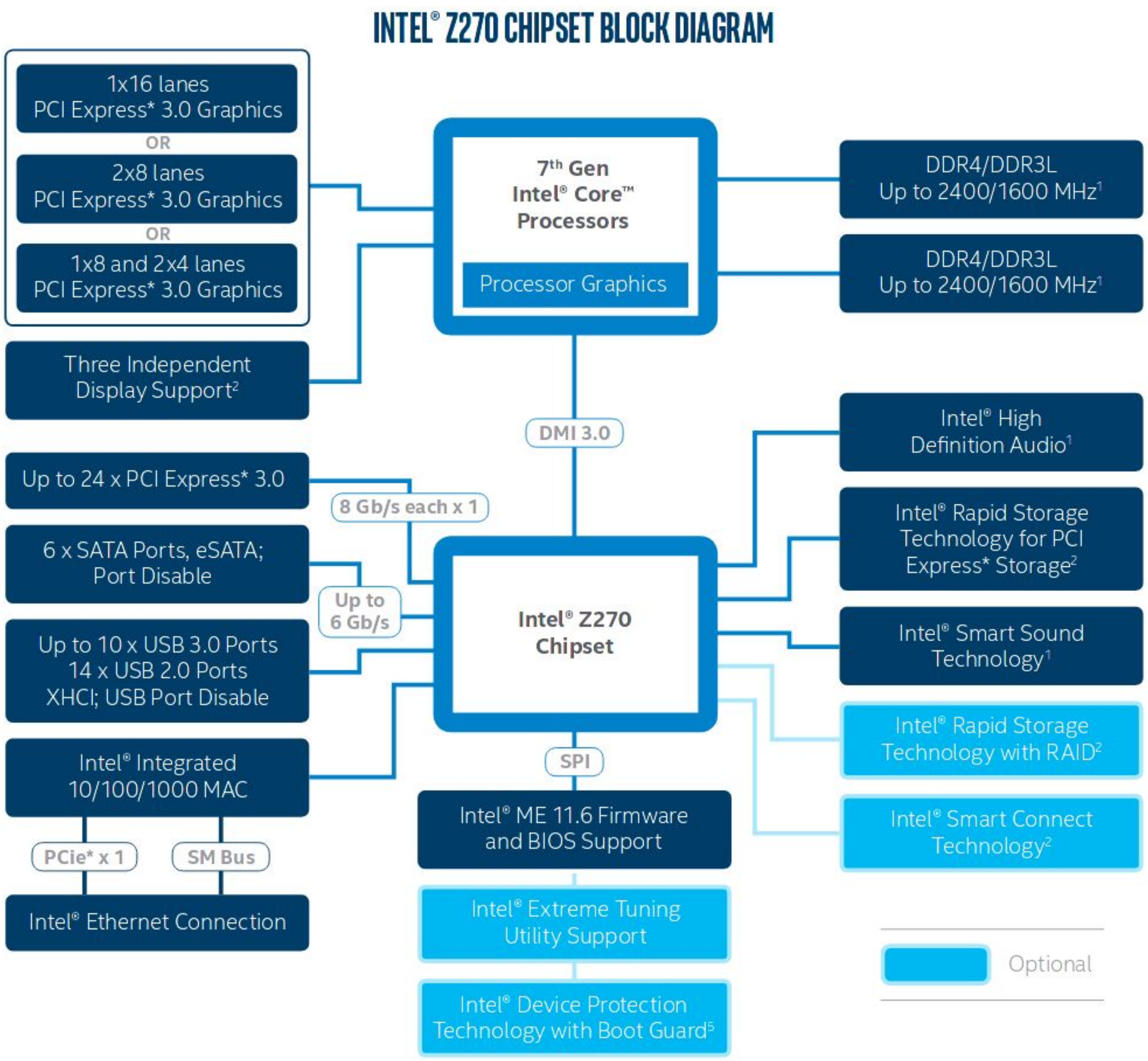
# Computer (Literacy) Skills

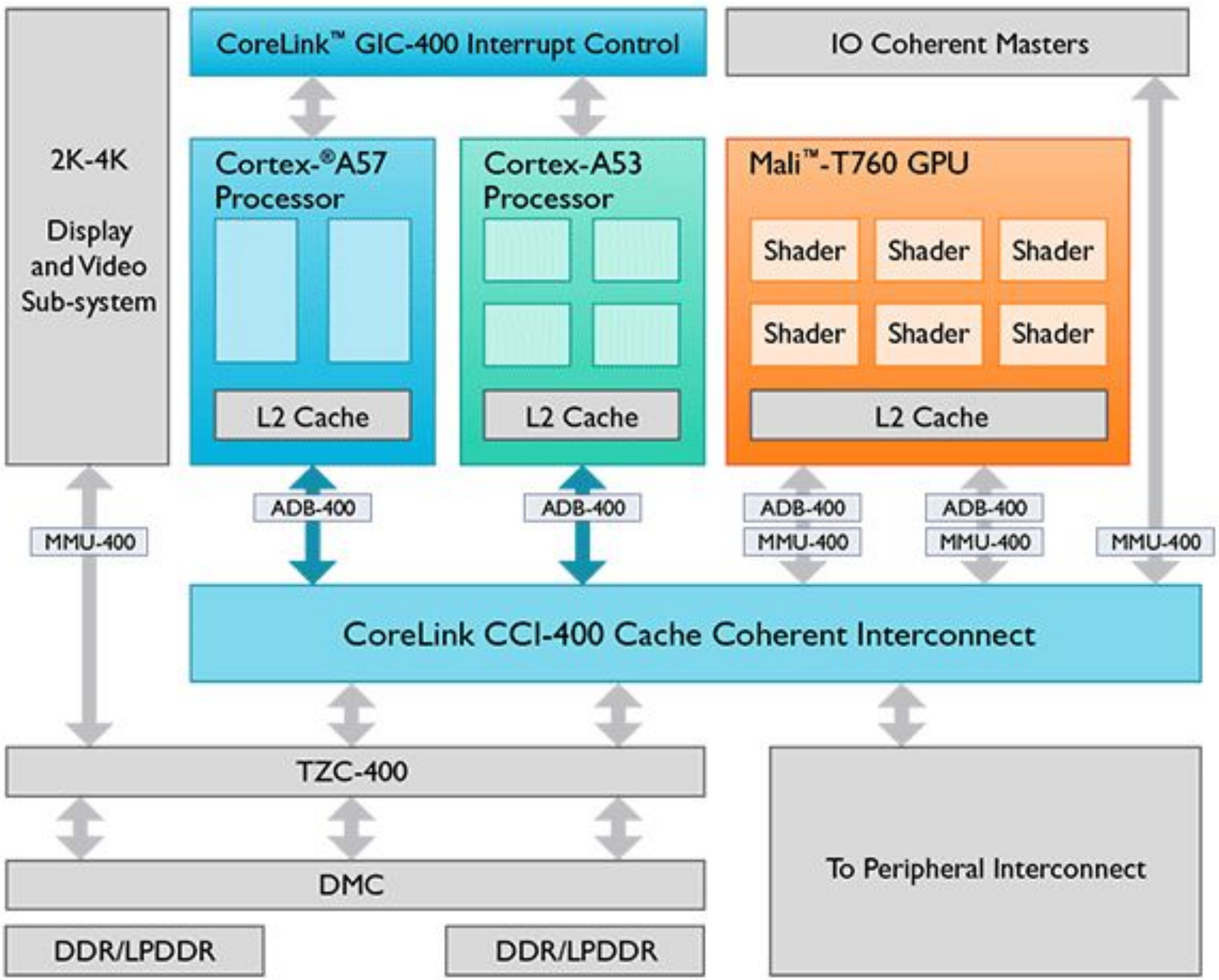
## Architecture, numbers, and operations

**Lubomír Bulej**
KDSS MFF UK

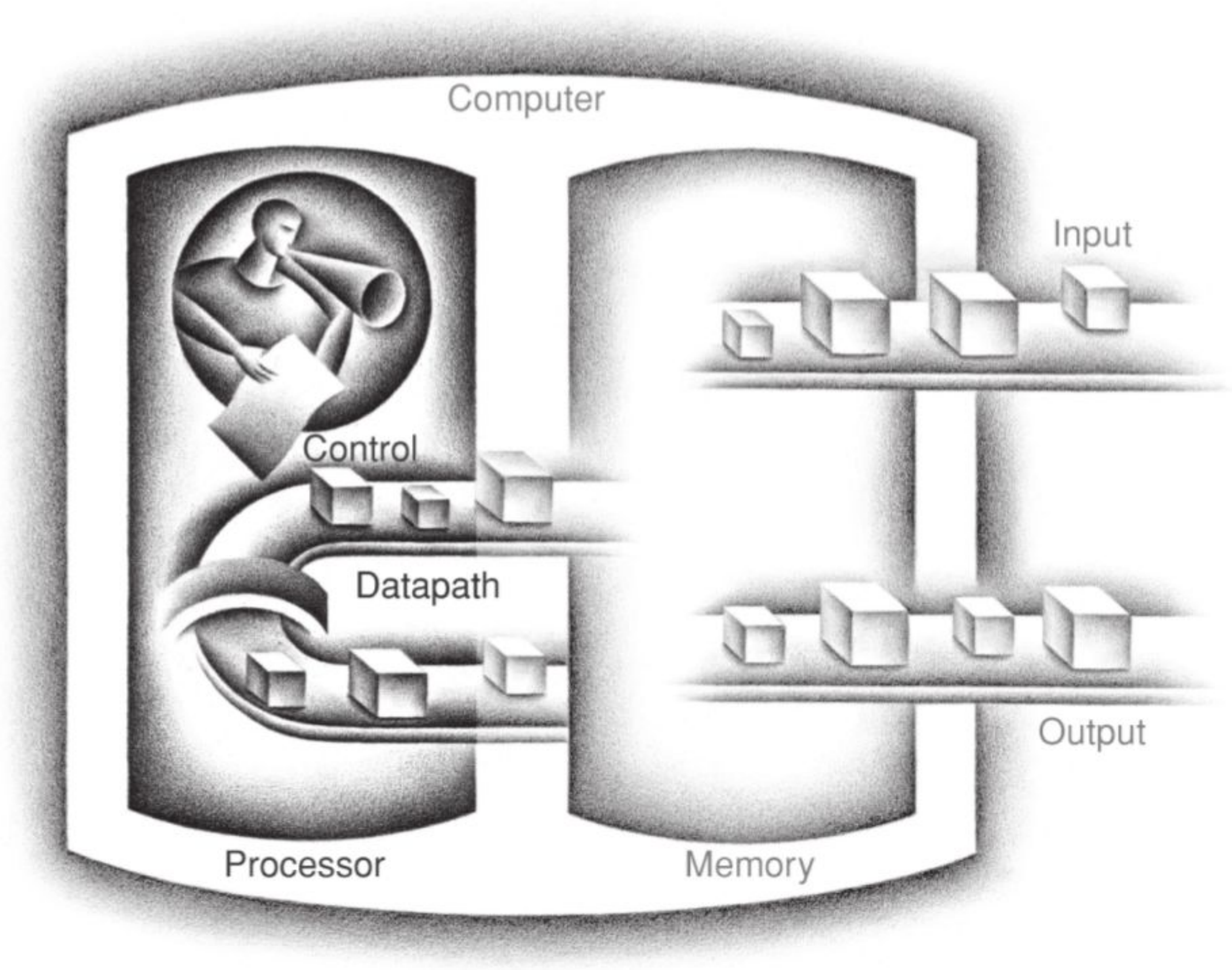
# Real computer architecture (Intel)



# Real computer architecture (ARM)



# Abstract computer architecture (H&P)



# Computers operate on numbers

**Everything is made of numbers...**

- Images, videos, music, documents

**... even programs!**

- Instructions identified by numbers

**Computer must be able to perform arithmetic**

- We can do a lot with just addition

**Additional operations make life easier**

- Multiplication, division, logical operations

# Numbers inside computer are binary

**1 bit (b) = 1 binary digit**

- Smallest unit of information
- A digit in a number (values `0` and `1`)
- A logical truth value (`0=false` and `1=true`)

**Easily represented in electronics**

- Only need to distinguish two states
- Voltage levels (difference), polarity, ...

**1 byte (B) = smallest addressable unit of memory**

- Consists of 8 bits (in modern computers)

# Representing numbers in base B

**Sequence of digits**

- Sum of positional values of all digits

**Positional value of digit $d_i$ in base B**

- $d_i \times B^i$ where $B^i$ represents weight of $d_i$

**Digit index = base power**

- $i \geq 0$ for integral part
- $i < 0$ for fractional part

**Right-to-left ordering with increasing weight**

- Digit with the highest weight is the leftmost

# Structure of a binary byte

## Bit weights

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $2^7 = 128$ | $2^6 = 64$ | $2^5 = 32$ | $2^4 = 16$ | $2^3 = 8$ | $2^2 = 4$ | $2^1 = 2$ | $2^0 = 1$ |
| $b_7$ (MSB) | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ (LSB) |

- MSB = Most Significant Bit (highest weight)
- LSB = Least Significant Bit (lowest weight)

## Byte value

$$b_7\times2^7 + b_6\times2^6 + b_5\times2^5 + b_4\times2^4 + b_3\times2^3 + b_2\times2^2 + b_1\times2^1 + b_0\times2^0$$

## Alternatively

$$((((((b_7\times2 + b_6)\times2 + b_5)\times2 + b_4)\times2 + b_3)\times2 + b_2)\times2 + b_1)\times2 + b_0$$

# Decimal ↔ binary conversion

## Smaller numbers

- Find/sum the right powers of 2

$$11_{10} = 8 + 2 + 1 = \mathbf{1}\times2^3 + \mathbf{0}\times2^2 + \mathbf{1}\times2^1 + \mathbf{1}\times2^0 = 1011_2$$

## Useful powers of 2

- Bit values
- Ranges
- Sizes

| Up to 8 bits | | Up to 16 bits | | Over 16 bits | |
|---|---|---|---|---|---|
| $2^0$ | 1 | $2^8$ | 256 | $2^{16}$ | 65536 = 64 Ki |
| $2^1$ | 2 | $2^9$ | 512 | $\mathbf{2^{20}}$ | **1 Mi ≈ $10^6$** |
| $2^2$ | 4 | $\mathbf{2^{10}}$ | **1024 (1 Ki ≈ $10^3$)** | $2^{24}$ | 16 Mi |
| $2^3$ | 8 | $2^{11}$ | 2048 (2 Ki) | $\mathbf{2^{30}}$ | **1 Gi ≈ $10^9$** |
| $2^4$ | 16 | $2^{12}$ | 4096 (4 Ki) | $2^{32}$ | 4 Gi |
| $2^5$ | 32 | $2^{13}$ | 8192 (8 Ki) | $\mathbf{2^{40}}$ | **1 Ti ≈ $10^{12}$** |
| $2^6$ | 64 | $2^{14}$ | 16384 (16 Ki) | $\mathbf{2^{50}}$ | **1 Pi ≈ $10^{15}$** |
| $2^7$ | 128 | $2^{15}$ | 32768 (32 Ki) | $\mathbf{2^{60}}$ | **1 Ei ≈ $10^{18}$** |

# Decimal ↔ binary conversion

## Larger numbers

- Avoid decimal, use hexadecimal (base 16)

## Simple algorithm

- Divide number by 2 (integer division)
- Remainder provides next bit, starting with LSB
- Repeat until quotient is zero
- Note: Works in any positional system with a single base

$151_{10} = ???_2$

```
151 : 2 = 75 (1 = b0)
75  : 2 = 37 (1 = b1)
37  : 2 = 18 (1 = b2)
18  : 2 =  9 (0 = b3)
9   : 2 =  4 (1 = b4)
4   : 2 =  2 (0 = b5)
2   : 2 =  1 (0 = b6)
1   : 2 =  0 (1 = b7)
```

$151_{10} = 10010111_2$

# Hexadecimal ↔ binary conversion

## Convert 4-bit groups using a "lookup table"

- Preferably "stored" in your head.
- For bin ↔ hex start from LSB.
  - Pad with zero bits if the leftmost group has less than 4 bits.

| Dec | Bin | Hex | Dec | Bin | Hex |
|---|---|---|---|---|---|
| 0 | 0000 | 0 | 8 | 1000 | 8 |
| 1 | 0001 | 1 | 9 | 1001 | 9 |
| 2 | 0010 | 2 | **10** | 1010 | **A** |
| 3 | 0011 | 3 | **11** | 1011 | **B** |
| 4 | 0100 | 4 | **12** | 1100 | **C** |
| 5 | 0101 | 5 | **13** | 1101 | **D** |
| 6 | 0110 | 6 | **14** | 1110 | **E** |
| 7 | 0111 | 7 | **15** | 1111 | **F** |

$\mathtt{CAFEBABE}_{16} = \mathtt{11001010111111101011101010111110}_2$

$\mathtt{11011110101011011111000000000001}_2 = \mathtt{DEADF001}_{16}$

# How processor works with numbers

**Basic operations on numbers**

- Arithmetic (later), bitwise logical
- Bit shifts and rotations

**Operands stored in registers**

- Numbered places inside the processor
  - Operands “going into” operations
  - Results “coming out” from operations
- Register size is always fixed
  - Determines how large numbers can the processor operate on efficiently
  - General purpose registers: 8, 16, 32, 64 bits
  - Special purpose registers: 128, …, 512 bits

# What if the register size does not fit

**Register too small to hold a number**

- Holding 12-bit number in 8-bit registers
- Store part of the number in another register

**Register “too big” to hold a number**

- Holding 4-bit number in 8-bit register
- Ignore the irrelevant bits
- Use the “free” bits to store another number

**We need operations to “slice’n’dice” the bits**

- Bitwise logical operations, shifts, rotations

# Bitwise logical operations

## Based on basic boolean functions

| NOT | |
|---|---|
| 0 | 1 |
| 1 | 0 |

| AND | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| OR | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| XOR | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

## Applied to individual bits of the operands

- Pairwise between bits at the same position

## Engineering interpretation

- NOT ~ flipping all bits
- AND ~ clearing selected bits (masking)
- OR ~ setting selected bits
- XOR ~ flipping selected bits

# Bit shift operations

**Shift Logical Left/Right**

- Shift all bits of the operand by **n** positions
- Insert zero bits in the "vacated" places

**Alternative interpretation**

- Multiply (shift left) or divide (shift right) by 2