FF)
            R          G          B

Magenta: ? (equal parts red and blue, no green)
Black:   ? (absence of colors)
White:   ? (all colors)
```

Colors in hex

- Colors in a computer usually is represented by RGB (3*8 bit colors)
  How many colors in total?

```
Red:        1111 1111 0000 0000 0000 0000 (FF0000)
                 R         G         B
Green:      0000 0000 1111 1111 0000 0000 (00FF00)
                 R         G         B
Blue:       0000 0000 0000 0000 1111 1111 (0000FF)
                 R         G         B

Magenta:    1111 1111 0000 0000 1111 1111 (FF00FF)
Black:      0000 0000 0000 0000 0000 0000 (000000)
White:      1111 1111 1111 1111 1111 1111 (FFFFFF)
```

Plain text using ASCII

- Each character is assigned a number in a table (ASCII table)
- Nowadays 8 bit ASCII (255 characters) “extended ascii”
- Plain text can only contain the characters in the ascii table
- TAB, newline and space are characters, just like any printable
- A section is created by two consecutive newlines
  - CRLF on windows, LF on Linux/Unix

Sections

Sections are created using two newlines:

one⏎
⏎
two

01101111 01101110 01100101 **00001010 00001010** 01110100 01110111 01101111

o n e ⏎ ⏎ t w o

Wrapping text

- Note that most applications are capable of wrapping text
- Don’t over-use newline :-)
- Don’t use your own hyphenation at “line-endings” (text wraps differently on different devices)
- If you insist on breaking the lines yourself, keep things at 80 chars

Translate from binary ascii to text

```
01001000 01101111 01110111 00100000 01100100 01101111 01100101
01110011 00100000 01110100 01101000 01101001 01110011 00100000
01100010 01101001 01101110 01100001 01110010 01111001 00100000
01110100 01101000 01101001 01101110 01100111 00100000 01110111
01101111 01110010 01101011 00111111 00001010 00001010 01001100
01101001 01101011 01100101 00100000 01100001 00100000 01100011
01101000 01100001 01110010 01101101 00100001
```

https://www.sciencebuddies.org/science-fair-projects/references/ascii-table

Translate from binary ascii to text

```
$ echo '01001000 01101111 01110111 00100000 01100100 01101111 01100101
01110011 00100000 01110100 01101000 01101001 01110011 00100000 01100010
01101001 01101110 01100001 01110010 01111001 00100000 01110100 01101000
01101001 01101110 01100111 00100000 01110111 01101111 01110010 01101011
00111111 00001010 00001010 01001100 01101001 01101011 01100101 00100000
01100001 00100000 01100011 01101000 01100001 01110010 01101101 00100001' |
ascii2binary -b b ; echo
```

How does this binary thing work?

Like a charm!